Министерство образования Республики Беларусь
Учреждение образование
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра интеллектуальных информационных технологий

В.В. Голенков, Р.Е. Сердюков

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ В ГРАФОДИНАМИЧЕСКИХ АССОЦИАТИВНЫХ МАШИНАХ

Учебное пособие по курсу
Операционные системы традиционных и интеллектуальных компьютеров
для студентов специальности I-40 03 01 «Искусственный интеллект» всех
форм обучения

Минск 2006

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

УДК 004.41 (075.8)
ББК 32.973.26-018.19 73
Г 60

Рецензент:
главный конструктор ИП РейнбоуТехнолоджи канд. техн. наук Ю.Г. Приходько

Голенков В.В.
Г60 Системное программирование в графодинамических ассоциативных машинах: Учеб. пособие по курсу «Операционные системы традиционных и интеллектуальных компьютеров» для студ. спец. I-40 03 01 «Искусственный интеллект» всех форм обуч. — Мн.: БГУИР, 2006. — 124 с.: ил.
ISBN 985-44-939-4

В учебном пособии рассматриваются вопросы архитектуры операционных систем традиционных компьютеров и компьютеров, специально ориентированных на обработку знаний, разновидностью которых являются графодинамические ассоциативные машины. Пособие предназначено для студентов специальности «Искусственный интеллект» всех форм обучения.

ISBN 985-444-939-4

© Голенков В.В., Сердюков
© БГУИР, 2006

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Содержание

Предисловие

Введение

1. Основные сведения об операционных системах

1.1.Определение операционной системы
1.2.Классификация операционных систем
1.3.Особенности аппаратных платформ
1.4.Состав и функционирование ОС
1.5.Контрольные вопросы

2. Управления локальными ресурсами

2.1.Управление процессами
2.2.Средства синхронизации и взаимодействия процессов
2.3.Управление памятью
2.4.Управление вводом-выводом
2.5.Файловая система
2.6.Контрольные вопросы

3. Управление распределенными ресурсами

3.1.Базовые примитивы передачи сообщений в распределенных системах
3.2.Вызов удаленных процедур (RPC)
3.3.Синхронизация в распределенных системах
3.4.Процессы и нити в распределенных системах
3.5.Распределенные файловые системы
3.6.Контрольные вопросы

4. Операционная среда графодинамических ассоциативных машин

4.1.Графодинамические ассоциативные машины
4.2.Архитектура операционной среды графодинамических ассоциативных машин
4.3.Подсистема управления памятью графодинамических ассоциативных машин
4.4.Подсистема управления процессами в графодинамических ассоциативных машинах
4.5.Подсистема управления внешними устройствами графодинамических ассоциативных машин
4.6.Навигационно-поисковая подсистема графодинамической ассоциативной машины

5. Базовый язык программирования для графодинамических ассоциативных машин

5.1.Принципы языка SCP (Semantic Code Programming)
5.2.Описание ядра языка SCP

Литература

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Предисловие

Данное пособие написано по материалам лекций и лабораторных работ, проводимых в рамках учебного курса «Операционные системы традиционных и интеллектуальных компьютеров» для студентов БГУИР, с необходимыми поправками и дополнениями. Оно может быть использовано для проведения всех форм учебных занятий по соответствующему учебному курсу, а также при выполнении курсовых и дипломных проектов и работ в рамках учебного плана специальности «Искусственный интеллект».

Автор выражает благодарность рецензентам, а также всем сотрудникам, аспирантам и студентам кафедры интеллектуальных информационных технологий БГУИР, принимавшим участие в работах по созданию, реализации и применению графодинамических моделей решения интеллектуальных задач.

Принятые обозначения

В работе для лучшего восприятия текста используются специальные обозначения.

Текст разбивается на нумеруемые разделы, подразделы и пункты. Их наименования выделяются жирным шрифтом разного размера. Для удобства восприятия фрагменты текста, имеющие свои заголовки, выделены **р а з р е ж е н н ы м ж и р н ы м ш р и ф т о м**. Такими заголовками выделяются: определения, утверждения, пояснения, доказательства, тексты формальных языков, рисунки, списки ключевых понятий, операции, микропрограммы и др. Указанные фрагменты текста могут иметь номер, который начинается с номера соответствующего раздела, подраздела или пункта. Второстепенные примечания в тексте выделяются мелким шрифтом.

В начале подразделов и пунктов приводится список ключевых понятий, рассматриваемых в данном подразделе или пункте. В списке ключевых понятий и в той части текста, где это ключевое понятие определяется, оно выделяется жирным шрифтом.

Наиболее важные фразы, на которые рекомендуется обратить особое внимание, выделяются подчеркиванием.

В пособии приводятся подробные примеры и иллюстрации. При этом формальные тексты (т.е. тексты приводимых в пособии формальных языков) выделяются жирным курсивом и иногда для наглядности заключаются в "рамочки".

Приводимые в тексте библиографические ссылки помимо порядкового номера в списке литературы содержат также (в круглых скобках) краткие

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

идентификаторы этих ссылок, в которых указываются фамилия автора, год издания и аббревиатура названия статьи или книги. Например, ссылка (***Поспелов Д.А. 1986кн-СитуаУ***) означает, что речь идет о книге, автор которой Поспелов Д.А., издана книга в 1986 г. и ее наименование «Ситуационное управление: Теория и практика».

При описании ссылок на учебные дисциплины специальности «Искусственный интеллект», которые связаны с рассматриваемой в данном пособии дисциплиной, указывается порядковый номер данной дисциплины в учебном плане специальности, а в скобках дополнительно указывается идентификатор соответствующей дисциплины. Например, идентификатор дисциплины «Математические основы искусственного интеллекта» имеет вид ***“УчДисц2-МОИИ”***. Кроме того, в тексте используются различные шрифты для всевозможных разделителей, таких, как точка, запятая, двоеточие, скобки и т.п.. Различие способов отображения разделителей продиктовано тем фактом, что описываемые формальные языки также содержат разделители, которые должны отличаться от используемых в обычном естественно-языковом тексте. В частности, используются три вида кавычек: "прямые" симметричные кавычки"**...**"– для выделения всевозможных идиоматичных высказываний; асимметричные жирные кавычки**“...”**– для явного выделения (если это требуется в данном контексте) фрагментов формального текста описываемых языков; угловые кавычки «**...**» – для выделения всевозможных наименований или цитат.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Введение

Расширение областей применения методов и средств искусственного интеллекта выявило недостаточную приспособленность традиционных компьютеров с фон-неймановской архитектурой в качестве основы для построения интеллектуальных систем. Основой интеллектуальных систем являются машины обработки знаний, частными видами которых являются машины логического вывода, решатели задач, реализующие самые различные логические исчисления и модели решения задач. Следовательно, важнейшим направлением повышения эффективности интеллектуальных систем является совершенствование машин обработки знаний (реализуемых как аппаратно, так и программно) и технологий их проектирования. Со стороны разработчиков прикладных интеллектуальных систем предъявляется ряд требований к машинам обработки знаний, таких, как открытость, расширяемость, гибкость, безопасность и т.д.

В настоящее время отчетливо обозначился ряд тенденций развития интеллектуальных систем, требующих перехода к параллельным и распределенным технологиям на всех этапах их разработки и применения. Для реализации этих тенденций наиболее перспективными моделями представления знаний являются семантические сети (В.Н. Вагин, В.П. Гладун, В.С. Лозовский, И.П. Кузнецов, Г.С. Плесневич, Ю.А. Загорулько, П.С. Сапатый и др.). Поэтому актуальным является развитие технологий проектирования машин обработки знаний, представленных в виде семантических сетей. Наиболее перспективным видом таких машин, удовлетворяющих указанным выше требованиям, являются предложенные в работах В.В. Голенкова графодинамические ассоциативные машины, в основе которых лежит ориентация на использование однородных семантических сетей, имеющих базовую теоретико-множественную интерпретацию, обладающих единством языка и метаязыка, обеспечивающих представление сложноструктурированных знаний.

Однако для машин обработки знаний, и в частности для графодинамических ассоциативных машин, в настоящие время недостаточно исследованными являются структура, методы и средства проектирования их системного программного обеспечения. Очевидно, что системное программное обеспечения машин обработки знаний имеет существенные отличия от системного программного обеспечения традиционных компьютеров. В состав системного программного обеспечения машин обработки знаний входят:

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

- Операционная система – средства эффективного управления ресурсами прикладных интеллектуальных систем, предоставляющие удобные и расширенные возможности по взаимодействию с ресурсами прикладных интеллектуальных систем.
- Пользовательский интерфейс – средства отображения информации и взаимодействия с пользователем.
- Базовая система программирования – средства разработки программ, в том числе средства их отладки и верификации.

Основной целью данного учебного пособия является рассмотрение вопросов построения операционных систем для традиционных и графодинамических ассоциативных машин, а также системного программирования для таких машин.

Курс “Операционные системы традиционных и интеллектуальных компьютеров” базируется на курсе “Основы организации и функционирования традиционных и интеллектуальных компьютеров”, а также на курсе “Конструирование программ, языки программирования в традицонных и интеллектуальных компьютерах”.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

1. Основные сведения об операционных системах

1.1. Определение операционной системы

Определение 1.1. Операционная система — это программа, контролирующая работу прикладных программ и системных приложений и исполняющая роль интерфейса между приложениями и аппаратным обеспечением компьютера. Ее предназначение можно разделить на три основные составляющие:

- удобство. Операционная система делает использование компьютера простым и удобным;
- эффективность. Операционная система позволяет эффективно использовать ресурсы компьютерной системы;
- возможность развития. Операционная система должна быть организована так, чтобы она допускала эффективную разработку, тестирование и внедрение новых приложений и системных функций, причем это не должно мешать нормальному функционированию вычислительной системы.

1.2. Классификация операционных систем

1.2.1. Поддержка многозадачности

По числу одновременно выполняемых задач операционные системы могут быть разделены на два класса:

- однозадачные (например, MS-DOS, MSX);
- многозадачные (ОС ЕС, OS/2, UNIX, Windows 95).

Однозадачные ОС в основном выполняют функцию предоставления пользователю виртуальной машины, делая более простым и удобным процесс взаимодействия пользователя с компьютером. Однозадачные ОС включают средства управления периферийными устройствами, средства управления файлами, средства общения с пользователем.

Многозадачные ОС, кроме вышеперечисленных функций, управляют разделением совместно используемых ресурсов, таких, как процессор, оперативная память, файлы и внешние устройства.

1.2.2. Поддержка многопользовательского режима

По числу одновременно работающих пользователей ОС делятся на следующие классы:

- однопользовательские (MS-DOS, Windows 3.x, ранние версии OS/2);

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

- многопользовательские (UNIX, Windows NT).

Главным отличием многопользовательских систем от однопользовательских является наличие средств защиты информации каждого пользователя от несанкционированного доступа других пользователей. Следует заметить, что не всякая многозадачная система является многопользовательской и не всякая однопользовательская ОС является однозадачной.

1.2.3. Вытесняющая и невытесняющая многозадачность

Важнейшим разделяемым ресурсом является процессорное время. Способ распределения процессорного времени между несколькими одновременно существующими в системе процессами (или нитями) во многом определяет специфику ОС. Среди множества существующих вариантов реализации многозадачности можно выделить две группы алгоритмов:
- невытесняющая многозадачность (NetWare, Windows 3.x);
- вытесняющая многозадачность (Windows NT, OS/2, UNIX).

Основным различием между вытесняющим и невытесняющим вариантами многозадачности является степень централизации механизма планирования процессов. В первом случае механизм планирования процессов целиком сосредоточен в операционной системе, а во втором – распределен между системой и прикладными программами. При невытесняющей многозадачности активный процесс выполняется до тех пор, пока он сам, по собственной инициативе, не отдаст управление операционной системе для того, чтобы та выбрала из очереди другой готовый к выполнению процесс. При вытесняющей многозадачности решение о переключении процессора с одного процесса на другой принимается операционной системой, а не самим активным процессом.

1.2.4. Поддержка многонитевости

Важным свойством операционных систем является возможность распараллеливания вычислений в рамках одной задачи. Многонитевая ОС разделяет процессорное время не между задачами, а между их отдельными ветвями (нитями).

1.2.5. Многопроцессорная обработка

Другим важным свойством ОС является отсутствие или наличие в ней средств поддержки многопроцессорной обработки — мультипроцессирование. Мультипроцессирование приводит к усложнению всех алгоритмов управления ресурсами.

В наши дни становится общепринятым введение в ОС функций поддержки многопроцессорной обработки данных. Такие функции имеются в операционных системах Solaris 2.x фирмы Sun, Open Server 3.x компании

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Santa Crus Operations, OS/2 фирмы IBM, Windows NT фирмы Microsoft и NetWare 4.1 фирмы Novell.

Многопроцессорные ОС могут классифицироваться по способу организации вычислительного процесса в системе с многопроцессорной архитектурой: асимметричные ОС и симметричные ОС. Асимметричная ОС целиком выполняется только на одном из процессоров системы, распределяя прикладные задачи по остальным процессорам. Симметричная ОС полностью децентрализована и использует весь пул процессоров, разделяя их между системными и прикладными задачами.

Выше были рассмотрены характеристики ОС, связанные с управлением только одним типом ресурсов — процессором. Важное влияние на облик операционной системы в целом, на возможности ее использования в той или иной области оказывают особенности и других подсистем управления локальными ресурсами — подсистем управления памятью, файлами, устройствами ввода-вывода.

Специфика ОС проявляется и в том, каким образом она реализует сетевые функции: распознавание и перенаправление в сеть запросов к удаленным ресурсам, передача сообщений по сети, выполнение удаленных запросов. При реализации сетевых функций возникает комплекс задач, связанных с распределенным характером хранения и обработки данных в сети: ведение справочной информации обо всех доступных в сети ресурсах и серверах, адресация взаимодействующих процессов, обеспечение прозрачности доступа, тиражирование данных, согласование копий, поддержка безопасности данных.

1.3. Особенности аппаратных платформ

На свойства операционной системы непосредственное влияние оказывают аппаратные средства, на которые она ориентирована. По типу аппаратуры различают операционные системы персональных компьютеров, мини-компьютеров, мейнфреймов, кластеров и сетей ЭВМ. Среди перечисленных типов компьютеров могут встречаться как однопроцессорные варианты, так и многопроцессорные. В любом случае специфика аппаратных средств, как правило, отражается на специфике операционных систем.

Очевидно, что ОС большой машины является более сложной и функциональной, чем ОС персонального компьютера. Так, в ОС больших машин функции по планированию потока выполняемых задач, очевидно, реализуются путем использования сложных приоритетных дисциплин и требуют большей вычислительной мощности, чем в ОС персональных компьютеров. Аналогично обстоит дело и с другими функциями.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Сетевая ОС имеет в своем составе средства передачи сообщений между компьютерами по линиям связи, которые совершенно не нужны в автономной ОС. На основе этих сообщений сетевая ОС поддерживает разделение ресурсов компьютера между удаленными пользователями, подключенными к сети. Для поддержания функций передачи сообщений сетевые ОС содержат специальные программные компоненты, реализующие популярные коммуникационные протоколы, такие, как IP, IPX, Ethernet и др.

Многопроцессорные комплексы требуют от операционной системы особой организации, с помощью которой она сама, а также поддерживаемые ею приложения могли бы выполняться параллельно отдельными процессорами системы. Параллельная работа отдельных частей ОС создает дополнительные проблемы для разработчиков ОС, так как в этом случае гораздо сложнее обеспечить согласованный доступ отдельных процессов к общим системным таблицам, исключить эффект гонок и прочие нежелательные последствия асинхронного выполнения работ.

Другие требования предъявляются к операционным системам кластеров. Кластер — слабо связанная совокупность нескольких вычислительных систем, работающих совместно для выполнения общих приложений, и представляющихся пользователю единой системой. Наряду со специальной аппаратурой для функционирования кластерных систем необходима и программная поддержка со стороны операционной системы, которая сводится в основном к синхронизации доступа к разделяемым ресурсам, обнаружению отказов и динамической реконфигурации системы. Одной из первых разработок в области кластерных технологий были решения компании Digital Equipment на базе компьютеров VAX. Недавно этой компанией заключено соглашение с корпорацией Microsoft о разработке кластерной технологии, использующей Windows NT. Несколько компаний предлагают кластеры на основе UNIX-машин.

Наряду с ОС, ориентированными на совершенно определенный тип аппаратной платформы, существуют операционные системы, специально разработанные таким образом, чтобы они могли быть легко перенесены с компьютера одного типа на компьютер другого типа — так называемые *мобильные* ОС. Наиболее ярким примером такой ОС является популярная система UNIX. В этих системах аппаратно-зависимые места тщательно локализованы, так что при переносе системы на новую платформу переписываются только они. Средством, облегчающем перенос остальной части ОС, является написание ее на машинно-независимом языке, например на С, который и был разработан для программирования операционных систем.

1.3.1. Особенности областей использования

Многозадачные ОС подразделяются на три типа в соответствии с использованными при их разработке критериями эффективности:

- <u>системы пакетной обработки</u> (например ОС ЕС);

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

- системы разделения времени (UNIX, VMS);
- системы реального времени (QNX, RT/11).

Системы пакетной обработки предназначались для решения задач в основном вычислительного характера, не требующих быстрого получения результатов. Главной целью и критерием эффективности систем пакетной обработки является максимальная пропускная способность, то есть решение максимального числа задач в единицу времени. Для достижения этой цели в системах пакетной обработки используется следующая схема функционирования: в начале работы формируется пакет заданий, каждое задание содержит требование к системным ресурсам; из этого пакета заданий формируется мультипрограммная смесь, то есть множество одновременно выполняемых задач. Для одновременного выполнения выбираются задачи, предъявляющие отличающиеся требования к ресурсам, так чтобы обеспечивалась сбалансированная загрузка всех устройств вычислительной машины; например, в мультипрограммной смеси желательно одновременное присутствие вычислительных задач и задач с интенсивным вводом-выводом. Таким образом, выбор нового задания из пакета заданий зависит от внутренней ситуации, складывающейся в системе, то есть выбирается "выгодное" задание. Следовательно, в таких ОС невозможно гарантировать выполнение того или иного задания в течение определенного периода времени. В системах пакетной обработки переключение процессора с выполнения одной задачи на выполнение другой происходит только в случае, если активная задача сама отказывается от процессора, например из-за необходимости выполнить операцию ввода-вывода. Поэтому одна задача может надолго занять процессор, что делает невозможным выполнение интерактивных задач. Таким образом, взаимодействие пользователя с вычислительной машиной, на которой установлена система пакетной обработки, сводится к тому, что он приносит задание, отдает его диспетчеру-оператору, а в конце дня после выполнения всего пакета заданий получает результат. Очевидно, что такой порядок снижает эффективность работы пользователя.

Системы разделения времени призваны исправить основной недостаток систем пакетной обработки – изоляцию пользователя-программиста от процесса выполнения его задач. Каждому пользователю системы разделения времени предоставляется терминал, с которого он может вести диалог со своей программой. Так как в системах разделения времени каждой задаче выделяется только квант процессорного времени, ни одна задача не занимает процессор надолго, и время ответа оказывается приемлемым. Если квант выбран достаточно небольшим, то у всех пользователей, одновременно работающих на одной и той же машине, складывается впечатление, что каждый из них единолично использует машину. Ясно, что системы разделения времени обладают меньшей пропускной способностью, чем системы пакетной обработки, так как на выполнение принимается каждая запущенная пользователем задача, а не та, которая "выгодна" системе, и, кроме того, имеются накладные расходы

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

вычислительной мощности на более частое переключение процессора с задачи на задачу. Критерием эффективности систем разделения времени является не максимальная пропускная способность, а удобство и эффективность работы пользователя.

Системы реального времени применяются для управления различными техническими объектами, такими, например, как станок, спутник, научная экспериментальная установка, или технологическими процессами, такими, как гальваническая линия, доменный процесс и т.п. Во всех этих случаях существует предельно допустимое время, в течение которого должна быть выполнена та или иная программа, управляющая объектом, в противном случае может произойти авария: спутник выйдет из зоны видимости, экспериментальные данные, поступающие с датчиков, будут потеряны, толщина гальванического покрытия не будет соответствовать норме. Таким образом, критерием эффективности для систем реального времени является их способность выдерживать заранее заданные интервалы времени между запуском программы и получением результата (управляющего воздействия). Это время называется временем реакции системы, а соответствующее свойство системы – реактивностью. Для этих систем мультипрограммная смесь представляет собой фиксированный набор заранее разработанных программ, а выбор программы на выполнение осуществляется исходя из текущего состояния объекта или в соответствии с расписанием плановых работ.

Некоторые операционные системы могут совмещать в себе свойства систем разных типов, например, часть задач может выполняться в режиме пакетной обработки, а часть – в режиме реального времени или в режиме разделения времени. В таких случаях режим пакетной обработки часто называют фоновым режимом.

1.3.2. Особенности методов построения

При описании операционной системы часто указываются особенности ее структурной организации и основные концепции, положенные в ее основу.

К таким базовым концепциям относятся:

- способы построения ядра системы – монолитное ядро или микроядерный подход. Большинство ОС использует монолитное ядро, которое компонуется как одна программа, работающая в привилегированном режиме и использующая быстрые переходы с одной процедуры на другую, не требующие переключения из привилегированного режима в пользовательский и наоборот. Альтернативой является построение ОС на базе микроядра, работающего также в привилегированном режиме и выполняющего только минимум функций по управлению аппаратурой, в то время как функции ОС более высокого уровня выполняют специализированные компоненты ОС – серверы, работающие в пользовательском режиме. При таком построении ОС работает более медленно, так как часто выполняются переходы между привилегированным режимом и пользовательским, зато система

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

получается более гибкой – ее функции можно наращивать, модифицировать или сужать, добавляя, модифицируя или исключая серверы пользовательского режима. Кроме того, серверы хорошо защищены друг от друга, как и любые пользовательские процессы;

- построение ОС на базе объектно-ориентированного подхода дает возможность использовать все его достоинства, хорошо зарекомендовавшие себя на уровне приложений, внутри операционной системы, а именно: аккумуляцию удачных решений в форме стандартных объектов, возможность создания новых объектов на базе имеющихся с помощью механизма наследования, хорошую защиту данных за счет их инкапсуляции во внутренние структуры объекта, что делает данные недоступными для несанкционированного использования извне, структуризованность системы, состоящей из набора хорошо определенных объектов;
- наличие нескольких прикладных сред дает возможность в рамках одной ОС одновременно выполнять приложения, разработанные для нескольких ОС. Многие современные операционные системы поддерживают одновременно прикладные среды MS-DOS, Windows, UNIX (POSIX), OS/2 или хотя бы некоторого подмножества из этого популярного набора. Концепция множественных прикладных сред наиболее просто реализуется в ОС на базе микроядра, над которым работают различные серверы, часть которых реализуют прикладную среду той или иной операционной системы;
- распределенная организация операционной системы позволяет упростить работу пользователей и программистов в сетевых средах. В распределенной ОС реализованы механизмы, которые дают возможность пользователю представлять и воспринимать сеть в виде традиционного однопроцессорного компьютера. Характерными признаками распределенной организации ОС являются: наличие единой справочной службы разделяемых ресурсов, единой службы времени, использование механизма вызова удаленных процедур (RPC) для прозрачного распределения программных процедур по машинам, многонитевой обработки, позволяющей распараллеливать вычисления в рамках одной задачи и выполнять эту задачу сразу на нескольких компьютерах сети, а также наличие других распределенных служб.

1.4. Состав и функционирование ОС

1.4.1. Структура сетевой операционной системы

Сетевая операционная система составляет основу любой вычислительной сети. Каждый компьютер в сети в значительной степени автономен, поэтому под сетевой операционной системой в широком смысле понимается совокупность операционных систем отдельных компьютеров, взаимодействующих с целью обмена сообщениями и разделения ресурсов по единым правилам — протоколам. В узком смысле сетевая ОС — это

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

операционная система отдельного компьютера, обеспечивающая ему возможность работать в сети.

В сетевой операционной системе отдельной машины можно выделить несколько частей (рис. 1.1):

- средства управления локальными ресурсами компьютера: функции распределения оперативной памяти между процессами, планирования и диспетчеризации процессов, управления процессорами в мультипроцессорных машинах, управления периферийными устройствами и другие функции управления ресурсами локальных ОС;
- средства предоставления собственных ресурсов и услуг в общее пользование — серверная часть ОС (сервер). Эти средства обеспечивают, например, блокировку файлов и записей, что необходимо для их совместного использования; ведение справочников имен сетевых ресурсов; обработку запросов удаленного доступа к собственной файловой системе и базе данных; управление очередями запросов удаленных пользователей к своим периферийным устройствам;
- средства запроса доступа к удаленным ресурсам и услугам и их использования — клиентская часть ОС (редиректор). Эта часть выполняет распознавание и перенаправление в сеть запросов к удаленным ресурсам от приложений и пользователей, при этом запрос поступает от приложения в локальной форме, а передается в сеть в другой форме, соответствующей требованиям сервера. Клиентская часть также осуществляет прием ответов от серверов и преобразование их в локальный формат, так что для приложения выполнение локальных и удаленных запросов неразличимо;
- коммуникационные средства ОС, с помощью которых происходит обмен сообщениями в сети. Эта часть обеспечивает адресацию и буферизацию сообщений, выбор маршрута передачи сообщения по сети, надежность передачи и т.п., то есть является средством транспортировки сообщений.

Рис. 1.1. Структура сетевой ОС

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

В зависимости от функций, возлагаемых на конкретный компьютер, в его операционной системе может отсутствовать либо клиентская, либо серверная часть.

На рис. 1.2 показано взаимодействие сетевых компонентов. Здесь компьютер 1 выполняет роль "чистого" клиента, а компьютер 2 — роль "чистого" сервера, соответственно на первой машине отсутствует серверная часть, а на второй — клиентская. На рисунке отдельно показан компонент клиентской части — редиректор. Именно редиректор перехватывает все запросы, поступающие от приложений, и анализирует их. Если выдан запрос к ресурсу данного компьютера, то он переадресовывается соответствующей подсистеме локальной ОС, если же это запрос к удаленному ресурсу, то он переправляется в сеть. При этом клиентская часть преобразует запрос из локальной формы в сетевой формат и передает его транспортной подсистеме, которая отвечает за доставку сообщений указанному серверу. Серверная часть операционной системы компьютера 2 принимает запрос, преобразует его и передает для выполнения своей локальной ОС. После того как результат получен, сервер обращается к транспортной подсистеме и направляет ответ клиенту, выдавшему запрос. Клиентская часть преобразует результат в соответствующий формат и адресует его тому приложению, которое выдало запрос.

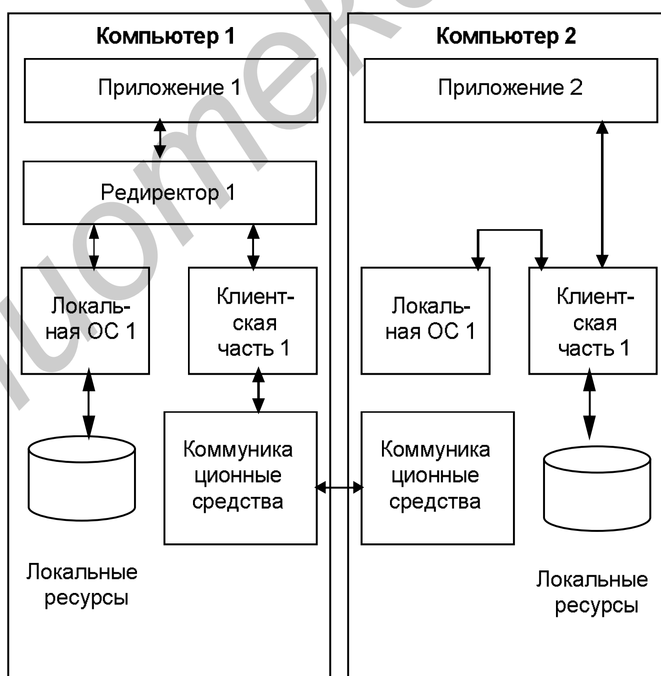


Рис. 1.2. Взаимодействие компонентов операционной системы при взаимодействии компьютеров

На практике сложилось несколько подходов к построению сетевых операционных систем (рис.1.3).

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

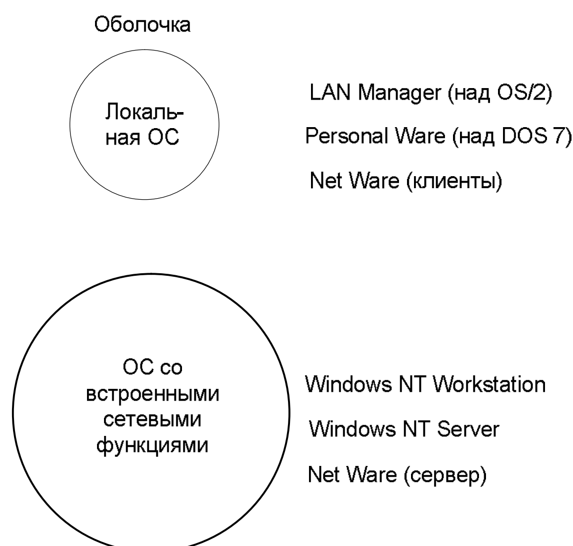


Рис. 1.3. Варианты построения сетевых ОС

Первые сетевые ОС представляли собой совокупность существующей локальной ОС и надстроенной над ней *сетевой оболочки*. При этом в локальную ОС встраивался минимум сетевых функций, необходимых для работы сетевой оболочки, которая выполняла основные сетевые функции. Примером такого подхода является использование на каждой машине сети операционной системы MS DOS (у которой начиная с ее третьей версии появились такие встроенные функции, как блокировка файлов и записей, необходимые для совместного доступа к файлам). Принцип построения сетевых ОС в виде сетевой оболочки над локальной ОС используется и в современных ОС, таких, например, как LANtastic или Personal Ware.

Однако более эффективным представляется путь разработки операционных систем, изначально предназначенных для работы в сети. Сетевые функции у ОС такого типа глубоко *встроены* в основные модули системы, что обеспечивает их логическую стройность, простоту эксплуатации и модификации, а также высокую производительность. Примером такой ОС является система Windows NT фирмы Microsoft, которая за счет встроенности сетевых средств обеспечивает более высокие показатели производительности и защищенности информации по сравнению с сетевой ОС LAN Manager той же фирмы (совместная разработка с IBM), являющейся надстройкой над локальной операционной системой OS/2.

1.4.2. Одноранговые сетевые ОС и ОС с выделенными серверами

В зависимости от того, как распределены функции между компьютерами сети, сетевые операционные системы, а следовательно, и сети делятся на два класса: одноранговые и двухранговые (рис. 1.4). Последние чаще называют сетями с выделенными серверами.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

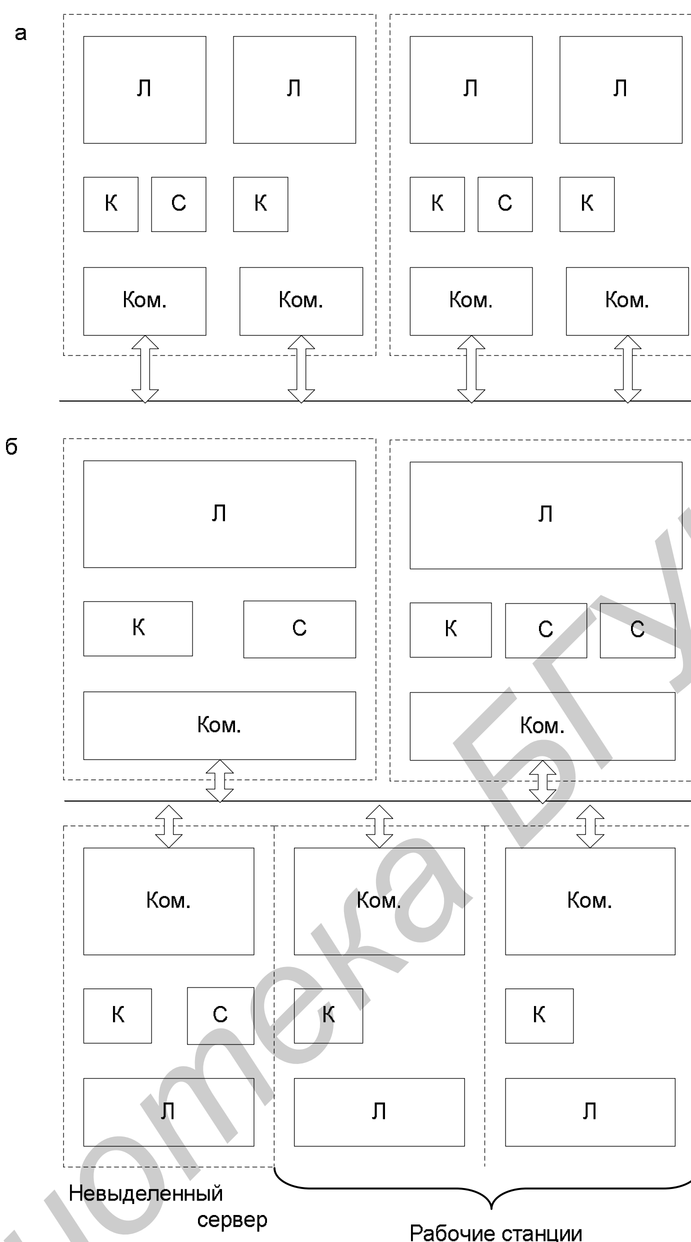


Рис. 1.4. Одноранговая (а) и двухранговая (б) сети

Если компьютер предоставляет свои ресурсы другим пользователям сети, то он играет роль сервера. При этом компьютер, обращающийся к ресурсам другой машины, является клиентом. Как уже было сказано, компьютер, работающий в сети, может выполнять функции либо клиента, либо сервера, либо совмещать обе эти функции.

Если выполнение каких-либо серверных функций является основным назначением компьютера (например, предоставление файлов в общее пользование всем остальным пользователям сети, или организация совместного использования факса, или предоставление всем пользователям сети возможности запуска на данном компьютере своих приложений), то такой компьютер называется выделенным сервером. В зависимости от того, какой ресурс сервера является разделяемым, он называется файл-сервером, факс-сервером, принт-сервером, сервером приложений и т.д.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Очевидно, что на выделенных серверах желательно устанавливать ОС, специально оптимизированные для выполнения тех или иных серверных функций. Поэтому в сетях с выделенными серверами чаще всего используются сетевые операционные системы, в состав которых входит несколько вариантов ОС, отличающихся возможностями серверных частей. Например, сетевая ОС Novell NetWare имеет серверный вариант, оптимизированный для работы в качестве файл-сервера, а также варианты оболочек для рабочих станций с различными локальными ОС, причем эти оболочки выполняют исключительно функции клиента. Другим примером ОС, ориентированной на построение сети с выделенным сервером, является операционная система Windows NT. В отличие от NetWare, оба варианта данной сетевой ОС — Windows NT Server (для выделенного сервера) и Windows NT Workstation (для рабочей станции) — могут поддерживать функции и клиента и сервера. Но серверный вариант Windows NT имеет больше возможностей для предоставления ресурсов своего компьютера другим пользователям сети, так как может выполнять более широкий набор функций, поддерживает большее количество одновременных соединений с клиентами, реализует централизованное управление сетью, имеет более развитые средства защиты.

Выделенный сервер не принято использовать в качестве компьютера для выполнения текущих задач, не связанных с его основным назначением, так как это может уменьшить производительность его работы как сервера. В связи с такими соображениями в ОС Novell NetWare на серверной части возможность выполнения обычных прикладных программ вообще не предусмотрена, то есть сервер не содержит клиентской части, а на рабочих станциях отсутствуют серверные компоненты. Однако в других сетевых ОС функционирование на выделенном сервере клиентской части вполне возможно. Например, под управлением Windows NT Server могут запускаться обычные программы локального пользователя, которые могут потребовать выполнения клиентских функций ОС при появлении запросов к ресурсам других компьютеров сети. При этом рабочие станции, на которых установлена ОС Windows NT Workstation, могут выполнять функции невыделенного сервера.

Важно понять, что несмотря на то, что в сети с выделенным сервером все компьютеры в общем случае могут выполнять одновременно роли и сервера, и клиента, эта сеть функционально не симметрична: аппаратно и программно в ней реализованы два типа компьютеров: одни — в большей степени ориентированные на выполнение серверных функций и работающие под управлением специализированных серверных ОС, а другие — в основном выполняющие клиентские функции и работающие под управлением соответствующего этому назначению варианта ОС. Функциональная несимметричность, как правило, вызывает и несимметричность аппаратуры для выделенных серверов используются более мощные компьютеры с большими объемами оперативной и внешней памяти. Таким образом, функциональная несимметричность в сетях с выделенным сервером сопровождается несимметричностью операционных

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

систем (специализация ОС) и аппаратной несимметричностью (специализация компьютеров).

В одноранговых сетях все компьютеры равны в правах доступа к ресурсам друг друга. Каждый пользователь может по своему желанию объявить какой-либо ресурс своего компьютера разделяемым, после чего другие пользователи могут его эксплуатировать. В таких сетях на всех компьютерах устанавливается одна и та же ОС, которая предоставляет всем компьютерам в сети *потенциально* равные возможности. Одноранговые сети могут быть построены, например, на базе ОС LANtastic, Personal Ware, Windows for Workgroup, Windows NT Workstation.

В одноранговых сетях также может возникнуть функциональная несимметричность: одни пользователи не желают разделять свои ресурсы с другими, и в таком случае их компьютеры выполняют роль клиента, за другими компьютерами администратор закрепил только функции по организации совместного использования ресурсов, а значит, они являются серверами, в третьем случае, когда локальный пользователь не возражает против использования его ресурсов и сам не исключает возможности обращения к другим компьютерам, ОС, устанавливаемая на его компьютере, должна включать и серверную и клиентскую части. В отличие от сетей с выделенными серверами, в одноранговых сетях отсутствует специализация ОС в зависимости от преобладающей функциональной направленности — клиента или сервера. Все вариации реализуются средствами конфигурирования одного и того же варианта ОС.

Одноранговые сети проще в организации и эксплуатации, однако они применяются в основном для объединения небольших групп пользователей, не предъявляющих больших требований к объемам хранимой информации, ее защищенности от несанкционированного доступа и к скорости доступа. При повышенных требованиях к этим характеристикам более подходящими являются двухранговые сети, где сервер лучше решает задачу обслуживания пользователей своими ресурсами, так как его аппаратура и сетевая операционная система специально спроектированы для этой цели.

1.4.3. ОС для рабочих групп и ОС для сетей масштаба предприятия

Сетевые операционные системы имеют разные свойства в зависимости от того, предназначены они для сетей масштаба рабочей группы (отдела), для сетей масштаба кампуса или для сетей масштаба предприятия:

- *сети отделов* — используются небольшой группой сотрудников, решающих общие задачи. Главной целью сети отдела является разделение локальных ресурсов, таких, как приложения, данные, лазерные принтеры и модемы. Сети отделов обычно не разделяются на подсети;

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

- *сети кампусов* — соединяют несколько сетей отделов внутри отдельного здания или внутри одной территории предприятия. Эти сети являются все еще локальными сетями, хотя и могут покрывать территорию в несколько квадратных километров. Сервисы такой сети включают взаимодействие между сетями отделов, доступ к базам данных предприятия, доступ к факс-серверам, высокоскоростным модемам и высокоскоростным принтерам;
- *сети предприятия (корпоративные сети)* — объединяют все компьютеры всех территорий отдельного предприятия. Они могут покрывать город, регион или даже континент. В таких сетях пользователям предоставляется доступ к информации и приложениям, находящимся в других рабочих группах, других отделах, подразделениях и штаб-квартирах корпорации.

Главной задачей операционной системы, используемой в сети масштаба отдела, является организация разделения ресурсов, таких, как приложения, данные, лазерные принтеры и, возможно, низкоскоростные модемы. Обычно сети отделов имеют один или два файловых сервера и не более чем 30 пользователей. Задачи управления на уровне отдела относительно просты. В задачи администратора входит добавление новых пользователей, устранение простых отказов, инсталляция новых узлов и установка новых версий программного обеспечения. Операционные системы сетей отделов хорошо отработаны и разнообразны, так же, как и сами сети отделов, уже давно применяющиеся и достаточно отлаженные. Такая сеть обычно использует одну или максимум две сетевые ОС. Чаще всего это сеть с выделенным сервером NetWare 3.x или Windows NT, или же одноранговая сеть, например сеть Windows for Workgroups.

Пользователи и администраторы сетей отделов вскоре осознают, что они могут улучшить эффективность своей работы путем получения доступа к информации других отделов своего предприятия. Если сотрудник, занимающийся продажами, может получить доступ к характеристикам конкретного продукта и включить их в презентацию, то эта информация будет более свежей и будет оказывать большее влияние на покупателей. Если отдел маркетинга может получить доступ к характеристикам продукта, который еще только разрабатывается инженерным отделом, то он может быстро подготовить маркетинговые материалы сразу же после окончания разработки.

Итак, следующим шагом в эволюции сетей является объединение локальных сетей нескольких отделов в единую сеть здания или группы зданий. Такие сети называют сетями кампусов. Сети кампусов могут простираться на несколько километров, но при этом глобальные соединения не требуются.

Операционная система, работающая в сети кампуса, должна обеспечивать для сотрудников одних отделов доступ к некоторым файлам и ресурсам сетей других отделов. Услуги, предоставляемые ОС сетей кампусов, не ограничиваются простым разделением файлов и принтеров, а часто предоставляют доступ и к серверам других типов, например, к факс-

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

серверам и к серверам высокоскоростных модемов. Важным сервисом, предоставляемым операционными системами данного класса, является доступ к корпоративным базам данных, независимо от того, располагаются ли они на серверах баз данных или на миникомпьютерах.

Именно на уровне сети кампуса начинаются проблемы интеграции. В общем случае отделы уже выбрали для себя типы компьютеров, сетевого оборудования и сетевых операционных систем. Например, инженерный отдел может использовать операционную систему UNIX и сетевое оборудование Ethernet, отдел продаж может использовать операционные среды DOS/Novell и оборудование Token Ring. Очень часто сеть кампуса соединяет разнородные компьютерные системы, в то время как сети отделов используют однотипные компьютеры.

Корпоративная сеть соединяет сети всех подразделений предприятия, в общем случае находящихся на значительных расстояниях. Корпоративные сети используют глобальные связи (WAN links) для соединения локальных сетей или отдельных компьютеров.

Пользователям корпоративных сетей требуются все те приложения и услуги, которые имеются в сетях отделов и кампусов, плюс некоторые дополнительные приложения и услуги, например, доступ к приложениям мейнфреймов и миникомпьютеров и к глобальным связям. Когда ОС разрабатывается для локальной сети или рабочей группы, то ее главной обязанностью является разделение файлов и других сетевых ресурсов (обычно принтеров) между локально подключенными пользователями. Такой подход неприменим для уровня предприятия. Наряду с базовыми сервисами, связанными с разделением файлов и принтеров, сетевая ОС, которая разрабатывается для корпораций, должна поддерживать более широкий набор сервисов, в который обычно входят почтовая служба, средства коллективной работы, поддержка удаленных пользователей, факс-сервис, обработка голосовых сообщений, организация видеоконференций и др.

Кроме того, многие существующие методы и подходы к решению традиционных задач сетей меньших масштабов для корпоративной сети оказались непригодными. На первый план вышли такие задачи и проблемы, которые в сетях рабочих групп, отделов и даже кампусов либо имели второстепенное значение, либо вообще не проявлялись. Например, простейшая для небольшой сети задача ведения учетной информации о пользователях выросла в сложную проблему для сети масштаба предприятия. А использование глобальных связей требует от корпоративных ОС поддержки протоколов, хорошо работающих на низкоскоростных линиях, и отказа от некоторых традиционно используемых протоколов (например, тех, которые активно используют широковещательные сообщения). Особое значение приобрели задачи преодоления гетерогенности — в сети появились многочисленные шлюзы,

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

обеспечивающие согласованную работу различных ОС и сетевых системных приложений.

К признакам корпоративных ОС могут быть отнесены также следующие особенности.

Поддержка приложений. В корпоративных сетях выполняются сложные приложения, требующие для выполнения большой вычислительной мощности. Такие приложения разделяются на несколько частей, например, на одном компьютере выполняется часть приложения, связанная с выполнением запросов к базе данных, на другом — запросов к файловому сервису, а на клиентских машинах — часть, реализующая логику обработки данных приложения и организующая интерфейс с пользователем. Вычислительная часть общих для корпорации программных систем может быть слишком объемной и неподъемной для рабочих станций клиентов, поэтому приложения будут выполняться более эффективно, если их наиболее сложные в вычислительном отношении части перенести на специально предназначенный для этого мощный компьютер — *сервер приложений.*

Сервер приложений должен базироваться на мощной аппаратной платформе (мультипроцессорные системы, часто на базе RISC-процессоров, специализированные кластерные архитектуры). ОС сервера приложений должна обеспечивать высокую производительность вычислений, а значит, поддерживать многонитевую обработку, вытесняющую многозадачность, мультипроцессирование, виртуальную память и наиболее популярные прикладные среды (UNIX, Windows, MS-DOS, OS/2). В этом отношении сетевую ОС NetWare трудно отнести к корпоративным продуктам, так как в ней отсутствуют почти все требования, предъявляемые к серверу приложений. В то же время хорошая поддержка универсальных приложений в Windows NT, собственно, и позволяет ей претендовать на место в мире корпоративных продуктов.

Справочная служба. Корпоративная ОС должна обладать способностью хранить информацию обо всех пользователях и ресурсах таким образом, чтобы обеспечивалось управление ею из одной центральной точки. Подобно большой организации, корпоративная сеть нуждается в централизованном хранении как можно более полной справочной информации о самой себе (начиная с данных о пользователях, серверах, рабочих станциях и кончая данными о кабельной системе). Естественно организовать эту информацию в виде базы данных. Данные из этой базы могут быть востребованы многими сетевыми системными приложениями, в первую очередь системами управления и администрирования. Кроме этого, такая база полезна при организации электронной почты, систем коллективной работы, службы безопасности, службы инвентаризации программного и аппаратного обеспечения сети, да и для практически любого крупного бизнес-приложения.

База данных, хранящая справочную информацию, предоставляет все то же многообразие возможностей и порождает все то же множество проблем,

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

что и любая другая крупная база данных. Она позволяет осуществлять различные операции поиска, сортировки, модификации и т.п., что очень сильно облегчает жизнь как администраторам, так и пользователям. Но за эти удобства приходится расплачиваться решением проблем распределенности, репликации и синхронизации.

В идеале сетевая справочная информация должна быть реализована в виде единой базы данных, а не представлять собой набор баз данных, специализирующихся на хранении информации того или иного вида, как это часто бывает в реальных операционных системах. Например, в Windows NT имеется по крайней мере пять различных типов справочных баз данных. Главный справочник домена (NT Domain Directory Service) хранит информацию о пользователях, которая используется при организации их логического входа в сеть. Данные о тех же пользователях могут содержаться и в другом справочнике, используемом электронной почтой Microsoft Mail. Еще три базы данных поддерживают разрешение низкоуровневых адресов: WINS — устанавливает соответствие NETBIOS-имен IP-адресам, справочник DNS оказывается полезным при подключении NT-сети к Internet и, наконец, справочник протокола DHCP используется для автоматического назначения IP-адресов компьютерам сети. Ближе к идеалу находятся справочные службы, поставляемые фирмой Banyan (продукт Streettalk III) и фирмой Novell (NetWare Directory Services), предлагающие единый справочник для всех сетевых приложений. Наличие единой справочной службы для сетевой операционной системы — один из важнейших признаков ее корпоративности.

Безопасность. Особую важность для ОС корпоративной сети приобретают вопросы безопасности данных. С одной стороны, в крупномасштабной сети объективно существует больше возможностей для несанкционированного доступа — из-за децентрализации данных и большой распределенности "законных" точек доступа, из-за большого числа пользователей, благонадежность которых трудно установить, а также из-за большого числа возможных точек несанкционированного подключения к сети. С другой стороны, корпоративные бизнес-приложения работают с данными, которые имеют жизненно важное значение для успешной работы корпорации в целом. И для защиты таких данных в корпоративных сетях наряду с различными аппаратными средствами используется весь спектр средств защиты, предоставляемый операционной системой: избирательные или мандатные права доступа, сложные процедуры аутентификации пользователей, программная шифрация.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

1.5. Контрольные вопросы

1. Может ли компьютер работать без операционной системы?
2. Может ли многопользовательская операционная система быть однозадачной?
3. Может ли однозадачная операционная система быть многонитевой?
4. Всегда ли многопроцессорные операционные системы являются многозадачными?
5. В чем отличие операционных систем пакетной обработки от операционных систем разделения времени?
6. Может ли операционная система разделения времени быть однозадачной?
7. В чем отличие операционных систем реального времени от операционных систем разделения времени?

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

2. Управление локальными ресурсами

Важнейшей функцией операционной системы является организация рационального использования всех аппаратных и программных ресурсов системы. К основным ресурсам могут быть отнесены: процессоры, память, внешние устройства, данные и программы. Располагающая одними и теми же ресурсами, но управляемая различными ОС вычислительная система может работать с разной степенью эффективности. Поэтому знание внутренних механизмов операционной системы позволяет косвенно судить о ее эксплуатационных возможностях и характеристиках.

2.1. Управление процессами

Важнейшей частью операционной системы, непосредственно влияющей на функционирование вычислительной машины, является подсистема управления процессами.

Определение 2.1. Процесс – абстракция, описывающая выполняющуюся программу. Для операционной системы процесс представляет собой единицу работы, заявку на потребление системных ресурсов.

Подсистема управления процессами планирует выполнение процессов, то есть распределяет процессорное время между несколькими одновременно существующими в системе процессами, а также занимается созданием и уничтожением процессов, обеспечивает процессы необходимыми системными ресурсами, поддерживает взаимодействие между процессами.

2.1.1. Состояние процессов

В многозадачной (многопроцессной) системе процесс может находиться в одном из трех основных состояний:

- выполнение — активное состояние процесса, во время которого процесс обладает всеми необходимыми ресурсами и непосредственно выполняется процессором;
- ожидание — пассивное состояние процесса, процесс заблокирован, он не может выполняться по своим внутренним причинам, а ждет осуществления некоторого события, например, завершения операции ввода-вывода, получения сообщения от другого процесса, освобождения какого-либо необходимого ему ресурса;
- готовность — также пассивное состояние процесса, но в этом случае процесс заблокирован в связи с внешними по отношению к нему обстоятельствами: процесс имеет все требуемые для него ресурсы, он готов выполняться, однако процессор занят выполнением другого процесса.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

В ходе жизненного цикла каждый процесс переходит из одного состояния в другое в соответствии с алгоритмом планирования процессов, реализуемым в данной операционной системе. Типичный граф состояний процесса показан на рис. 2.1.

В состоянии *выполнение* в однопроцессорной системе может находиться только один процесс, а в каждом из состояний *ожидание* и *готовность* — несколько процессов; эти процессы образуют очереди соответственно ожидающих и готовых процессов. Жизненный цикл процесса начинается с состояния *готовность*, когда процесс готов к выполнению и ждет своей очереди. При активизации процесс переходит в состояние *выполнение* и находится в нем до тех пор, пока либо он сам освободит процессор, перейдя в состояние *ожидания* какого-нибудь события, либо будет насильно "вытеснен" из процессора, например, вследствие исчерпания отведенного данному процессу кванта процессорного времени. В последнем случае процесс возвращается в состояние *готовность*. В это же состояние процесс переходит из состояния *ожидание*, после того как ожидаемое событие произойдет.

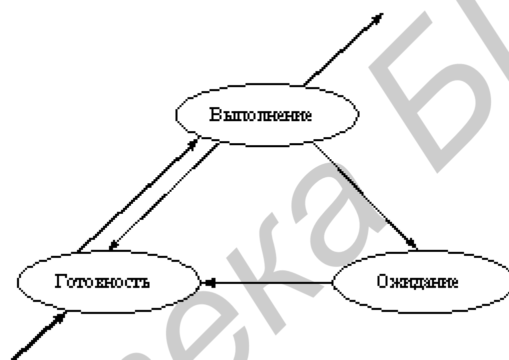


Рис. 2.1. Граф состояний процесса в многозадачной среде

2.1.2. Контекст и дескриптор процесса

На протяжении существования процесса его выполнение может быть многократно прервано и продолжено. Для того чтобы возобновить выполнение процесса, необходимо восстановить состояние его операционной среды.

Определение 2.2. Контекст процесса – это информация, отображающая состояние регистров и программного счетчика, режимом работы процессора, указателями на открытые файлы, информацией о незавершенных операциях ввода-вывода, кодами ошибок выполняемых данным процессом системных вызовов и т.д.

Кроме этого, операционной системе для реализации планирования процессов требуется дополнительная информация: идентификатор процесса, состояние процесса, данные о степени привилегированности процесса, местонахождение кодового сегмента и другая информация. В некоторых ОС (например, в ОС UNIX) информацию такого рода, используемую ОС для планирования процессов, называют дескриптором процесса.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Дескриптор процесса по сравнению с контекстом содержит более оперативную информацию, которая должна быть легко доступна подсистеме планирования процессов. Контекст процесса содержит менее актуальную информацию и используется операционной системой только после того, как принято решение о возобновлении прерванного процесса.

Очереди процессов представляют собой дескрипторы отдельных процессов, объединенные в списки. Таким образом, каждый дескриптор, кроме всего прочего, содержит по крайней мере один указатель на другой дескриптор, соседствующий с ним в очереди. Такая организация очередей позволяет легко их переупорядочивать, включать и исключать процессы, переводить процессы из одного состояния в другое.

Программный код только тогда начнет выполняться, когда для него операционной системой будет создан процесс. Создать процесс — это значит:

- создать информационные структуры, описывающие данный процесс, то есть его дескриптор и контекст;
- включить дескриптор нового процесса в очередь готовых процессов;
- загрузить кодовый сегмент процесса в оперативную память или в область свопинга.

2.1.3. Алгоритмы планирования процессов

Планирование процессов включает в себя решение следующих задач:

- определение момента времени для смены выполняемого процесса;
- выбор процесса на выполнение из очереди готовых процессов;
- переключение контекстов "старого" и "нового" процессов.

Первые две задачи решаются программными средствами, а последняя в значительной степени аппаратно.

Существует множество различных алгоритмов планирования процессов, по-разному решающих вышеперечисленные задачи, преследующих различные цели и обеспечивающих различное качество мультипрограммирования. Среди этого множества алгоритмов рассмотрим подробнее две группы наиболее часто встречающихся алгоритмов: алгоритмы, основанные на *квантовании,* и алгоритмы, основанные на *приоритетах*.

В соответствии с алгоритмами, основанными на квантовании, смена активного процесса происходит, если:

- процесс завершился и покинул систему;
- произошла ошибка;
- процесс перешел в состояние *ожидание*;
- исчерпан квант процессорного времени, отведенный данному процессу.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Процесс, который исчерпал свой квант, переводится в состояние *готовность* и ожидает, когда ему будет предоставлен новый квант процессорного времени, а на выполнение в соответствии с определенным правилом выбирается новый процесс из очереди готовых. Таким образом, ни один процесс не занимает процессор надолго, поэтому квантование широко используется в системах разделения времени. Граф состояний процесса, изображенный на рис. 2.1, соответствует алгоритму планирования, основанному на квантовании.

Кванты, выделяемые процессам, могут быть одинаковыми для всех процессов или различными. Кванты, выделяемые одному процессу, могут быть фиксированной величины или изменяться в разные периоды жизни процесса. Процессы, которые не полностью использовали выделенный им квант (например, из-за ухода на выполнение операций ввода-вывода), могут получить или не получить компенсацию в виде привилегий при последующем обслуживании. По-разному может быть организована очередь готовых процессов: циклически, по правилу "первый пришел — первый обслужился" (FIFO) или по правилу "последний пришел — первый обслужился" (LIFO).

Другая группа алгоритмов использует понятие "приоритет" процесса. *Приоритет* — это число, характеризующее степень привилегированности процесса при использовании ресурсов вычислительной машины, в частности, процессорного времени: чем выше приоритет, тем выше привилегии.

Приоритет может выражаться целыми или дробными, положительным или отрицательным значением. Чем выше привилегии процесса, тем меньше времени он будет проводить в очередях. Приоритет может назначаться директивно администратором системы в зависимости от важности работы или внесенной платы либо вычисляться самой ОС по определенным правилам, он может оставаться фиксированным на протяжении всей жизни процесса либо изменяться во времени в соответствии с некоторым законом. В последнем случае приоритеты называются динамическими.

Существует две разновидности приоритетных алгоритмов: алгоритмы, использующие относительные приоритеты, и алгоритмы, использующие абсолютные приоритеты.

В обоих случаях выбор процесса на выполнение из очереди готовых осуществляется одинаково: выбирается процесс, имеющий наивысший приоритет. По-разному решается проблема определения момента смены активного процесса. В системах с относительными приоритетами активный процесс выполняется до тех пор, пока он сам не покинет процессор, перейдя в состояние *ожидание* (или же произойдет ошибка, или процесс завершится). В системах с абсолютными приоритетами выполнение активного процесса прерывается еще при одном условии: если в очереди готовых процессов появился процесс, приоритет которого выше приоритета активного процесса. В этом случае прерванный процесс переходит в состояние готовности. На рис. 2.2 показаны графы состояний процесса для алгоритмов с относительными (а) и абсолютными (б) приоритетами.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Во многих операционных системах алгоритмы планирования построены с использованием как квантования, так и приоритетов. Например, в основе планирования лежит квантование, но величина кванта и/или порядок выбора процесса из очереди готовых определяется приоритетами процессов.

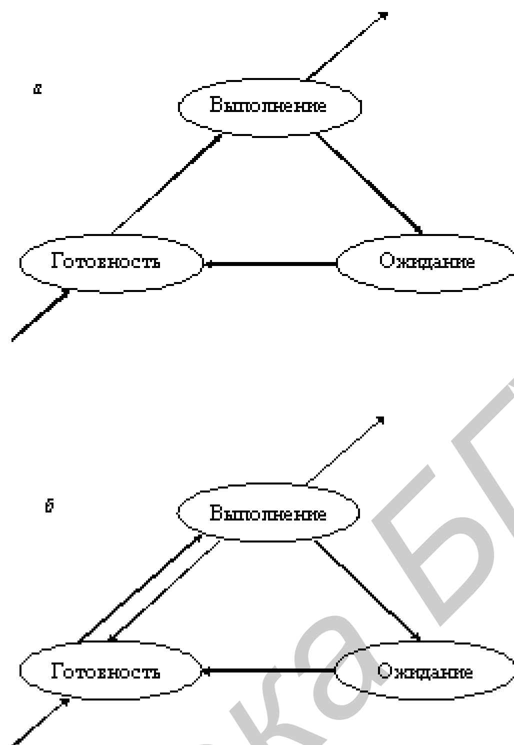


Рис. 2.2. Графы состояний процессов в системах с относительными (а) и абсолютными (б) приоритетами

2.1.4. Вытесняющие и невытесняющие алгоритмы планирования

Существует два основных типа процедур планирования процессов: вытесняющие и невытесняющие.

Невытесняющая многозадачность — это способ планирования процессов, при котором активный процесс выполняется до тех пор, пока он сам, по собственной инициативе, не отдаст управление планировщику операционной системы, для того чтобы тот выбрал из очереди другой, готовый к выполнению процесс.

Вытесняющая многозадачность — это такой способ, при котором решение о переключении процессора с выполнения одного процесса на выполнение другого процесса принимается планировщиком операционной системы, а не самой активной задачей.

Понятия вытесняющей и невытесняющей многозадачности иногда отождествляются с понятиями приоритетных и бесприоритетных дисциплин, что совершенно неверно, а также с понятиями абсолютных и относительных приоритетов, что неверно отчасти. Вытесняющая и невытесняющая многозадачность — это более широкие понятия, чем типы приоритетности. Приоритеты задач могут как использоваться, так и не использоваться и при вытесняющих, и при невытесняющих способах

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

планирования. Так, в случае использования приоритетов дисциплина относительных приоритетов может быть отнесена к классу систем с невытесняющей многозадачностью, а дисциплина абсолютных приоритетов — к классу систем с вытесняющей многозадачностью. Неприоритетная дисциплина планирования, основанная на выделении равных квантов времени для всех задач, относится к вытесняющим алгоритмам.

Основным различием между вытесняющей и невытесняющей вариантами многозадачности является степень централизации механизма планирования задач. При вытесняющей многозадачности механизм планирования задач целиком сосредоточен в операционной системе и программист пишет свое приложение, не заботясь о том, что оно будет выполняться параллельно с другими задачами. При этом операционная система выполняет следующие функции: определяет момент снятия с выполнения активной задачи, запоминает ее контекст, выбирает из очереди готовых задач следующую и запускает ее на выполнение, загружая ее контекст.

При невытесняющей многозадачности механизм планирования распределен между системой и прикладными программами. Прикладная программа, получив управление от операционной системы, сама определяет момент завершения своей очередной итерации и передает управление ОС с помощью какого-либо системного вызова, а ОС формирует очереди задач и выбирает в соответствии с некоторым алгоритмом (например с учетом приоритетов) следующую задачу на выполнение. Такой механизм создает проблемы как для пользователей, так и для разработчиков.

Для пользователей это означает, что управление системой теряется на произвольный период времени, который определяется приложением (а не пользователем). Если приложение тратит слишком много времени на выполнение какой-либо работы, например на форматирование диска, пользователь не может переключиться с этой задачи на другую задачу, например, на текстовый редактор, в то время как форматирование продолжалось бы в фоновом режиме. Эта ситуация нежелательна, так как пользователи обычно не хотят долго ждать, когда машина завершит свою задачу.

Поэтому разработчики приложений для non-preemptive операционной среды, возлагая на себя функции планировщика, должны создавать приложения так, чтобы они выполняли свои задачи небольшими частями. Например, программа форматирования может отформатировать одну дорожку дискеты и вернуть управление системе. После выполнения других задач система возвратит управление программе форматирования, чтобы та отформатировала следующую дорожку. Подобный метод разделения времени между задачами работает, но он существенно затрудняет разработку программ и предъявляет повышенные требования к квалификации программиста. Программист должен обеспечить "дружественное" отношение своей программы к другим выполняемым одновременно с ней программам, достаточно часто отдавая им управление. Крайним проявлением "недружественности" приложения является его зависание, которое приводит к общему краху системы. В системах с

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

вытесняющей многозадачностью такие ситуации, как правило, исключены, так как центральный планирующий механизм снимет зависшую задачу с выполнения.

Однако распределение функций планировщика между системой и приложениями не всегда является недостатком, а при определенных условиях может быть и преимуществом, потому что дает возможность разработчику приложений самому проектировать алгоритм планирования, наиболее подходящий для данного фиксированного набора задач. Так как разработчик сам определяет в программе момент времени отдачи управления, то при этом исключаются нерациональные прерывания программ в "неудобные" для них моменты времени. Кроме того, легко разрешаются проблемы совместного использования данных: задача во время каждой итерации использует их монопольно и уверена, что на протяжении этого периода никто другой не изменит эти данные. Существенным преимуществом non-preemptive систем является более высокая скорость переключения с задачи на задачу.

Примером эффективного использования невытесняющей многозадачности является файл-сервер NetWare, в котором, в значительной степени благодаря этому, достигнута высокая скорость выполнения файловых операций. Менее удачным оказалось использование невытесняющей многозадачности в операционной среде Windows 3.x.

Однако почти во всех современных операционных системах, ориентированных на высокопроизводительное выполнение приложений (UNIX, Windows NT, OS/2, VAX/VMS), реализована вытесняющая многозадачность. В последнее время дошла очередь и до ОС класса настольных систем, например, OS/2 Warp и Windows 95. Возможно, в связи с этим вытесняющую многозадачность часто называют истинной многозадачностью.

2.2. Средства синхронизации и взаимодействия процессов

2.2.1. Проблема синхронизации

Процессам часто нужно взаимодействовать друг с другом, например, один процесс может передавать данные другому процессу, или несколько процессов могут обрабатывать данные из общего файла. Во всех этих случаях возникает проблема синхронизации процессов, которая может решаться приостановкой и активизацией процессов, организацией очередей, блокированием и освобождением ресурсов.

Пренебрежение вопросами синхронизации процессов, выполняющихся в режиме мультипрограммирования, может привести к их неправильной работе или даже к краху системы. Рассмотрим, например (рис. 2.3), программу печати файлов (принт-сервер). Эта программа печатает по очереди все файлы, имена которых последовательно в порядке поступления записывают в специальный общедоступный файл "заказов" другие программы. Особая переменная NEXT, также доступная всем

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

процессам-клиентам, содержит номер первой свободной для записи имени файла позиции файла "заказов". Процессы-клиенты читают эту переменную, записывают в соответствующую позицию файла "заказов" имя своего файла и наращивают значение NEXT на единицу. Предположим, что в некоторый момент процесс R решил распечатать свой файл, для этого он прочитал значение переменной NEXT, которое для определенности предположим равным 4. Процесс запомнил это значение, но поместить имя файла не успел, так как его выполнение было прервано (например, в следствие исчерпания кванта). Очередной процесс S, желающий распечатать файл, прочитал то же самое значение переменной NEXT, поместил в четвертую позицию имя своего файла и нарастил значение переменной на единицу. Когда в очередной раз управление будет передано процессу R, то он, продолжая свое выполнение в полном соответствии со значением текущей свободной позиции, полученным во время предыдущей итерации, запишет имя файла также в позицию 4, поверх имени файла процесса S.

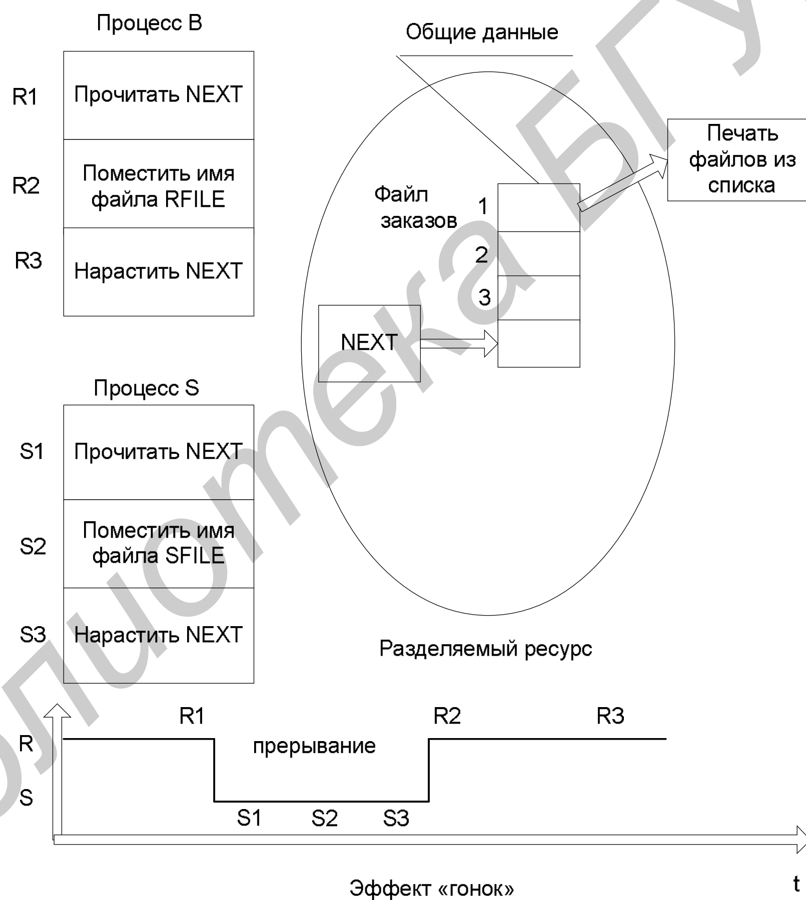


Рис. 2.3. Пример необходимости синхронизации

Таким образом, процесс S никогда не увидит свой файл распечатанным. Сложность проблемы синхронизации состоит в нерегулярности возникающих ситуаций – в предыдущем примере можно представить и другое развитие событий: были потеряны файлы нескольких процессов или, напротив, не был потерян ни один файл. В данном случае все определяется взаимными скоростями процессов и моментами их прерывания. Поэтому отладка взаимодействующих процессов является сложной задачей.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Определение 2.3. Гонка – ситуация, когда два или более процессов обрабатывают разделяемые данные и конечный результат зависит от соотношения скоростей процессов и/или порядка выполнения процессов.

2.2.2. Критическая секция

Важным понятием синхронизации процессов является понятие "критическая секция" программы.

Определение 2.4. Критическая секция – это часть программы, в которой осуществляется доступ к разделяемым данным.

Для того чтобы исключить эффект гонок по отношению к некоторому ресурсу, необходимо обеспечить, чтобы в каждый момент в критической секции, связанной с этим ресурсом, находился максимум один процесс.

Определение 2.5. Взаимное исключение – это принцип организации доступа к некоторому разделяемому ресурсу, когда в каждый момент времени с ресурсом работает максимум один процесс.

Простейший способ обеспечить взаимное исключение — позволить процессу, находящемуся в критической секции, запрещать все прерывания. Однако этот способ непригоден, так как опасно доверять управление системой пользовательскому процессу: он может надолго занять процессор, а при крахе процесса в критической области крах потерпит вся система, потому что прерывания никогда не будут разрешены.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

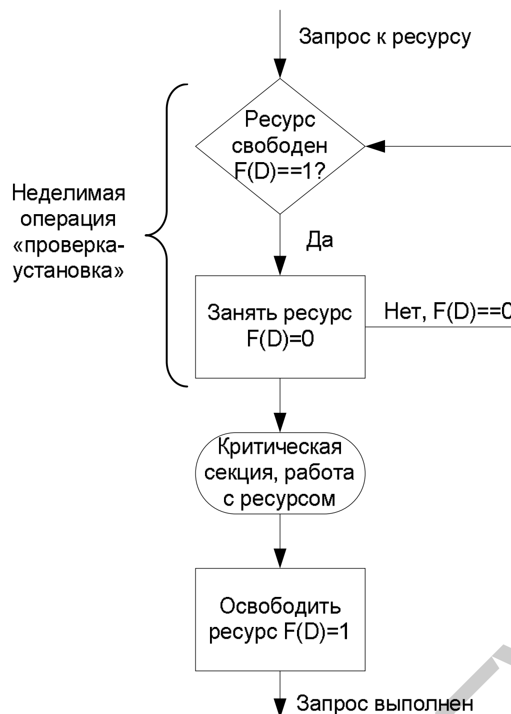


Рис. 2.4. Реализация критических секций с использованием блокирующих переменных

Другим способом является использование блокирующих переменных. С каждым разделяемым ресурсом связывается двоичная переменная, которая принимает значение 1, если ресурс свободен (то есть ни один процесс не находится в данный момент в критической секции, связанной с данным процессом), и значение 0, если ресурс занят. На рис. 2.4 показан фрагмент алгоритма процесса, использующего для реализации взаимного исключения доступа к разделяемому ресурсу D блокирующую переменную F(D). Перед входом в критическую секцию процесс проверяет, свободен ли ресурс D. Если он занят, то проверка циклически повторяется, если свободен, то значение переменной F(D) устанавливается равным 0 и процесс входит в критическую секцию. После того как процесс выполнит все действия с разделяемым ресурсом D, значение переменной F(D) снова устанавливается равным 1.

Если все процессы написаны с использованием вышеописанных соглашений, то взаимное исключение гарантируется. Следует заметить, что операция проверки и установки блокирующей переменной должна быть неделимой. Поясним это. Пусть в результате проверки переменной процесс определил, что ресурс свободен, но сразу после этого, не успев установить переменную равную 0, был прерван. За время его приостановки другой процесс занял ресурс, вошел в свою критическую секцию, но также был прерван, не завершив работы с разделяемым ресурсом. Когда управление было возвращено первому процессу, он, считая ресурс свободным, установил признак занятости и начал выполнять свою критическую секцию. Таким образом был нарушен принцип взаимного исключения, что потенциально может привести к нежелаемым последствиям. Во избежание таких ситуаций в системе команд машины желательно иметь единую

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

команду "проверка-установка" или же реализовывать системными средствами соответствующие программные примитивы, которые бы запрещали прерывания на протяжении всей операции проверки и установки.

Реализация критических секций с использованием блокирующих переменных имеет существенный недостаток: в течение времени, когда один процесс находится в критической секции, другой процесс, которому требуется тот же ресурс, будет выполнять рутинные действия по опросу блокирующей переменной, бесполезно тратя процессорное время. Для устранения таких ситуаций может быть использован так называемый аппарат событий. С помощью этого средства могут решаться не только проблемы взаимного исключения, но и более общие задачи синхронизации процессов. В разных операционных системах аппарат событий реализуется по-своему, но в любом случае используются системные функции аналогичного назначения, которые условно назовем WAIT(x) и POST(x), где x — идентификатор некоторого события. На рис. 2.5 показан фрагмент алгоритма процесса, использующего эти функции. Если ресурс занят, то процесс не выполняет циклический опрос, а вызывает системную функцию WAIT(D), здесь D обозначает событие, заключающееся в освобождении ресурса D. Функция WAIT(D) переводит активный процесс в состояние *ожидание* и делает отметку в его дескрипторе о том, что процесс ожидает события D. Процесс, который в это время использует ресурс D, после выхода из критической секции выполняет системную функцию POST(D), в результате чего операционная система просматривает очередь ожидающих процессов и переводит процесс, ожидающий события D, в состояние *готовность*.

Обобщающее средство синхронизации процессов предложил Дейкстра, который ввел два новых примитива. В абстрактной форме эти примитивы, обозначаемые P и V, оперируют целыми неотрицательными переменными, называемыми *семафорами*. Пусть S — такой семафор. Операции определяются следующим образом:

V(S) : переменная S увеличивается на 1 одним неделимым действием; выборка, инкремент и запоминание не могут быть прерваны, и к S нет доступа другим процессам во время выполнения этой операции.

P(S) : уменьшение S на 1, если это возможно. Если S = 0, то невозможно уменьшить S и остаться в области целых неотрицательных значений; в этом случае процесс, вызывающий P-операцию, ждет, пока это уменьшение станет возможным. Успешная проверка и уменьшение также является неделимой операцией.

В частном случае, когда семафор S может принимать только значения 0 и 1, он превращается в блокирующую переменную. Операция P заключает в себе потенциальную возможность перехода процесса, который ее

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

выполняет, в состояние ожидания, в то время как V-операция может при некоторых обстоятельствах активизировать другой процесс, приостановленный операцией P (сравните эти операции с системными функциями WAIT и POST).

Рассмотрим использование семафоров на классическом примере взаимодействия двух процессов, выполняющихся в режиме мультипрограммирования, один из которых пишет данные в буферный пул, а другой считывает их из буферного пула. Пусть буферный пул состоит из N буферов, каждый из которых может содержать одну запись. Процесс "писатель" должен приостанавливаться, когда все буфера оказываются занятыми, и активизироваться при освобождении хотя бы одного буфера. Напротив, процесс "читатель" приостанавливается, когда все буфера пусты, и активизируется при появлении хотя бы одной записи.

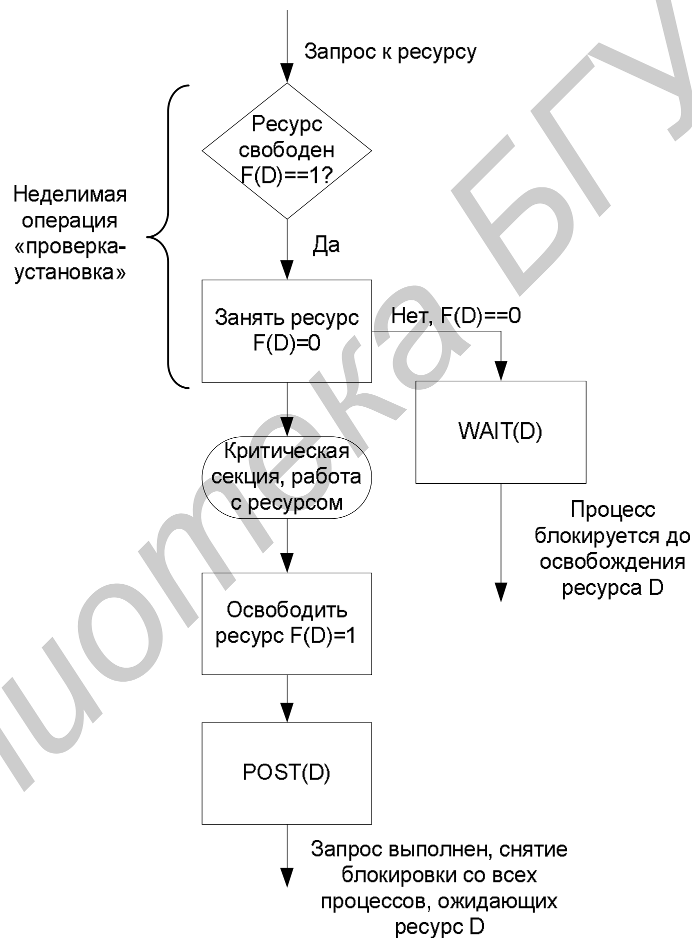


Рис. 2.5. Реализация критической секции с использованием системных функций WAIT(D) и POST(D)

Введем два семафора: e — число пустых буферов и f — число заполненных буферов. Предположим, что запись в буфер и считывание из буфера являются критическими секциями (как в примере с принт-сервером в начале данного раздела). Введем также двоичный семафор b, используемый для обеспечения взаимного исключения.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

2.2.3. Тупики

Приведенный выше пример поможет нам проиллюстрировать еще одну проблему синхронизации — *взаимные блокировки*, называемые также *дедлоками (deadlocks), клинчами (clinch)* или *тупиками*. Если переставить местами операции P(e) и P(b) в программе "писатель", то при некотором стечении обстоятельств эти два процесса могут взаимно заблокировать друг друга. Действительно, пусть "писатель" первым войдет в критическую секцию и обнаружит отсутствие свободных буферов; он начнет ждать, когда "читатель" возьмет очередную запись из буфера, но "читатель" не сможет этого сделать, так как для этого необходимо войти в критическую секцию, вход в которую заблокирован процессом "писателем".

Рассмотрим еще один пример тупика. Пусть двум процессам, выполняющимся в режиме мультипрограммирования, для выполнения их работы нужно два ресурса, например, принтер и диск. На рис. 2.6, а показаны фрагменты соответствующих программ. И пусть после того, как процесс А занял принтер (установил блокирующую переменную), он был прерван. Управление получил процесс В, который сначала занял диск, но при выполнении следующей команды был заблокирован, так как принтер оказался уже занятым процессом А. Управление снова получил процесс А, который в соответствии со своей программой сделал попытку занять диск и был заблокирован: диск уже распределен процессу В. В таком положении процессы А и В могут находиться сколь угодно долго.

В зависимости от соотношения скоростей процессов, они могут либо совершенно независимо использовать разделяемые ресурсы (рис. 2.6, г), либо образовывать очереди к разделяемым ресурсам (рис. 2.6, в), либо взаимно блокировать друг друга (рис. 2.6, б). Тупиковые ситуации надо отличать от простых очередей, хотя и те и другие возникают при совместном использовании ресурсов и внешне выглядят похоже: процесс приостанавливается и ждет освобождения ресурса. Однако очередь — это нормальное явление, неотъемлемый признак высокого коэффициента использования ресурсов при случайном поступлении запросов. Она возникает тогда, когда ресурс недоступен в данный момент, но через некоторое время он освобождается и процесс продолжает свое выполнение. Тупик же, что видно из его названия, является в некотором роде неразрешимой ситуацией.

В рассмотренных примерах тупик был образован двумя процессами, но взаимно блокировать друг друга может и большее число процессов.

Проблема тупиков включает в себя следующие задачи:

- предотвращение тупиков;
- распознавание тупиков;
- восстановление системы после тупиков.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Тупики могут быть предотвращены на стадии написания программ, то есть программы должны быть написаны таким образом, чтобы тупик не мог возникнуть ни при каком соотношении взаимных скоростей процессов. Так, если бы в предыдущем примере процесс А и процесс В запрашивали ресурсы в одинаковой последовательности, то тупик был бы в принципе невозможен. Второй подход к предотвращению тупиков называется динамическим и заключается в использовании определенных правил при назначении ресурсов процессам – например, ресурсы могут выделяться в определенной последовательности, общей для всех процессов.

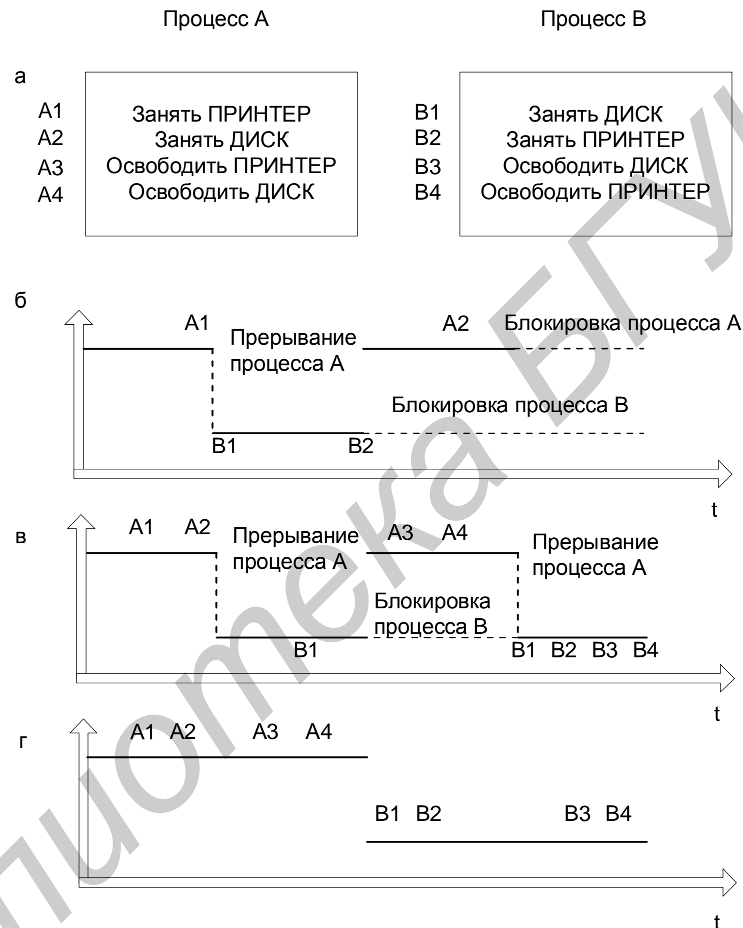


Рис. 2.6. Фрагменты программ А и В, разделяющих принтер и диск (а); взаимная блокировка (б); очередь к разделяемому диску (в); независимое использование ресурсов (г)

В некоторых случаях, когда тупиковая ситуация образована многими процессами, использующими много ресурсов, распознавание тупика является нетривиальной задачей. Существуют формальные, программно-реализованные методы распознавания тупиков, основанные на ведении таблиц распределения ресурсов и таблиц запросов к занятым ресурсам. Анализ этих таблиц позволяет обнаружить взаимные блокировки.

Если же тупиковая ситуация возникла, то не обязательно снимать с выполнения все заблокированные процессы. Можно снять только часть из них, при этом освобождаются ресурсы, ожидаемые остальными

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

процессами, можно вернуть некоторые процессы в область свопинга, можно совершить "откат" некоторых процессов до так называемой контрольной точки, в которой запоминается вся информация, необходимая для восстановления выполнения программы с данного места. Контрольные точки расставляются в программе в местах, после которых возможно возникновение тупика.

Из всего вышесказанного ясно, что использовать семафоры нужно очень осторожно, так как одна незначительная ошибка может привести к останову системы. Для того чтобы облегчить написание корректных программ, было предложено высокоуровневое средство синхронизации, называемое монитором. *Монитор* — это набор процедур, переменных и структур данных. Процессы могут вызывать процедуры монитора, но не имеют доступа к внутренним данным монитора. Мониторы имеют важное свойство, которое делает их полезными для достижения взаимного исключения: только один процесс может быть активным по отношению к монитору. Компилятор обрабатывает вызовы процедур монитора особым образом. Обычно, когда процесс вызывает процедуру монитора, то первые несколько инструкций этой процедуры проверяют, не активен ли какой-либо другой процесс по отношению к этому монитору. Если да, то вызывающий процесс приостанавливается, пока другой процесс не освободит монитор. Таким образом, исключение входа нескольких процессов в монитор реализуется не программистом, а компилятором, что делает ошибки менее вероятными.

В распределенных системах, состоящих из нескольких процессоров, каждый из которых имеет собственную оперативную память, семафоры и мониторы оказываются непригодными. В таких системах синхронизация может быть реализована только с помощью обмена сообщениями.

2.2.4. Нити

Многозадачность является важнейшим свойством ОС. Для поддержки этого свойства ОС определяет и оформляет для себя те внутренние единицы работы, между которыми и будут разделяться процессор и другие ресурсы компьютера. Эти внутренние единицы работы в разных ОС носят разные названия: задача, задание, процесс, нить. В некоторых случаях сущности, обозначаемые этими понятиями, принципиально отличаются друг от друга.

Говоря о процессах, мы отмечали, что операционная система поддерживает их обособленность: у каждого процесса имеется свое виртуальное адресное пространство, каждому процессу назначаются свои ресурсы — файлы, окна, семафоры и т.д. Такая обособленность нужна для того, чтобы защитить один процесс от другого, поскольку они, совместно используя все ресурсы машины, конкурируют с друг другом. В общем случае процессы принадлежат разным пользователям, разделяющим один

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

компьютер, и ОС берет на себя роль арбитра в спорах процессов за ресурсы.

При мультипрограммировании повышается пропускная способность системы, но отдельный процесс никогда не может быть выполнен быстрее, чем если бы он выполнялся в однопрограммном режиме (всякое разделение ресурсов замедляет работу одного из участников за счет дополнительных затрат времени на ожидание освобождения ресурса). Однако задача, решаемая в рамках одного процесса, может обладать внутренним параллелизмом, который в принципе позволяет ускорить ее решение. Например, в ходе выполнения задачи происходит обращение к внешнему устройству, и на время этой операции можно не блокировать полностью выполнение процесса, а продолжить вычисления по другой "ветви" процесса.

Для этих целей современные ОС предлагают использовать сравнительно новый механизм *многонитевой обработки (multithreading)*. При этом вводится новое понятие "нить" (thread), а понятие "процесс" в значительной степени меняет смысл.

Мультипрограммирование теперь реализуется на уровне нитей, и задача, оформленная в виде нескольких нитей в рамках одного процесса, может быть выполнена быстрее за счет псевдопараллельного (или параллельного в мультипроцессорной системе) выполнения ее отдельных частей. Например, если электронная таблица была разработана с учетом возможностей многонитевой обработки, то пользователь может запросить пересчет своего рабочего листа и одновременно продолжать заполнять таблицу. Особенно эффективно можно использовать многонитевость для выполнения распределенных приложений, например, многонитевый сервер может параллельно выполнять запросы сразу нескольких клиентов.

Нити, относящиеся к одному процессу, не настолько изолированы друг от друга, как процессы в традиционной многозадачной системе, между ними легко организовать тесное взаимодействие. Действительно, в отличие от процессов, которые принадлежат разным, вообще говоря, конкурирующим приложениям, все нити одного процесса всегда принадлежат одному приложению, поэтому программист, пишущий это приложение, может заранее продумать работу множества нитей процесса таким образом, чтобы они могли взаимодействовать, а не бороться за ресурсы.

В традиционных ОС понятие "нить" тождественно понятию "процесс". В действительности часто бывает желательно иметь несколько нитей, разделяющих единое адресное пространство, но выполняющихся квазипараллельно, благодаря чему нити становятся подобными процессам (за исключением разделяемого адресного пространства).

Нити иногда называют облегченными процессами или мини-процессами. Действительно, нити во многих отношениях подобны процессам. Каждая нить выполняется строго последовательно и имеет свой собственный программный счетчик и стек. Нити, как и процессы, могут, например,

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

порождать нити-потомки, могут переходить из состояния в состояние. Подобно традиционным процессам (то есть процессам, состоящим из одной нити), нити могут находится в одном из следующих состояний: *выполнение*, *ожидание* и *готовность*. Пока одна нить заблокирована, другая нить того же процесса может выполняться. Нити разделяют процессор так, как это делают процессы, в соответствии с различными вариантами планирования.

Однако различные нити в рамках одного процесса не настолько независимы, как отдельные процессы. Все такие нити имеют одно и то же адресное пространство. Это означает, что они разделяют одни и те же глобальные переменные. Поскольку каждая нить может иметь доступ к каждому виртуальному адресу, одна нить может использовать стек другой нити. Между нитями нет полной защиты, потому что, во-первых, это невозможно, а во-вторых, не нужно. Все нити одного процесса всегда решают общую задачу одного пользователя, и аппарат нитей используется здесь для более быстрого решения задачи путем ее распараллеливания. При этом программисту очень важно получить в свое распоряжения удобные средства организации взаимодействия частей одной задачи. Кроме разделения адресного пространства, все нити разделяют также набор открытых файлов, таймеров, сигналов и т.п.

Итак, нити имеют собственные:

- программный счетчик;
- стек;
- регистры;
- нити-потомки;
- состояние.

Нити разделяют:

- адресное пространство;
- глобальные переменные;
- открытые файлы;
- таймеры;
- семафоры;
- статистическую информацию.

Многонитевая обработка повышает эффективность работы системы по сравнению с многозадачной обработкой. Например, в многозадачной среде Windows можно одновременно работать с электронной таблицей и текстовым редактором. Однако, если пользователь запрашивает пересчет своего рабочего листа, электронная таблица блокируется до тех пор, пока эта операция не завершится, что может потребовать значительного времени. В многонитевой среде в случае, если электронная таблица была разработана с учетом возможностей многонитевой обработки,

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

предоставляемых программисту, этой проблемы не возникает и пользователь всегда имеет доступ к электронной таблице.

Широкое применение находит многонитевая обработка в распределенных системах. Смотрите об этом в разделе "Процессы и нити в распределенных системах".

Некоторые прикладные задачи легче программировать, используя параллелизм, например задачи типа "писатель—читатель", в которых одна нить выполняет запись в буфер, а другая считывает записи из него. Поскольку они разделяют общий буфер, не стоит их делать отдельными процессами. Другой пример использования нитей — это управление сигналами, такими, как прерывание с клавиатуры (del или break). Вместо обработки сигнала прерывания, одна нить назначается для постоянного ожидания поступления сигналов. Таким образом, использование нитей может сократить необходимость в прерываниях пользовательского уровня. В этих примерах важно не столько параллельное выполнение, сколько важна ясность программы.

Наконец, в мультипроцессорных системах для нитей из одного адресного пространства имеется возможность выполнения параллельно на разных процессорах. Это действительно один из главных путей реализации разделения ресурсов в таких системах. С другой стороны, правильно сконструированные программы, которые используют нити, должны работать одинаково хорошо как на однопроцессорной машине в режиме разделения времени между нитями, так и на настоящем мультипроцессоре.

2.3. Управление памятью

Память является важнейшим ресурсом, требующим тщательного управления со стороны мультипрограммной операционной системы. Распределению подлежит вся оперативная память, не занятая операционной системой. Обычно ОС располагается в самых младших адресах, однако может занимать и самые старшие адреса. Функциями ОС по управлению памятью являются: отслеживание свободной и занятой памяти, выделение памяти процессам и освобождение памяти при завершении процессов, вытеснение процессов из оперативной памяти на диск, когда размеры основной памяти недостаточны для размещения в ней всех процессов, и возвращение их в оперативную память, когда в ней освобождается место, а также настройка адресов программы на конкретную область физической памяти.

2.3.1. Типы адресов

Для идентификации переменных и команд используются символьные имена (метки), виртуальные адреса и физические адреса (рис. 2.7).

Символьные имена присваивает пользователь при написании программы на алгоритмическом языке или ассемблере.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Виртуальные адреса вырабатывает транслятор, переводящий программу на машинный язык. Так как во время трансляции в общем случае неизвестно, в какое место оперативной памяти будет загружена программа, то транслятор присваивает переменным и командам виртуальные (условные) адреса, обычно считая по умолчанию, что программа будет размещена начиная с нулевого адреса.

Определение 2.6. Виртуальное адресное пространство – это совокупность виртуальных адресов процесса. Каждый процесс имеет собственное виртуальное адресное пространство. Максимальный размер виртуального адресного пространства ограничивается разрядностью адреса, присущей данной архитектуре компьютера, и, как правило, не совпадает с объемом физической памяти, имеющимся в компьютере.

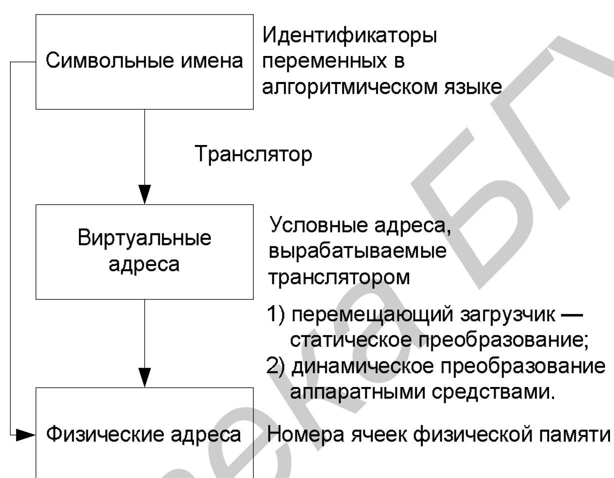


Рис. 2.7. Типы адресов

Физические адреса соответствуют номерам ячеек оперативной памяти, где в действительности расположены или будут расположены переменные и команды. Переход от виртуальных адресов к физическим может осуществляться двумя способами. В первом случае замену виртуальных адресов на физические делает специальная системная программа — перемещающий загрузчик. Перемещающий загрузчик на основании имеющихся у него исходных данных о начальном адресе физической памяти, в которую предстоит загружать программу, и информации, предоставленной транслятором об адресно-зависимых константах программы выполняет загрузку программы, совмещая ее с заменой виртуальных адресов физическими.

Второй способ заключается в том, что программа загружается в память в неизмененном виде в виртуальных адресах, при этом операционная система фиксирует смещение действительного расположения программного кода относительно виртуального адресного пространства. Во время выполнения программы при каждом обращении к оперативной памяти выполняется преобразование виртуального адреса в физический. Второй способ является более гибким, он допускает перемещение

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

программы во время ее выполнения, в то время как перемещающий загрузчик жестко привязывает программу к первоначально выделенному ей участку памяти. Вместе с тем использование перемещающего загрузчика уменьшает накладные расходы, так как преобразование каждого виртуального адреса происходит только один раз во время загрузки, а во втором случае — каждый раз при обращении по данному адресу.

В некоторых случаях (обычно в специализированных системах), когда заранее точно известно, в какой области оперативной памяти будет выполняться программа, транслятор выдает исполняемый код сразу в физических адресах.

2.3.2. Методы распределения памяти

Все методы управления памятью могут быть разделены на два класса: методы, которые используют перемещение процессов между оперативной памятью и диском, и методы, которые не делают этого. Начнем с рассмотрения последнего, более простого класса методов – без использования дискового пространства.

2.3.3. Распределение памяти фиксированными разделами

Самым простым способом управления оперативной памятью является разделение ее на несколько разделов фиксированной величины. Это может быть выполнено вручную оператором во время старта системы или во время ее генерации. Очередная задача, поступившая на выполнение, помещается либо в общую очередь (рис. 2.8, а), либо в очередь к некоторому разделу (рис. 2.8, б).

Подсистема управления памятью в этом случае выполняет следующие задачи:

- сравнивая размер программы, поступившей на выполнение, и свободных разделов, выбирает подходящий раздел;
- осуществляет загрузку программы и настройку адресов.

При очевидном преимуществе — простоте реализации — данный метод имеет существенный недостаток — жесткость. Так как в каждом разделе может выполняться только одна программа, то уровень мультипрограммирования заранее ограничен числом разделов независимо от того, какой размер имеют программы. Даже если программа имеет небольшой объем, она будет занимать весь раздел, что приводит к неэффективному использованию памяти. С другой стороны, даже если объем оперативной памяти машины позволяет выполнить некоторую программу, разбиение памяти на разделы не позволяет сделать этого.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

2.3.4. Распределение памяти разделами переменной величины

В этом случае память машины не делится заранее на разделы. Сначала вся память свободна. Каждой вновь поступающей задаче выделяется необходимая ей память. Если достаточный объем памяти отсутствует, то задача не принимается на выполнение и стоит в очереди. После завершения задачи память освобождается и на это место может быть загружена другая задача. Таким образом, в произвольный момент времени оперативная память представляет собой случайную последовательность занятых и свободных участков (разделов) произвольного размера. На рис. 2.9 показано состояние памяти в различные моменты времени при использовании динамического распределения. Так, в момент t0 в памяти находится только ОС, а к моменту t1 память разделена между пятью задачами, причем задача П4, завершаясь, покидает память. На освободившееся после задачи П4 место загружается задача П6, поступившая в момент t3.

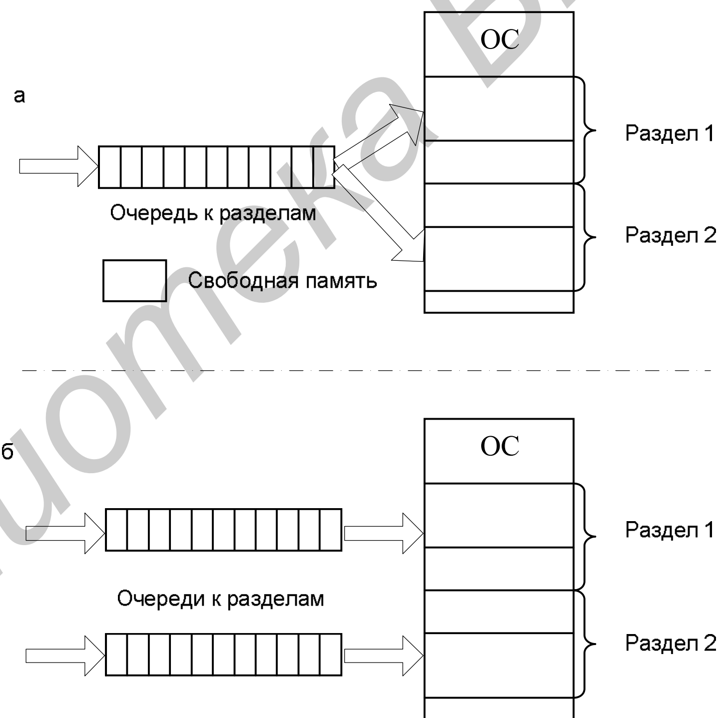


Рис. 2.8. Распределение памяти фиксированными разделами:
а — с общей очередью; б — с отдельными очередями

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

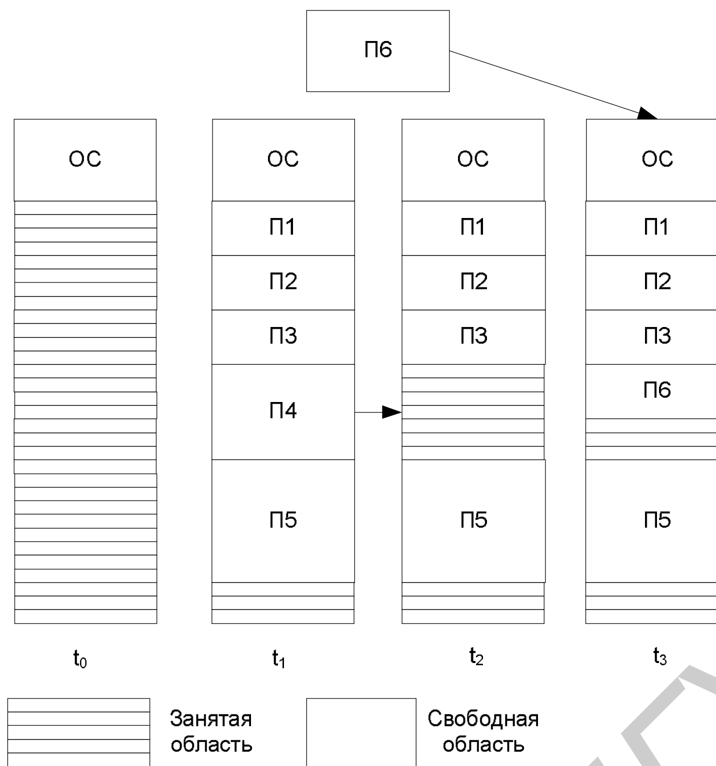


Рис. 2.9. Распределение памяти динамическими разделами

Задачами операционной системы при реализации данного метода управления памятью являются:

- ведение таблиц свободных и занятых областей, в которых указываются начальные адреса и размеры участков памяти;
- при поступлении новой задачи — анализ запроса, просмотр таблицы свободных областей и выбор раздела, размер которого достаточен для размещения поступившей задачи;
- загрузка задачи в выделенный ей раздел и корректировка таблиц свободных и занятых областей;
- после завершения задачи корректировка таблиц свободных и занятых областей.

Программный код не перемещается во время выполнения, то есть может быть проведена единовременная настройка адресов посредством использования перемещающего загрузчика.

Выбор раздела для вновь поступившей задачи может осуществляться по разным правилам, таким, например, как "первый попавшийся раздел достаточного размера", или "раздел, имеющий наименьший достаточный размер", или "раздел, имеющий наибольший достаточный размер". Все эти правила имеют свои преимущества и недостатки.

По сравнению с методом распределения памяти фиксированными разделами данный метод обладает гораздо большей гибкостью, но ему присущ очень серьезный недостаток — *фрагментация памяти*. Фрагментация — это наличие большого числа несмежных участков свободной памяти очень маленького размера (фрагментов). Настолько маленького, что ни одна из вновь поступающих программ не может

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

поместиться ни в одном из участков, хотя суммарный объем фрагментов может составить значительную величину, намного превышающую требуемый объем памяти.

2.3.5. Перемещаемые разделы

Одним из методов борьбы с фрагментацией является перемещение всех занятых участков в сторону старших либо в сторону младших адресов, так чтобы вся свободная память образовывала единую свободную область (рис. 2.10). В дополнение к функциям, которые выполняет ОС при распределении памяти переменными разделами, в данном случае она должна еще время от времени копировать содержимое разделов из одного места памяти в другое, корректируя таблицы свободных и занятых областей. Эта процедура называется "сжатием". Сжатие может выполняться либо при каждом завершении задачи, либо только тогда, когда для вновь поступившей задачи нет свободного раздела достаточного размера. В первом случае требуется меньше вычислительной работы при корректировке таблиц, а во втором — реже выполняется процедура сжатия. Так как программы перемещаются по оперативной памяти в ходе своего выполнения, то преобразование адресов из виртуальной формы в физическую должно выполняться динамическим способом.

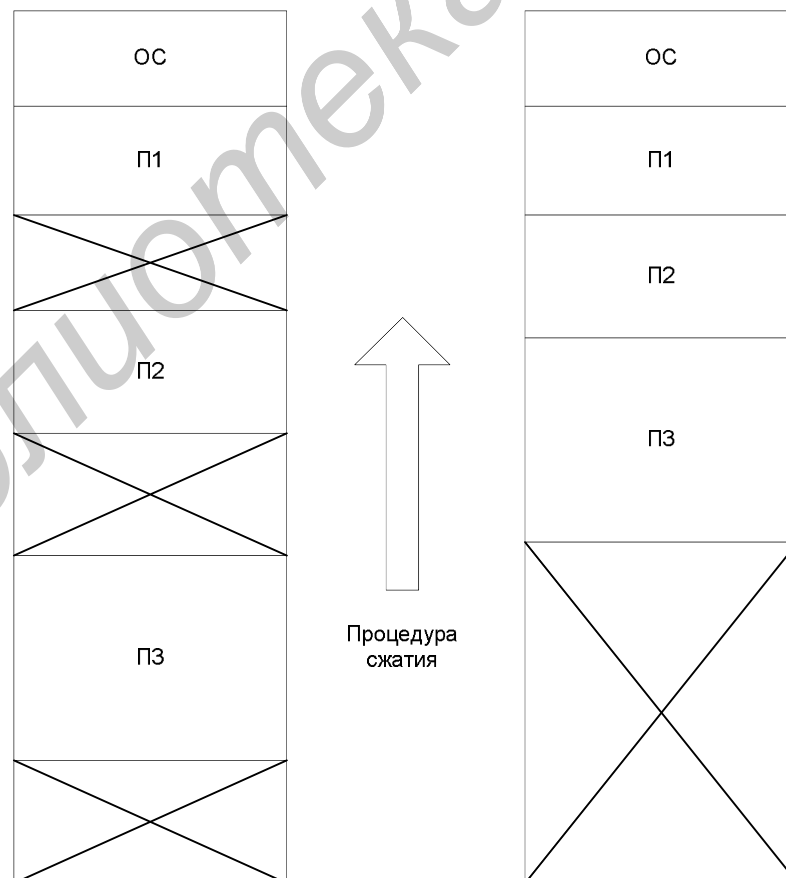


Рис. 2.10. Распределение памяти перемещаемыми разделами

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Хотя процедура сжатия и приводит к более эффективному использованию памяти, она может потребовать значительного времени, что часто перевешивает преимущества данного метода.

2.3.6. Понятие виртуальной памяти

Уже достаточно давно пользователи столкнулись с проблемой размещения в памяти программ, размер которых превышал имеющуюся в наличии свободную память. Решением было разбиение программы на части, называемые *оверлеями*. Нулевой оверлей начинал выполняться первым. Когда он заканчивал свое выполнение, то вызывал другой оверлей. Все оверлеи хранились на диске и перемещались между памятью и диском средствами операционной системы. Однако разбиение программы на части и планирование их загрузки в оперативную память должен был осуществлять программист.

Развитие методов организации вычислительного процесса в этом направлении привело к появлению метода, известного под названием *виртуальная память*. Виртуальным называется ресурс, который пользователю или пользовательской программе представляется обладающим свойствами, которыми он в действительности не обладает. Так, например, пользователю может быть предоставлена виртуальная оперативная память, размер которой превосходит всю имеющуюся в системе реальную оперативную память. Пользователь пишет программы так, как будто в его распоряжении имеется однородная оперативная память большого объема, но в действительности все данные, используемые программой, хранятся на одном или нескольких разнородных запоминающих устройствах, обычно на дисках, и при необходимости частями отображаются в реальную память.

Таким образом, виртуальная память — это совокупность программно-аппаратных средств, позволяющих пользователям писать программы, размер которых превосходит имеющуюся оперативную память. Для этого виртуальная память решает следующие задачи:

- размещает данные в запоминающих устройствах разного типа, например, часть программы в оперативной памяти, а часть — на диске;
- перемещает по мере необходимости данные между запоминающими устройствами разного типа, например, подгружает нужную часть программы с диска в оперативную память;
- преобразует виртуальные адреса в физические.

Все эти действия выполняются *автоматически*, без участия программиста, то есть механизм виртуальной памяти является прозрачным по отношению к пользователю.

Наиболее распространенными реализациями виртуальной памяти являются страничное, сегментное и странично-сегментное распределение памяти, а также свопинг.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

2.3.7. Страничное распределение

На рис. 2.11 показана схема страничного распределения памяти. Виртуальное адресное пространство каждого процесса делится на части одинакового, фиксированного для данной системы размера, называемые виртуальными страницами. В общем случае размер виртуального адресного пространства не является кратным размеру страницы, поэтому последняя страница каждого процесса дополняется фиктивной областью.

Вся оперативная память машины также делится на части такого же размера, называемые физическими страницами (или блоками). Размер страницы обычно выбирается равным степени двойки: 512, 1024 и т.д.; это позволяет упростить механизм преобразования адресов.

При загрузке процесса часть его виртуальных страниц помещается в оперативную память, а остальные — на диск. Смежные виртуальные страницы не обязательно располагаются в смежных физических страницах. При загрузке операционная система создает для каждого процесса информационную структуру — таблицу страниц, в которой устанавливается соответствие между номерами виртуальных и физических страниц для страниц, загруженных в оперативную память, или делается отметка о том, что виртуальная страница выгружена на диск. Кроме того, в таблице страниц содержится управляющая информация, такая, как признак модификации страницы, признак невыгружаемости (выгрузка некоторых страниц может быть запрещена), признак обращения к странице (используется для подсчета числа обращений за определенный период времени) и другие данные, формируемые и используемые механизмом виртуальной памяти.

При активизации очередного процесса в специальный регистр процессора загружается адрес таблицы страниц данного процесса.

При каждом обращении к памяти происходит чтение из таблицы страниц информации о виртуальной странице, к которой произошло обращение. Если данная виртуальная страница находится в оперативной памяти, то выполняется преобразование виртуального адреса в физический. Если же нужная виртуальная страница в данный момент выгружена на диск, то происходит так называемое страничное прерывание. Выполняющийся процесс переводится в состояние ожидания, и активизируется другой процесс из очереди готовых. Параллельно программа обработки страничного прерывания находит на диске требуемую виртуальную страницу и пытается загрузить ее в оперативную память. Если в памяти имеется свободная физическая страница, то загрузка выполняется немедленно, если же свободных страниц нет, то решается вопрос, какую страницу следует выгрузить из оперативной памяти.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

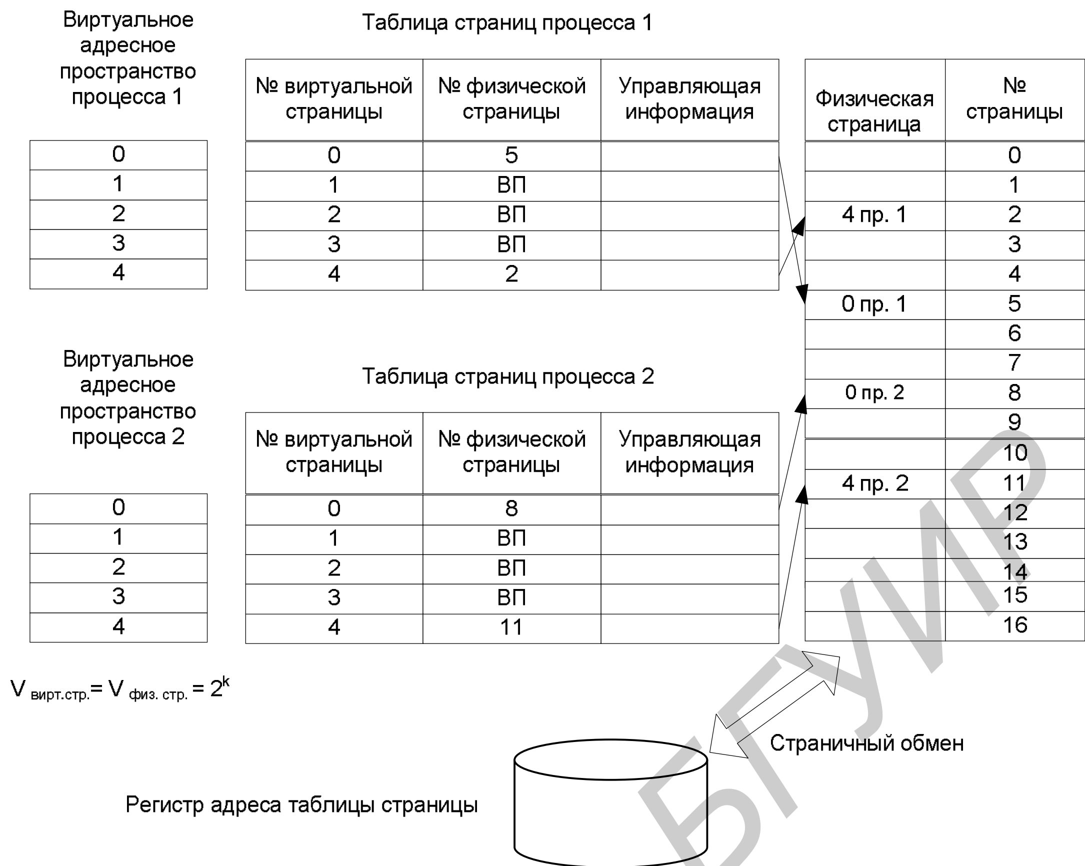


Рис. 2.11. Страничное распределение памяти

В данной ситуации может быть использовано много разных критериев выбора, наиболее популярные из них следующие:

- дольше всего не использовавшаяся страница;
- первая попавшаяся страница;
- страница, к которой в последнее время было меньше всего обращений.

В некоторых системах используется понятие рабочего множества страниц. Рабочее множество определяется для каждого процесса и представляет собой перечень наиболее часто используемых страниц, которые должны постоянно находиться в оперативной памяти и поэтому не подлежат выгрузке.

После того как выбрана страница, которая должна покинуть оперативную память, анализируется ее признак модификации (из таблицы страниц). Если выталкиваемая страница с момента загрузки была модифицирована, то ее новая версия должна быть переписана на диск. Если нет, то она может быть просто уничтожена, то есть соответствующая физическая страница объявляется свободной.

Рассмотрим механизм преобразования виртуального адреса в физический при страничной организации памяти (рис. 2.12).

Виртуальный адрес при страничном распределении может быть представлен в виде пары (p, s), где p — номер виртуальной страницы процесса (нумерация страниц начинается с 0), а s — смещение в пределах

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

виртуальной страницы. Учитывая, что размер страницы равен 2 в степени к, смещение s может быть получено простым отделением k младших разрядов в двоичной записи виртуального адреса. Оставшиеся старшие разряды представляют собой двоичную запись номера страницы p.

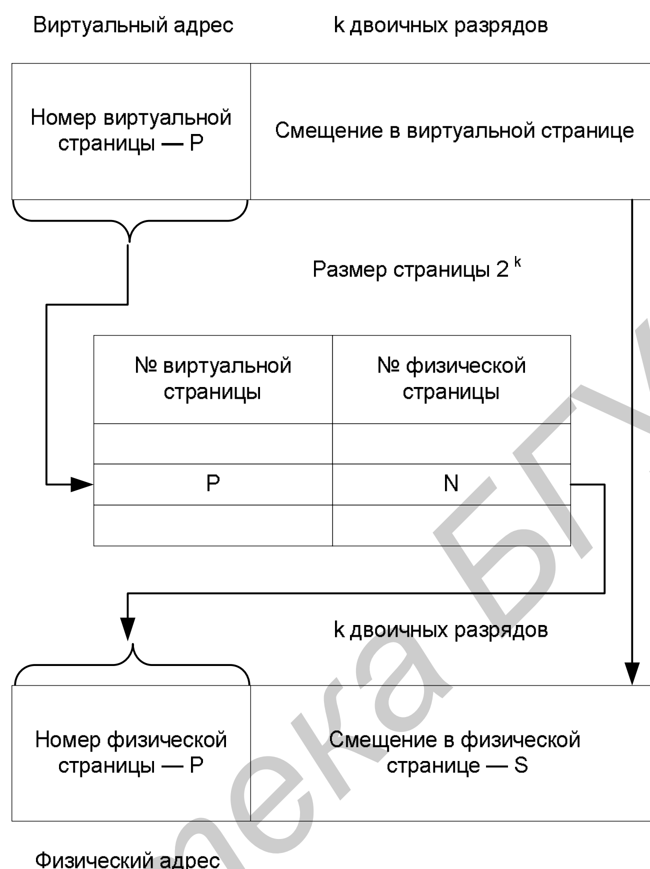


Рис. 2.12. Механизм преобразования виртуального адреса в физический при страничной организации памяти

При каждом обращении к оперативной памяти аппаратными средствами выполняются следующие действия:

- на основании начального адреса таблицы страниц (содержимое регистра адреса таблицы страниц), номера виртуальной страницы (старшие разряды виртуального адреса) и длины записи в таблице страниц (системная константа) определяется адрес нужной записи в таблице;
- из этой записи извлекается номер физической страницы;
- к номеру физической страницы присоединяется смещение (младшие разряды виртуального адреса).

Использование того факта, что размер страницы равен степени 2, позволяет применить операцию конкатенации (присоединения) вместо более длительной операции сложения, что уменьшает время получения физического адреса, а значит, повышает производительность компьютера.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

На производительность системы со страничной организацией памяти влияют временные затраты, связанные с обработкой страничных прерываний и преобразованием виртуального адреса в физический. При часто возникающих страничных прерываниях система может тратить большую часть времени впустую, на свопинг страниц. Чтобы уменьшить частоту страничных прерываний, следовало бы увеличивать размер страницы. Кроме того, увеличение размера страницы уменьшает размер таблицы страниц, а значит, уменьшает затраты памяти. С другой стороны, если страница велика, значит, велика и фиктивная область в последней виртуальной странице каждой программы. В среднем на каждой программе теряется половина объема страницы, что в сумме при большой странице может составить существенную величину. Время преобразования виртуального адреса в физический в значительной степени определяется временем доступа к таблице страниц. В связи с этим таблицу страниц стремятся размещать в "быстрых" запоминающих устройствах. Это может быть, например, набор специальных регистров или память, использующая для уменьшения времени доступа ассоциативный поиск и кэширование данных.

Страничное распределение памяти может быть реализовано в упрощенном варианте, без выгрузки страниц на диск. В этом случае все виртуальные страницы всех процессов постоянно находятся в оперативной памяти. Такой вариант страничной организации хотя и не предоставляет пользователю виртуальной памяти, но почти исключает фрагментацию за счет того, что программа может загружаться в несмежные области, а также того, что при загрузке виртуальных страниц никогда не образуется остатков.

2.3.8. Сегментное распределение

При страничной организации виртуальное адресное пространство процесса делится механически на равные части. Это не позволяет дифференцировать способы доступа к разным частям программы (сегментам), а такая дифференциация часто бывает очень полезной. Например, можно запретить обращаться с операциями записи и чтения в кодовый сегмент программы, а для сегмента данных разрешить только чтение. Кроме того, разбиение программы на "осмысленные" части делает принципиально возможным разделение одного сегмента несколькими процессами. Например, если два процесса используют одну и ту же математическую подпрограмму, то в оперативную память может быть загружена только одна копия этой подпрограммы.

Рассмотрим, каким образом сегментное распределение памяти реализует эти возможности (рис. 2.13). Виртуальное адресное пространство процесса делится на сегменты, размер которых определяется программистом с учетом смыслового значения содержащейся в них информации. Отдельный сегмент может представлять собой подпрограмму, массив данных и т.п. Иногда сегментация программы выполняется по умолчанию компилятором.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

При загрузке процесса часть сегментов помещается в оперативную память (при этом для каждого из этих сегментов операционная система подыскивает подходящий участок свободной памяти), а часть сегментов размещается в дисковой памяти. Сегменты одной программы могут занимать в оперативной памяти несмежные участки. Во время загрузки система создает таблицу сегментов процесса (аналогичную таблице страниц), в которой для каждого сегмента указывается начальный физический адрес сегмента в оперативной памяти, размер сегмента, правила доступа, признак модификации, признак обращения к данному сегменту за последний интервал времени и некоторая другая информация. Если виртуальные адресные пространства нескольких процессов включают один и тот же сегмент, то в таблицах сегментов этих процессов делаются ссылки на один и тот же участок оперативной памяти, в который данный сегмент загружается в единственном экземпляре.

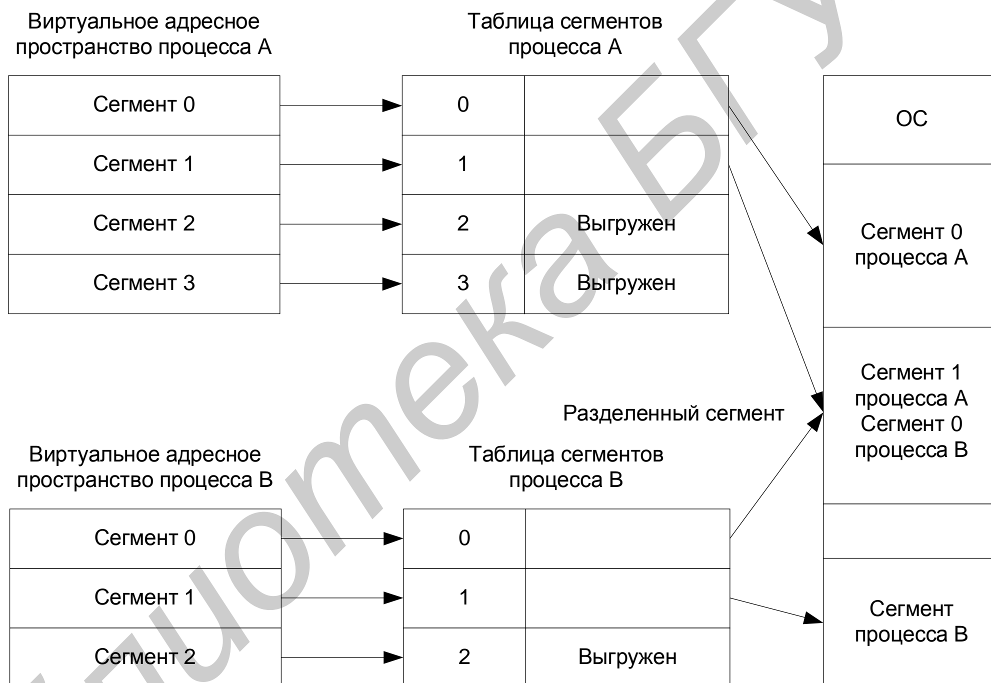


Рис. 2.13. Распределение памяти сегментами

Система с сегментной организацией функционирует аналогично системе со страничной организацией: время от времени происходят прерывания, связанные с отсутствием нужных сегментов в памяти, при необходимости освобождения памяти некоторые сегменты выгружаются, при каждом обращении к оперативной памяти выполняется преобразование виртуального адреса в физический. Кроме того, при обращении к памяти проверяется, разрешен ли доступ требуемого типа к данному сегменту.

Виртуальный адрес при сегментной организации памяти может быть представлен парой (g, s), где g — номер сегмента, а s — смещение в сегменте. Физический адрес получается путем сложения начального физического адреса сегмента, найденного в таблице сегментов по номеру g, и смещения s.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Недостатком данного метода распределения памяти является фрагментация на уровне сегментов и более медленное по сравнению со страничной организацией преобразование адреса.

2.3.9. Странично-сегментное распределение

Как видно из названия, данный метод представляет собой комбинацию страничного и сегментного распределения памяти и, вследствие этого, сочетает в себе достоинства обоих подходов. Виртуальное пространство процесса делится на сегменты, а каждый сегмент в свою очередь делится на виртуальные страницы, которые нумеруются в пределах сегмента. Оперативная память делится на физические страницы. Загрузка процесса выполняется операционной системой постранично, при этом часть страниц размещается в оперативной памяти, а часть — на диске. Для каждого сегмента создается своя таблица страниц, структура которой полностью совпадает со структурой таблицы страниц, используемой при страничном распределении. Для каждого процесса создается таблица сегментов, в которой указываются адреса таблиц страниц для всех сегментов данного процесса. Адрес таблицы сегментов загружается в специальный регистр процессора, когда активизируется соответствующий процесс. На рис. 2.14 показана схема преобразования виртуального адреса в физический для данного метода.

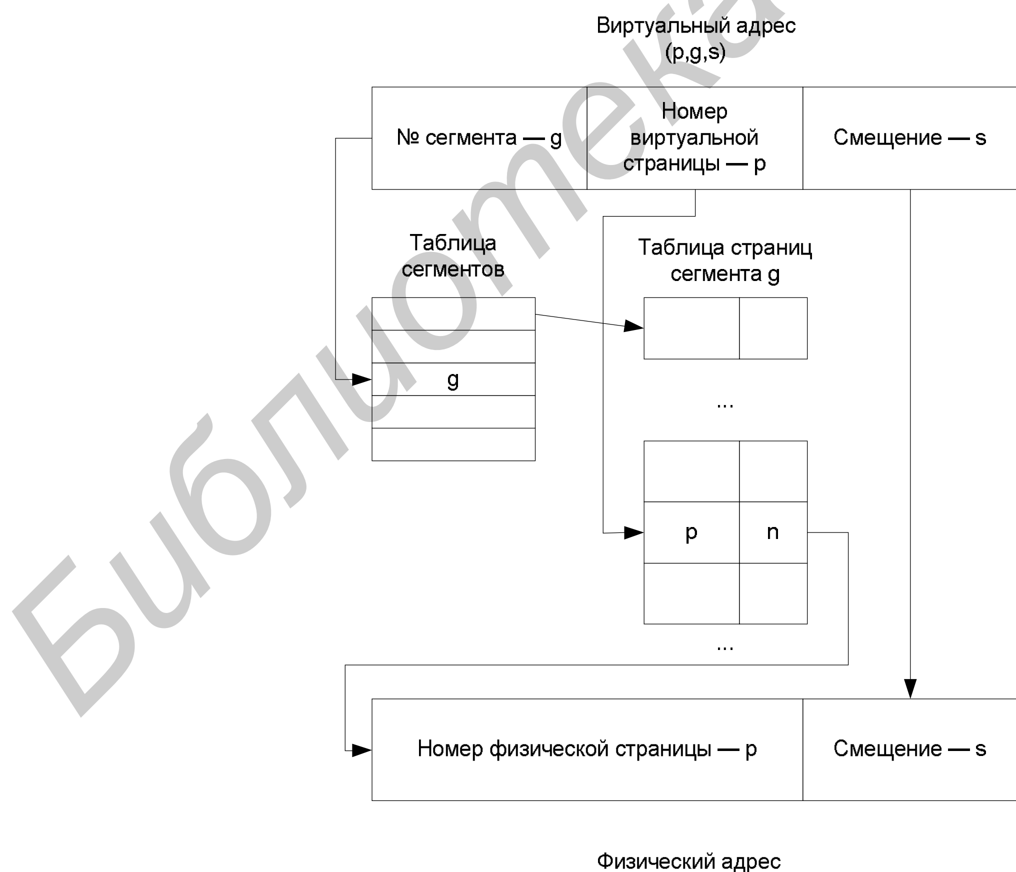


Рис. 2.14. Схема преобразования виртуального адреса в физический для сегментно-страничной организации памяти

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

2.3.10. Свопинг

Разновидностью виртуальной памяти является свопинг. На рис. 2.15 показан график зависимости коэффициента загрузки процессора от числа одновременно выполняемых процессов и доли времени, проводимого этими процессами в состоянии ожидания ввода-вывода.

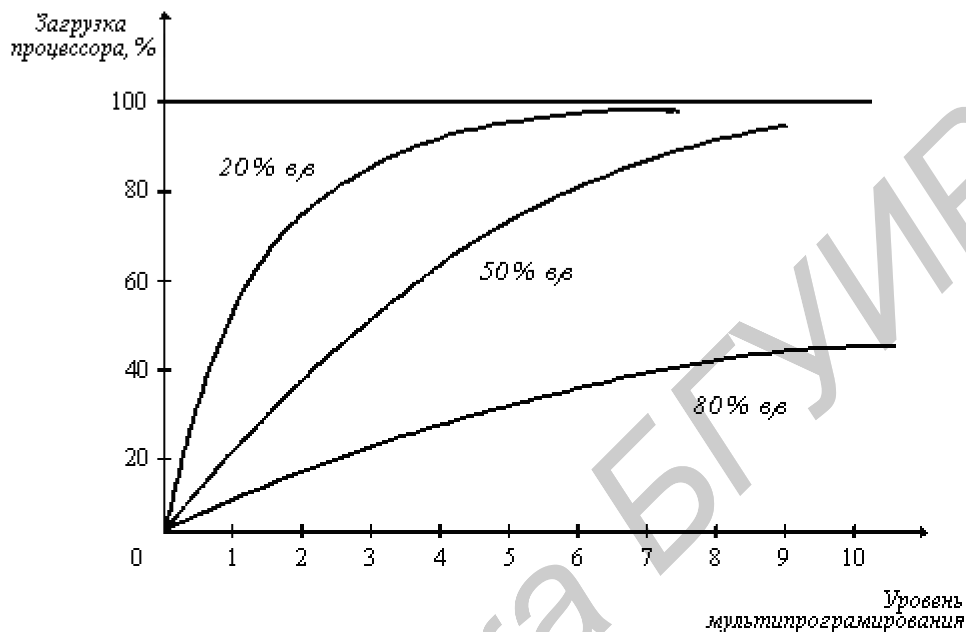


Рис. 2.15. Зависимость загрузки процессора от числа задач и интенсивности ввода-вывода

Из рисунка видно, что для загрузки процессора на 90 % достаточно всего трех счетных задач. Однако для того, чтобы обеспечить такую же загрузку интерактивными задачами, выполняющими интенсивный ввод-вывод, потребуются десятки таких задач. Необходимым условием для выполнения задачи является загрузка ее в оперативную память, объем которой ограничен. В этих условиях был предложен метод организации вычислительного процесса, называемый свопингом. В соответствии с этим методом некоторые процессы (обычно находящиеся в состоянии ожидания) временно выгружаются на диск. Планировщик операционной системы не исключает их из своего рассмотрения, и при наступлении условий активизации некоторого процесса, находящегося в области свопинга на диске, этот процесс перемещается в оперативную память. Если свободного места в оперативной памяти не хватает, то выгружается другой процесс.

При свопинге, в отличие от рассмотренных ранее методов реализации виртуальной памяти, процесс перемещается между памятью и диском целиком, то есть в течение некоторого времени процесс может полностью отсутствовать в оперативной памяти. Существуют различные алгоритмы выбора процессов на загрузку и выгрузку, а также различные способы выделения оперативной и дисковой памяти загружаемому процессу.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

2.3.11. Иерархия запоминающих устройств. Принцип кэширования данных

Память вычислительной машины представляет собой иерархию запоминающих устройств (внутренние регистры процессора, различные типы сверхоперативной и оперативной памяти, диски, ленты), отличающихся средним временем доступа и стоимостью хранения данных в расчете на один бит (рис. 2.16). Пользователю хотелось бы иметь и недорогую и быструю память. Кэш-память представляет некоторое компромиссное решение этой проблемы.

Определение 2.7. Кэш-память – это способ организации совместного функционирования двух типов запоминающих устройств, отличающихся временем доступа и стоимостью хранения данных, который позволяет уменьшить среднее время доступа к данным за счет динамического копирования в "быстрое" ЗУ наиболее часто используемой информации из "медленного" ЗУ.

Кэш-памятью часто называют не только способ организации работы двух типов запоминающих устройств, но и одно из устройств — "быстрое" ЗУ. Оно стоит дороже и, как правило, имеет сравнительно небольшой объем. Важно, что механизм кэш-памяти является прозрачным для пользователя, который не должен сообщать никакой информации об интенсивности использования данных и не должен никак участвовать в перемещении данных из ЗУ одного типа в ЗУ другого типа — все это делается автоматически системными средствами.

Библиотека БГУИР

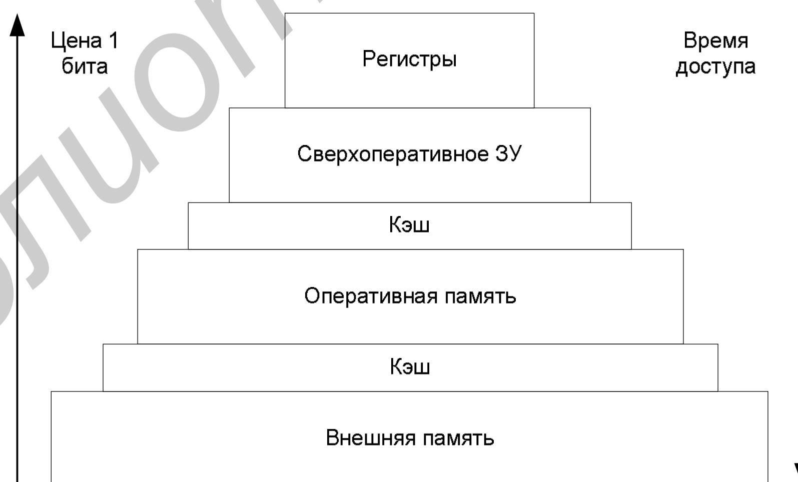
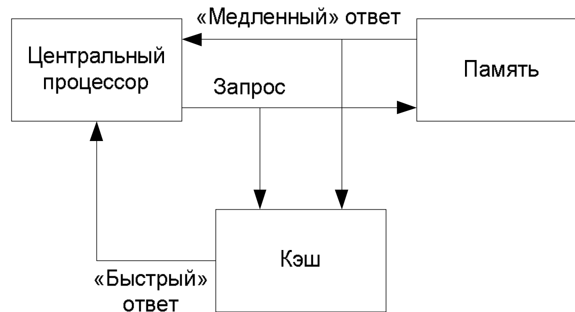


Рис. 2.16. Иерархия ЗУ

Рассмотрим частный случай использования кэш-памяти для уменьшения среднего времени доступа к данным, хранящимся в оперативной памяти. Для этого между процессором и оперативной памятью помещается быстрое ЗУ, называемое просто кэш-памятью (рис. 2.17). В качестве такового может быть использована, например, ассоциативная память. Содержимое кэш-памяти представляет собой совокупность записей обо

PDF created with pdfFactory Pro trial version www.pdffactory.com

всех загруженных в нее элементах данных. Каждая запись об элементе данных включает в себя адрес, который этот элемент данных имеет в оперативной памяти, и управляющую информацию: признак модификации и признак обращения к данным за некоторый последний период времени.



Структура кэш памяти

Адрес данных в ОП	Данные	Управляющая информация	
		Бит модиф.	Бит обращ.

Рис. 2.17. Кэш-память

В системах, оснащенных кэш-памятью, каждый запрос к оперативной памяти выполняется в соответствии со следующим алгоритмом:

1. Просматривается содержимое кэш-памяти, чтобы определить нахождение в ней нужных данных; кэш-память не является адресуемой, поэтому поиск нужных данных осуществляется по содержимому — значению поля "адрес в оперативной памяти", взятому из запроса.
2. Если данные обнаруживаются в кэш-памяти, то они считываются из нее и результат передается в процессор.
3. Если нужных данных нет, то они вместе со своим адресом копируются из оперативной памяти в кэш-память и результат выполнения запроса передается в процессор. При копировании данных может оказаться, что в кэш-памяти нет свободного места, тогда выбираются данные, к которым в последний период было меньше всего обращений, для вытеснения из кэш-памяти. Если вытесняемые данные были модифицированы за время нахождения в кэш-памяти, то они переписываются в оперативную память. Если же эти данные не были модифицированы, то их место в кэш-памяти объявляется свободным.

На практике в кэш-память считывается не один элемент данных, к которому произошло обращение, а целый блок данных; это увеличивает вероятность так называемого "попадания в кэш", то есть нахождения нужных данных в кэш-памяти.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Покажем, как среднее время доступа к данным зависит от вероятности попадания в кэш. Пусть имеется основное запоминающие устройство со средним временем доступа к данным t1 и кэш-память, имеющая время доступа t2, очевидно, что $t2 < t1$. Обозначим через t среднее время доступа к данным в системе с кэш-памятью, а через p — вероятность попадания в кэш. По формуле полной вероятности имеем: $t = t1((1 - p) + t2(p))$.

Из нее видно, что среднее время доступа к данным в системе с кэш-памятью линейно зависит от вероятности попадания в кэш и изменяется от среднего времени доступа в основное ЗУ (при $p = 0$) до среднего времени доступа непосредственно в кэш-память (при $p = 1$).

В реальных системах вероятность попадания в кэш составляет примерно 0,9. Высокое значение вероятности нахождения данных в кэш-памяти связано с наличием у данных объективных свойств: пространственной и временной локальности.

- Пространственная локальность. Если произошло обращение по некоторому адресу, то с высокой степенью вероятности в ближайшее время произойдет обращение к соседним адресам.
- Временная локальность. Если произошло обращение по некоторому адресу, то следующее обращение по этому же адресу с большой вероятностью произойдет в ближайшее время.

Все предыдущие рассуждения справедливы и для других пар запоминающих устройств, например, для оперативной памяти и внешней памяти. В этом случае уменьшается среднее время доступа к данным, расположенным на диске, и роль кэш-памяти выполняет буфер в оперативной памяти.

2.4. Управление вводом-выводом

Одной из главных функций ОС является управление всеми устройствами ввода-вывода компьютера. ОС должна передавать устройствам команды, перехватывать прерывания и обрабатывать ошибки; она также должна обеспечивать интерфейс между устройствами и остальной частью системы. В целях развития интерфейс должен быть одинаковым для всех типов устройств (независимость от устройств).

2.4.1. Физическая организация устройств ввода-вывода

Устройства ввода-вывода делятся на два типа: блок-ориентированные и байт-ориентированные. Блок-ориентированные устройства хранят информацию в блоках фиксированного размера, каждый из которых имеет свой собственный адрес. Самое распространенное блок-ориентированное устройство — диск. Байт-ориентированные устройства не адресуемы и не позволяют производить операцию поиска, они генерируют или потребляют последовательность байтов. Примерами являются терминалы, строчные принтеры, сетевые адаптеры. Однако некоторые внешние устройства не

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

относятся ни к одному классу, например, часы, которые, с одной стороны, не адресуемы, а с другой — не порождают потока байтов. Это устройство только выдает сигнал прерывания в некоторые моменты времени.

Внешнее устройство обычно состоит из механического и электронного компонента. Электронный компонент называется контроллером устройства или адаптером. Механический компонент представляет собой собственно устройство. Некоторые контроллеры могут управлять несколькими устройствами. Если интерфейс между контроллером и устройством стандартизован, то независимые производители могут выпускать совместимые как контроллеры, так и устройства.

Операционная система обычно имеет дело не с устройством, а с контроллером. Контроллер, как правило, выполняет простые функции, например, преобразует поток бит в блоки, состоящие из байт, и осуществляют контроль и исправление ошибок. Каждый контроллер имеет несколько регистров, которые используются для взаимодействия с центральным процессором. В некоторых компьютерах эти регистры являются частью физического адресного пространства. В таких компьютерах нет специальных операций ввода-вывода. В других компьютерах адреса регистров ввода-вывода, называемых часто портами, образуют собственное адресное пространство за счет введения специальных операций ввода-вывода (например команд IN и OUT в процессорах i86).

ОС выполняет ввод-вывод, записывая команды в регистры контроллера. Например, контроллер гибкого диска IBM PC принимает 15 команд, таких, как READ, WRITE, SEEK, FORMAT и т.д. Когда команда принята, процессор оставляет контроллер и занимается другой работой. При завершении команды контроллер организует прерывание, для того чтобы передать управление процессором операционной системе, которая должна проверить результаты операции. Процессор получает результаты и статус устройства, читая информацию из регистров контроллера.

2.4.2. Организация программного обеспечения ввода-вывода

Основная идея организации программного обеспечения ввода-вывода состоит в разбиении его на несколько уровней, причем нижние уровни обеспечивают экранирование особенностей аппаратуры от верхних, а те, в свою очередь, обеспечивают удобный интерфейс для пользователей.

Ключевым принципом является независимость от устройств. Вид программы не должен зависеть от того, читает ли она данные с гибкого или с жесткого диска.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Очень близкой к идее независимости от устройств является идея единообразного именования, то есть для именования устройств должны быть приняты единые правила.

Другим важным вопросом для программного обеспечения ввода-вывода является обработка ошибок. Вообще говоря, ошибки следует обрабатывать как можно ближе к аппаратуре. Если контроллер обнаруживает ошибку чтения, то он должен попытаться ее скорректировать. Если же это ему не удается, то исправлением ошибок должен заняться драйвер устройства. Многие ошибки могут исчезать при повторных попытках выполнения операций ввода-вывода, например, ошибки, вызванные наличием пылинок на головках чтения или на диске. И только если нижний уровень не может справиться с ошибкой, он сообщает об ошибке верхнему уровню.

Еще один ключевой вопрос — это использование блокирующих (синхронных) и неблокирующих (асинхронных) передач. Большинство операций физического ввода-вывода выполняется асинхронно: процессор начинает передачу и переходит на другую работу, пока не наступает прерывание. Пользовательские программы намного легче писать, если операции ввода-вывода блокирующие: после команды READ программа автоматически приостанавливается до тех пор, пока данные не попадут в буфер программы. ОС выполняет операции ввода-вывода асинхронно, но представляет их для пользовательских программ в синхронной форме.

Последняя проблема состоит в том, что одни устройства являются разделяемыми, а другие — выделенными. Диски — это разделяемые устройства, так как одновременный доступ нескольких пользователей к диску не представляет собой проблему. Принтеры — это выделенные устройства, потому что нельзя смешивать строчки, печатаемые различными пользователями. Наличие выделенных устройств создает для операционной системы некоторые проблемы.

Для решения поставленных проблем целесообразно разделить программное обеспечение ввода-вывода на четыре слоя (рис. 2.18):

- обработка прерываний;
- драйверы устройств;
- независимый от устройств слой операционной системы;
- пользовательский слой программного обеспечения.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

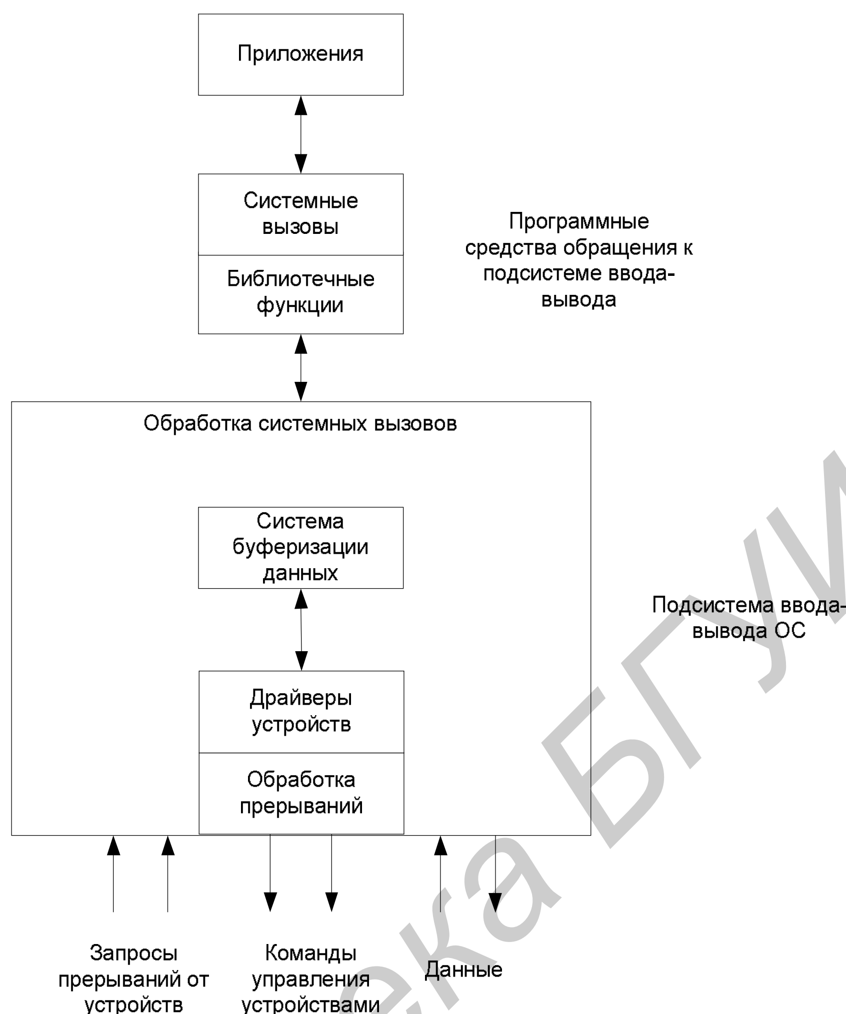


Рис. 2.18. Многоуровневая организация подсистемы ввода-вывода

2.4.3. Обработка прерываний

Прерывания должны быть скрыты как можно глубже в недрах операционной системы, чтобы как можно меньшая часть ОС имела с ними дело. Наилучший способ состоит в разрешении процессу, инициировавшему операцию ввода-вывода, блокировать себя до завершения операции и наступления прерывания. Процесс может блокировать себя, используя, например, вызов DOWN для семафора, или вызов WAIT для переменной условия, или вызов RECEIVE для ожидания сообщения. При наступлении прерывания процедура обработки прерывания выполняет разблокирование процесса, инициировавшего операцию ввода-вывода, используя вызовы UP, SIGNAL или посылая процессу сообщение. В любом случае эффект от прерывания будет состоять в том, что ранее заблокированный процесс теперь продолжит свое выполнение.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

2.4.4. Драйверы устройств

Весь зависимый от устройства код помещается в драйвер устройства. Каждый драйвер управляет устройствами одного типа или, может быть, одного класса.

В операционной системе только драйвер устройства знает о конкретных особенностях какого-либо устройства. Например, только драйвер диска имеет дело с дорожками, секторами, цилиндрами, временем установления головки и другими факторами, обеспечивающими правильную работу диска.

Драйвер устройства принимает запрос от устройств программного слоя и решает, как его выполнить. Типичным запросом является чтение n блоков данных. Если драйвер был свободен во время поступления запроса, то он начинает выполнять запрос немедленно. Если же он был занят обслуживанием другого запроса, то вновь поступивший запрос присоединяется к очереди уже имеющихся запросов и будет выполнен, когда наступит его очередь.

Первый шаг в реализации запроса ввода-вывода, например, для диска, состоит в преобразовании его из абстрактной формы в конкретную. Для дискового драйвера это означает преобразование номеров блоков в номера цилиндров, головок, секторов, проверку, работает ли мотор, находится ли головка над нужным цилиндром. Короче говоря, он должен решить, какие операции контроллера нужно выполнить и в какой последовательности.

После передачи команды контроллеру драйвер должен решить, блокировать ли себя до окончания заданной операции или нет. Если операция занимает значительное время, как при печати некоторого блока данных, то драйвер блокируется до тех пор, пока операция не завершится и обработчик прерывания не разблокирует его. Если команда ввода-вывода выполняется быстро (например, прокрутка экрана), то драйвер ожидает ее завершения без блокирования.

2.4.5. Независимый от устройств слой операционной системы

Большая часть программного обеспечения ввода-вывода является независимой от устройств. Точная граница между драйверами и независимыми от устройств программами определяется системой, так как некоторые функции, которые могли бы быть реализованы независимым способом, в действительности выполнены в виде драйверов для повышения эффективности или по другим причинам.

Типичными функциями для независимого от устройств слоя являются:

- обеспечение общего интерфейса к драйверам устройств;
- именование устройств;
- защита устройств;
- обеспечение независимого размера блока;

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

- буферизация;
- распределение памяти на блок-ориентированных устройствах;
- распределение и освобождение выделенных устройств;
- уведомление об ошибках.

Остановимся на некоторых функциях данного перечня. Верхним слоям программного обеспечения неудобно работать с блоками разной величины, поэтому данный слой обеспечивает единый размер блока, например, за счет объединения нескольких различных блоков в единый логический блок. В связи с этим верхние уровни имеют дело с абстрактными устройствами, которые используют единый размер логического блока независимо от размера физического сектора.

При создании файла или заполнении его новыми данными необходимо выделить ему новые блоки. Для этого ОС должна вести список или битовую карту свободных блоков диска. На основании информации о наличии свободного места на диске может быть разработан алгоритм поиска свободного блока, независимый от устройства и реализуемый программным слоем, находящимся выше слоя драйверов.

2.4.6. Пользовательский слой программного обеспечения

Хотя большая часть программного обеспечения ввода-вывода находится внутри ОС, некоторая его часть содержится в библиотеках, связываемых с пользовательскими программами. Системные вызовы, включающие вызовы ввода-вывода, обычно делаются библиотечными процедурами. Если программа, написанная на языке С, содержит вызов

count = write (fd, buffer, nbytes),

то библиотечная процедура write будет связана с программой. Набор подобных процедур является частью системы ввода-вывода. В частности, форматирование ввода или вывода выполняется библиотечными процедурами. Примером может служить функция printf языка С, которая принимает строку формата и, возможно, некоторые переменные в качестве входной информации, затем строит строку символов ASCII и делает вызов write для вывода этой строки. Стандартная библиотека ввода-вывода содержит большое число процедур, которые выполняют ввод-вывод и работают как часть пользовательской программы.

Другой категорией программного обеспечения ввода-вывода является подсистема спулинга (spooling). Спулинг — это способ работы с выделенными устройствами в мультипрограммной системе. Рассмотрим типичное устройство, требующее спулинга — строчный принтер. Хотя технически легко позволить каждому пользовательскому процессу открыть специальный файл, связанный с принтером, такой способ опасен из-за того, что пользовательский процесс может монополизировать принтер на

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

произвольное время. Вместо этого создается специальный процесс-монитор, который получает исключительные права на использование этого устройства. Также создается специальный каталог, называемый каталогом спулинга. Для того чтобы напечатать файл, пользовательский процесс помещает выводимую информацию в этот файл и помещает его в каталог спулинга. Процесс-монитор по очереди распечатывает все файлы, содержащиеся в каталоге спулинга.

2.5. Файловая система

Определение 2.8. Файловая система – это часть операционной системы, назначение которой состоит в том, чтобы обеспечить пользователю удобный интерфейс при работе с данными, хранящимися на диске, и обеспечить совместное использование файлов несколькими пользователями и процессами.

В широком смысле понятие "файловая система" включает:

- совокупность всех файлов на диске;
- наборы структур данных, используемых для управления файлами, такие, например, как каталоги файлов, дескрипторы файлов, таблицы распределения свободного и занятого пространства на диске;
- комплекс системных программных средств, реализующих управление файлами, в частности: создание, уничтожение, чтение, запись, именование, поиск и другие операции над файлами.

2.5.1. Имена файлов

Файлы идентифицируются именами. Пользователи дают файлам символьные имена, при этом учитываются ограничения ОС как на используемые символы, так и на длину имени. До недавнего времени эти границы были весьма узкими. Так, в популярной файловой системе FAT длина имен ограничивается известной схемой 8.3 (8 символов — собственно имя, 3 символа — расширение имени), а в ОС UNIX System V имя не может содержать более 14 символов. Однако пользователю гораздо удобнее работать с длинными именами, поскольку они позволяют дать файлу действительно мнемоническое название, по которому даже через достаточно большой промежуток времени можно будет вспомнить, что содержит этот файл. Поэтому современные файловые системы, как правило, поддерживают длинные символьные имена файлов. Например, Windows NT в своей новой файловой системе NTFS устанавливает, что имя файла может содержать до 255 символов, не считая завершающего нулевого символа.

При переходе к длинным именам возникает проблема совместимости с ранее созданными приложениями, использующими короткие имена. Чтобы приложения могли обращаться к файлам в соответствии с принятыми ранее

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

соглашениями, файловая система должна уметь предоставлять эквивалентные короткие имена (псевдонимы) файлам, имеющим длинные имена. Таким образом, одной из важных задач становится проблема генерации соответствующих коротких имен.

Длинные имена поддерживаются не только новыми файловыми системами, но и новыми версиями хорошо известных файловых систем. Например, в ОС Windows 95 используется файловая система VFAT, представляющая собой существенно измененный вариант FAT. Среди многих других усовершенствований одним из главных достоинств VFAT является поддержка длинных имен. Кроме проблемы генерации эквивалентных коротких имен, при реализации нового варианта FAT важной задачей была задача хранения длинных имен при условии, что принципиально метод хранения и структура данных на диске не должны были измениться.

Обычно разные файлы могут иметь одинаковые символьные имена. В этом случае файл однозначно идентифицируется так называемым составным именем, представляющем собой последовательность символьных имен каталогов. В некоторых системах одному и тому же файлу не может быть дано несколько разных имен, а в других такое ограничение отсутствует. В последнем случае операционная система присваивает файлу дополнительно уникальное имя, так чтобы можно было установить взаимно-однозначное соответствие между файлом и его уникальным именем. Уникальное имя представляет собой числовой идентификатор и используется программами операционной системы. Примером такого уникального имени файла является номер индексного дескриптора в системе UNIX.

2.5.2. Типы файлов

Файлы бывают разных типов: обычные, специальные и файлы-каталоги.

Обычные файлы в свою очередь подразделяются на текстовые и двоичные. Текстовые файлы состоят из строк символов, представленных в ASCII-коде. Это могут быть документы, исходные тексты программ и т.п. Текстовые файлы можно прочитать на экране и распечатать на принтере. Двоичные файлы не используют ASCII-коды, они часто имеют сложную внутреннюю структуру, например, объектный код программы или архивный файл. Все операционные системы должны уметь распознавать хотя бы один тип файлов — их собственные исполняемые файлы.

Специальные файлы — это файлы, ассоциированные с устройствами ввода-вывода, которые позволяют пользователю выполнять операции ввода-вывода, используя обычные команды записи в файл или чтения из файла. Эти команды обрабатываются вначале программами файловой системы, а затем на некотором этапе выполнения запроса преобразуются ОС в команды управления соответствующим устройством. Специальные файлы, так же как и устройства ввода-вывода, делятся на блок-ориентированные и байт-ориентированные.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Каталог — это, с одной стороны, группа файлов, объединенных пользователем исходя из некоторых соображений (например, файлы, содержащие программы игр, или файлы, составляющие один программный пакет), а с другой стороны — это файл, содержащий системную информацию о группе файлов, его составляющих. В каталоге содержится список файлов, входящих в него, и устанавливается соответствие между файлами и их характеристиками (атрибутами).

В разных файловых системах могут использоваться в качестве атрибутов разные характеристики, например:

- информация о разрешенном доступе;
- пароль для доступа к файлу;
- владелец файла;
- создатель файла;
- признак "только для чтения";
- признак "скрытый файл";
- признак "системный файл";
- признак "архивный файл";
- признак "двоичный/символьный";
- признак "временный" (удалить после завершения процесса);
- признак блокировки;
- длина записи;
- указатель на ключевое поле в записи;
- длина ключа;
- времена создания, последнего доступа и последнего изменения;
- текущий размер файла;
- максимальный размер файла.

Каталоги могут непосредственно содержать значения характеристик файлов, как это сделано в файловой системе MS-DOS, или ссылаться на таблицы, содержащие эти характеристики, как это реализовано в ОС UNIX (рис. 2.19). Каталоги могут образовывать иерархическую структуру за счет того, что каталог более низкого уровня может входить в каталог более высокого уровня (рис. 2.20).

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Рис. 2.19. Структура каталогов: а — структура записи каталога MS-DOS (32 байта);
б — структура записи каталога ОС UNIX

Иерархия каталогов может быть деревом или сетью. Каталоги образуют дерево, если файлу разрешено входить только в один каталог, и сеть, если файл может входить сразу в несколько каталогов. В MS-DOS каталоги образуют древовидную структуру, а в UNIX'е — сетевую. Как и любой другой файл, каталог имеет символьное имя и однозначно идентифицируется составным именем, содержащим цепочку символьных имен всех каталогов, через которые проходит путь от корня до данного каталога.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

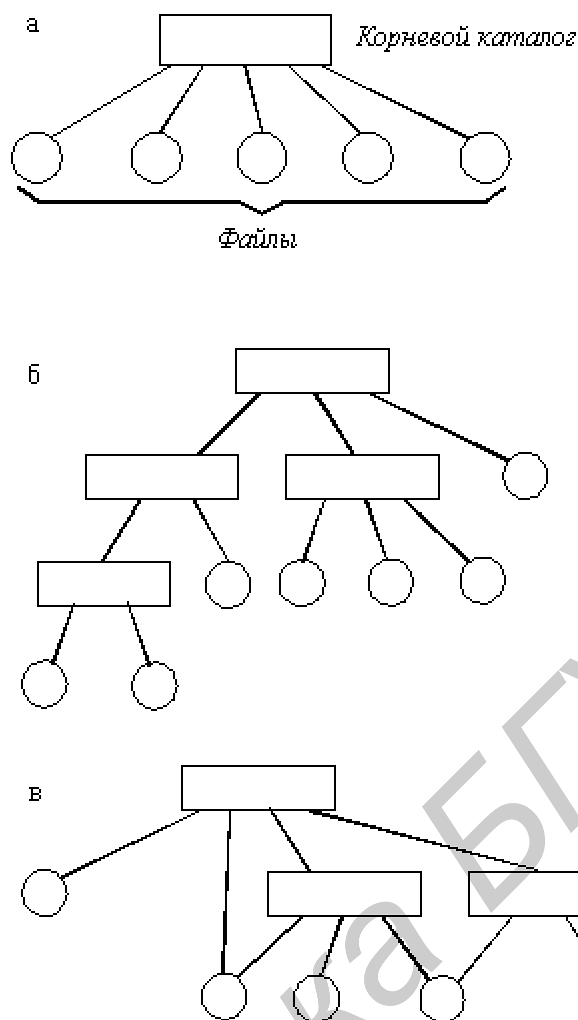


Рис. 2.20. Логическая организация файловой системы
а — одноуровневая; б — иерархическая (дерево); в — иерархическая (сеть)

2.5.3. Логическая организация файла

Программист имеет дело с логической организацией файла, представляя файл в виде определенным образом организованных логических записей. Логическая запись — это наименьший элемент данных, которым может оперировать программист при обмене с внешним устройством. Даже если физический обмен с устройством осуществляется большими единицами, операционная система обеспечивает программисту доступ к отдельной логической записи. На рис. 2.21 показаны несколько схем логической организации файла. Записи могут быть фиксированной или переменной длины. Они могут быть расположены в файле последовательно (последовательная организация) или в более сложном порядке, с использованием так называемых индексных таблиц, позволяющих обеспечить быстрый доступ к отдельной логической записи (индексно-последовательная организация). Для идентификации записи может быть использовано специальное поле записи, называемое ключом. В файловых системах ОС UNIX и MS-DOS файл имеет простейшую логическую структуру — последовательность однобайтовых записей.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Индекс	1	2	3
Адрес	213	321	122

Рис. 2.21. Способы логической организации файлов

2.5.4. Физическая организация и адрес файла

Физическая организация файла описывает правила расположения файла на устройстве внешней памяти, в частности на диске. Файл состоит из физических записей — блоков. Блок — наименьшая единица данных, которой внешнее устройство обменивается с оперативной памятью. Непрерывное размещение — простейший вариант физической организации (рис. 2.34, а), при котором файлу предоставляется последовательность блоков диска, образующих единый сплошной участок дисковой памяти. Для задания адреса файла в этом случае достаточно указать только номер начального блока. Другое достоинство этого метода — простота. Но имеются и два существенных недостатка. Во-первых, во время создания файла заранее не известна его длина, а значит, неизвестно, сколько памяти надо зарезервировать для этого файла, во-вторых, при таком порядке размещения неизбежно возникает фрагментация, и пространство на диске используется неэффективно, так как отдельные участки маленького размера (минимально 1 блок) могут остаться неиспользуемыми.

Следующий способ физической организации — размещение в виде связанного списка блоков дисковой памяти (рис. 2.22, б). При таком способе в начале каждого блока содержится указатель на следующий блок. В этом случае адрес файла также может быть задан одним числом — номером первого блока. В отличие от предыдущего способа, каждый блок может быть присоединен в цепочку какого-либо файла, следовательно, фрагментация отсутствует. Файл может изменяться во время своего существования, наращивая число блоков. Недостатком является сложность реализации доступа к произвольно заданному месту файла: для того, чтобы

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

прочитать пятый по порядку блок файла, необходимо последовательно прочитать четыре первых блока, прослеживая цепочку номеров блоков. Кроме того, при этом способе количество данных файла, содержащихся в одном блоке, не равно степени двойки (одно слово израсходовано на номер следующего блока), а многие программы читают данные блоками, размер которых равен степени двойки.

Популярным способом, используемым, например, в файловой системе FAT операционной системы MS-DOS, является использование связанного списка индексов. С каждым блоком связывается некоторый элемент — индекс. Индексы располагаются в отдельной области диска (в MS-DOS это таблица FAT). Если некоторый блок распределен некоторому файлу, то индекс этого блока содержит номер следующего блока данного файла. При такой физической организации сохраняются все достоинства предыдущего способа, но снимаются оба отмеченных недостатка: во-первых, для доступа к произвольному месту файла достаточно прочитать только блок индексов, отсчитать нужное количество блоков файла по цепочке и определить номер нужного блока, и, во-вторых, данные файла занимают блок целиком, а значит, имеют объем, равный степени двойки.

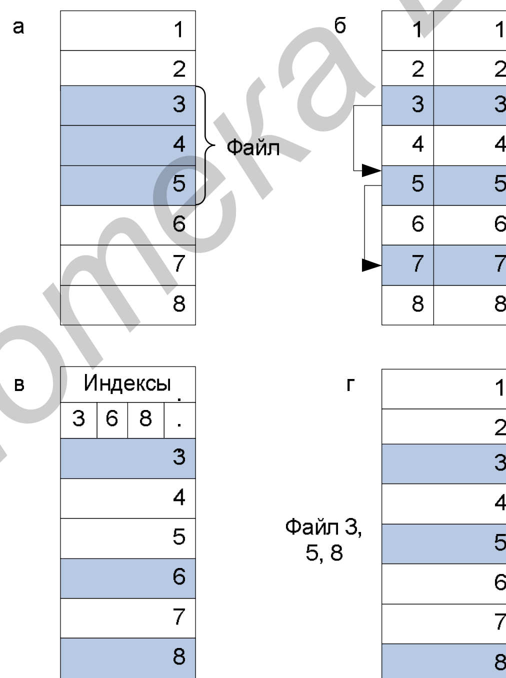


Рис. 2.22. Физическая организация файла
а — непрерывное размещение; б — связанный список блоков;
в — связанный список индексов; г — перечень номеров блоков

В заключение рассмотрим задание физического расположения файла путем простого перечисления номеров блоков, занимаемых этим файлом. ОС UNIX использует вариант данного способа, позволяющий обеспечить фиксированную длину адреса, независимо от размера файла. Для хранения адреса файла выделено 13 полей. Если размер файла меньше или равен 10 блокам, то номера этих блоков непосредственно перечислены в первых десяти полях адреса. Если размер файла больше 10 блоков, то следующее,

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

11-е поле содержит адрес блока, в котором могут быть расположены еще 128 номеров следующих блоков файла. Если файл больше, чем 10+128 блоков, то используется 12-е поле, в котором находится номер блока, содержащего 128 номеров блоков, которые содержат по 128 номеров блоков данного файла. И, наконец, если файл больше 10+128+128(128), то используется последнее, 13-е поле для тройной косвенной адресации, что позволяет задать адрес файла, имеющего размер максимум 10+ 128 + 128(128 + 128(128(128))).

2.5.5. Права доступа к файлу

Определить права доступа к файлу — значит определить для каждого пользователя набор операций, которые он может применить к данному файлу. В разных файловых системах может быть определен свой список дифференцируемых операций доступа. Этот список может включать следующие операции:

- создание файла;
- уничтожение файла;
- открытие файла;
- закрытие файла;
- чтение файла;
- запись в файл;
- дополнение файла;
- поиск в файле;
- получение атрибутов файла;
- установление новых значений атрибутов;
- переименование;
- выполнение файла;
- чтение каталога;
- и другие операции с файлами и каталогами.

В самом общем случае права доступа могут быть описаны матрицей прав доступа, в которой столбцы соответствуют всем файлам системы, строки — всем пользователям, а на пересечении строк и столбцов указываются разрешенные операции (рис. 2.23). В некоторых системах пользователи могут быть разделены на отдельные категории. Для всех пользователей одной категории определяются единые права доступа. Например, в системе UNIX все пользователи подразделяются на три категории: владельца файла, членов его группы и всех остальных.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

	Modern.txt	Win.exe	Class.dbf	Unix.ppt
Kira	читать	выполнять	-	выполнять
Genya	читать	выполнять	-	выполнять
Nataly	читать	-	-	выполнять читать
Roman	читать писать	-	-	

Рис. 2.23. Матрица прав доступа

Различают два основных подхода к определению прав доступа:

- избирательный доступ, когда для каждого файла и каждого пользователя сам владелец может определить допустимые операции;
- мандатный подход, когда система наделяет пользователя определенными правами по отношению к каждому разделяемому ресурсу (в данном случае файлу) в зависимости от того, к какой группе пользователь отнесен.

2.5.6. Кэширование диска

В некоторых файловых системах запросы к внешним устройствам, в которых адресация осуществляется блоками (диски, ленты), перехватываются промежуточным программным слоем — подсистемой буферизации. Подсистема буферизации представляет собой буферный пул, располагающийся в оперативной памяти, и комплекс программ, управляющих этим пулом. Каждый буфер пула имеет размер, равный одному блоку. При поступлении запроса на чтение некоторого блока подсистема буферизации просматривает свой буферный пул, и если находит требуемый блок, то копирует его в буфер запрашивающего процесса. Операция ввода-вывода считается выполненной, хотя физического обмена с устройством не происходило. Очевиден выигрыш во времени доступа к файлу. Если же нужный блок в буферном пуле отсутствует, то он считывается с устройства и одновременно с передачей запрашивающему процессу копируется в один из буферов подсистемы буферизации. При отсутствии свободного буфера на диск вытесняется наименее используемая информация. Таким образом, подсистема буферизации работает по принципу кэш-памяти.

2.5.7. Общая модель файловой системы

Функционирование любой файловой системы можно представить многоуровневой моделью (рис. 2.24), в которой каждый уровень предоставляет некоторый интерфейс (набор функций) вышележащему

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

уровню, а сам, в свою очередь, для выполнения своей работы использует интерфейс (обращается с набором запросов) нижележащего уровня.

Задачей символьного уровня является определение по символьному имени файла его уникального имени. В файловых системах, в которых каждый файл может иметь только одно символьное имя (например MS-DOS), этот уровень отсутствует, так как символьное имя, присвоенное файлу пользователем, является одновременно уникальным и может быть использовано операционной системой. В других файловых системах, в которых один и тот же файл может иметь несколько символьных имен, на данном уровне просматривается цепочка каталогов для определения уникального имени файла. В файловой системе UNIX, например, уникальным именем является номер индексного дескриптора файла (i-node).

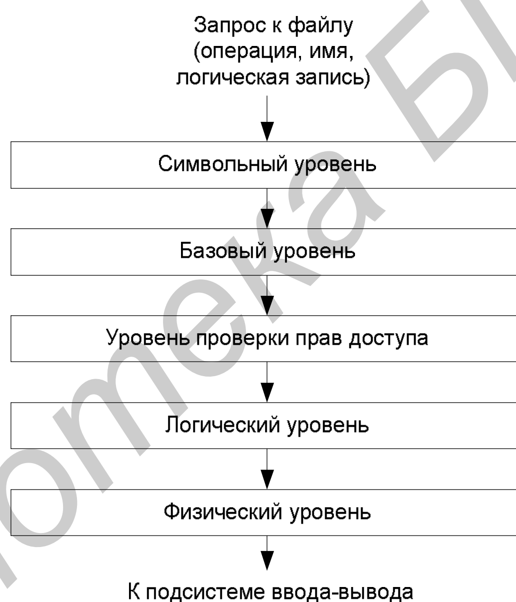


Рис. 2.24. Общая модель файловой системы

На следующем, базовом уровне по уникальному имени файла определяются его характеристики: права доступа, адрес, размер и др. Как уже было сказано, характеристики файла могут входить в состав каталога или храниться в отдельных таблицах. При открытии файла его характеристики перемещаются с диска в оперативную память, чтобы уменьшить среднее время доступа к файлу. В некоторых файловых системах (например HPFS) при открытии файла вместе с его характеристиками в оперативную память перемещаются несколько первых блоков файла, содержащих данные.

Следующим этапом реализации запроса к файлу является проверка прав доступа к нему. Для этого сравниваются полномочия пользователя или процесса, выдавших запрос, со списком разрешенных видов доступа к

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

данному файлу. Если запрашиваемый вид доступа разрешен, то выполнение запроса продолжается, если нет, то выдается сообщение о нарушении прав доступа.

На логическом уровне определяются координаты запрашиваемой логической записи в файле, то есть требуется определить, на каком расстоянии (в байтах) от начала файла находится требуемая логическая запись. При этом абстрагируются от физического расположения файла, он представляется в виде непрерывной последовательности байт. Алгоритм работы данного уровня зависит от логической организации файла. Например, если файл организован как последовательность логических записей фиксированной длины l, то n-я логическая запись имеет смещение l(n-1) байт. Для определения координат логической записи в файле с индексно-последовательной организацией выполняется чтение таблицы индексов (ключей), в которой непосредственно указывается адрес логической записи.

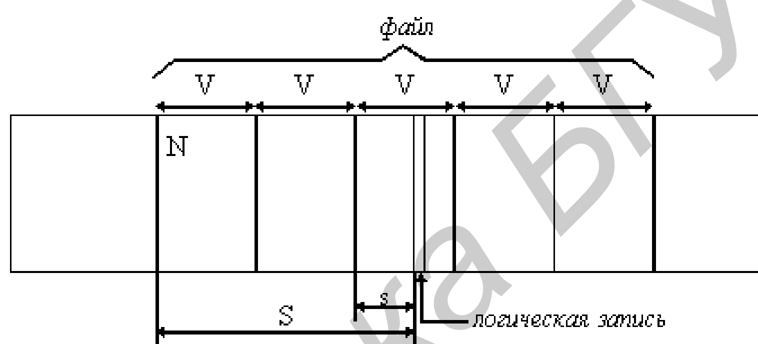


Рис. 2.25. Функции физического уровня файловой системы

Исходные данные:
V — размер блока;
N — номер первого блока файла;
S — смещение логической записи в файле.

Требуется определить на физическом уровне:

- n — номер блока, содержащего требуемую логическую запись;
- s — смещение логической записи в пределах блока.

n = N + [S/V], где [S/V] — целая часть числа S/V;
s = R [S/V] — дробная часть числа S/V.

На физическом уровне файловая система определяет номер физического блока, который содержит требуемую логическую запись, и смещение логической записи в физическом блоке. Для решения этой задачи используются результаты работы логического уровня — смещение логической записи в файле, адрес файла на внешнем устройстве, а также сведения о физической организации файла, включая размер блока. Рис. 2.25 иллюстрирует работу физического уровня для простейшей физической организации файла в виде непрерывной последовательности блоков. Подчеркнем, что задача физического уровня решается независимо от того, как был логически организован файл.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

После определения номера физического блока, файловая система обращается к системе ввода-вывода для выполнения операции обмена с внешним устройством. В ответ на этот запрос в буфер файловой системы будет передан нужный блок, в котором на основании полученного при работе физического уровня смещения выбирается требуемая логическая запись.

2.5.8. Отображаемые в память файлы

По сравнению с доступом к памяти традиционный доступ к файлам выглядит запутанным и неудобным. По этой причине некоторые ОС, начиная с MULTICS, обеспечивают отображение файлов в адресное пространство выполняемого процесса. Это выражается в появлении двух новых системных вызовов: MAP (отобразить) и UNMAP (отменить отображение). Первый вызов передает операционной системе в качестве параметров имя файла и виртуальный адрес, и операционная система отображает указанный файл в виртуальное адресное пространство по указанному адресу.

Предположим, например, что файл *f* имеет длину 64 К и отображается на область виртуального адресного пространства с начальным адресом 512 К. После этого любая машинная команда, которая читает содержимое байта по адресу 512 К, получает нулевой байт этого файла и т.д. Очевидно, что запись по адресу 512 К + 1100 изменяет 1100 байт файла. При завершении процесса на диске остается модифицированная версия файла, как если бы он был изменен комбинацией вызовов SEEK и WRITE.

В действительности при отображении файла внутренние системные таблицы изменяются так, чтобы данный файл служил хранилищем страниц виртуальной памяти на диске. Таким образом, чтение по адресу 512 К вызывает страничный отказ, в результате чего страница 0 переносится в физическую память. Аналогично запись по адресу 512 К + 1100 вызывает страничный отказ, в результате которого страница, содержащая этот адрес, перемещается в память, после чего осуществляется запись в память по требуемому адресу. Если эта страница вытесняется из памяти алгоритмом замены страниц, то она записывается обратно в файл в соответствующее его место. При завершении процесса все отображенные и модифицированные страницы переписываются из памяти в файл.

Отображение файлов лучше всего работает в системе, которая поддерживает сегментацию. В такой системе каждый файл может быть отображен в свой собственный сегмент, так что k-й байт в файле является k-м байтом сегмента. На рис. 2.26а изображен процесс, который имеет два сегмента-кода и данных. Предположим, что этот процесс копирует файлы. Для этого он сначала отображает файл-источник, например abc. Затем он создает пустой сегмент и отображает на него файл назначения, например файл ddd.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

С этого момента процесс может копировать сегмент-источник в сегмент-приемник с помощью обычного программного цикла, использующего команды пересылки в памяти типа *mov*. Никакие вызовы READ или WRITE не нужны. После выполнения копирования процесс может выполнить вызов UNMAP для удаления файла из адресного пространства, а затем завершиться. Выходной файл ddd будет существовать на диске, как если бы он был создан обычным способом.

Хотя отображение файлов исключает потребность в выполнении ввода-вывода и тем самым облегчает программирование, этот способ порождает и некоторые новые проблемы. Во-первых, для системы сложно узнать точную длину выходного файла, в данном примере ddd. Проще указать наибольший номер записанной страницы, но нет способа узнать, сколько байт в этой странице было записано. Предположим, что программа использует только страницу номер 0 и после выполнения все байты все еще установлены в значение 0 (их начальное значение). Быть может, файл состоит из 10 нулей. А может быть, он состоит из 100 нулей. Как это определить? Операционная система не может это сообщить. Все, что она может сделать, так это создать файл, длина которого равна размеру страницы.

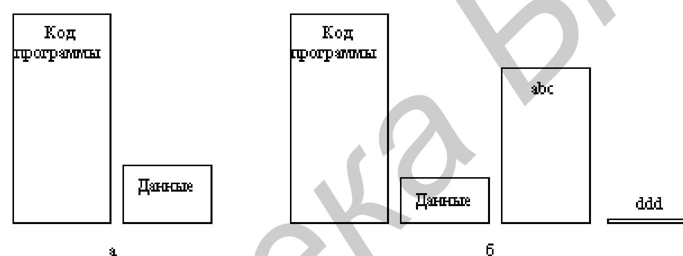


Рис. 2.26. (а) Сегменты процесса перед отображением файлов в адресное пространство; (б) Процесс после отображения существующего файла abc в один сегмент и создания нового сегмента для файла ddd

Вторая проблема проявляется (потенциально), если один процесс отображает файл, а другой процесс открывает его для обычного файлового доступа. Если первый процесс изменяет страницу, то это изменение не будет отражено в файле на диске до тех пор, пока страница не будет вытеснена на диск. Поддержание согласованности данных файла для этих двух процессов требует от системы больших забот.

Третья проблема состоит в том, что файл может быть больше, чем сегмент, и даже больше, чем все виртуальное адресное пространство. Единственный способ ее решения состоит в реализации вызова MAP таким образом, чтобы он мог отображать не весь файл, а его часть. Хотя такая работа, очевидно, менее удобна, чем отображение целого файла.

2.5.9. Современные архитектуры файловых систем

Разработчики новых операционных систем стремятся обеспечить пользователя возможностью работать сразу с несколькими файловыми системами. В новом понимании файловая система состоит из многих составляющих, в число которых входят и файловые системы в традиционном понимании.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Новая файловая система имеет многоуровневую структуру (рис. 2.27), на верхнем уровне которой располагается так называемый переключатель файловых систем (в Windows 95, например, такой переключатель называется устанавливаемым диспетчером файловой системы — installable filesystem manager, IFS). Он обеспечивает интерфейс между запросами приложения и конкретной файловой системой, к которой обращается это приложение. Переключатель файловых систем преобразует запросы в формат, воспринимаемый следующим уровнем — уровнем файловых систем.

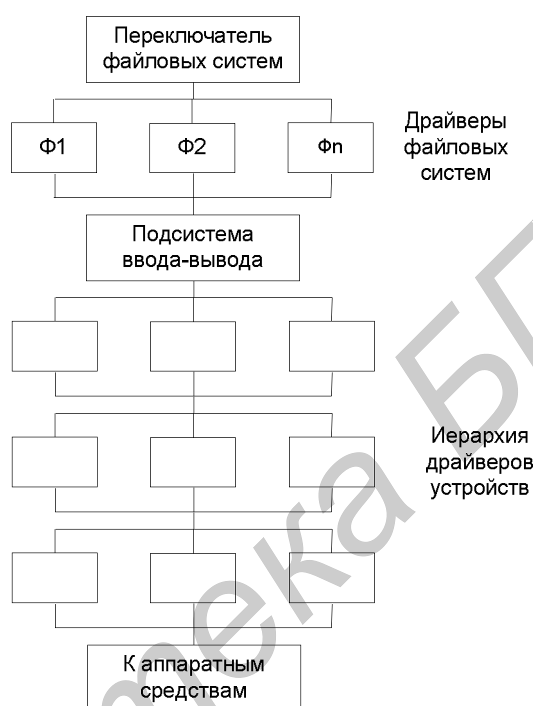


Рис. 2.27. Архитектура современной файловой системы

Каждый компонент уровня файловых систем выполнен в виде драйвера соответствующей файловой системы и поддерживает определенную организацию файловой системы. Переключатель является единственным модулем, который может обращаться к драйверу файловой системы. Приложение не может обращаться к нему напрямую. Драйвер файловой системы может быть написан в виде реентерабельного кода, что позволяет сразу нескольким приложениям выполнять операции с файлами. Каждый драйвер файловой системы в процессе собственной инициализации регистрируется у переключателя, передавая ему таблицу точек входа, которые будут использоваться при последующих обращениях к файловой системе.

Для выполнения своих функций драйверы файловых систем обращаются к подсистеме ввода-вывода, образующей следующий слой файловой системы новой архитектуры. Подсистема ввода-вывода — это составная часть файловой системы, которая отвечает за загрузку, инициализацию и управление всеми модулями низших уровней файловой системы. Обычно эти модули представляют собой драйверы портов, которые непосредственно занимаются работой с аппаратными средствами. Кроме

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

этого, подсистема ввода-вывода обеспечивает некоторый сервис драйверам файловой системы, что позволяет им осуществлять запросы к конкретным устройствам. Подсистема ввода-вывода должна постоянно присутствовать в памяти и организовывать совместную работу иерархии драйверов устройств. В эту иерархию могут входить драйверы устройств определенного типа (драйверы жестких дисков или накопителей на лентах), драйверы, поддерживаемые поставщиками (такие драйверы перехватывают запросы к блочным устройствам и могут частично изменить поведение существующего драйвера этого устройства, например зашифровать данные), драйверы портов, которые управляют конкретными адаптерами.

Большое число уровней архитектуры файловой системы обеспечивает авторам драйверов устройств большую гибкость: драйвер может получить управление на любом этапе выполнения запроса — от вызова приложением функции, которая занимается работой с файлами, до того момента, когда работающий на самом низком уровне драйвер устройства начинает просматривать регистры контроллера. Многоуровневый механизм работы файловой системы реализован посредством цепочек вызова.

В ходе инициализации драйвер устройства может добавить себя к цепочке вызова некоторого устройства, определив при этом уровень последующего обращения. Подсистема ввода-вывода помещает адрес целевой функции в цепочку вызова устройства, используя заданный уровень для того, чтобы должным образом упорядочить цепочку. По мере выполнения запроса подсистема ввода-вывода последовательно вызывает все функции, ранее помещенные в цепочку вызова.

Внесенная в цепочку вызова процедура драйвера может решить передать запрос дальше — в измененном или в неизмененном виде — на следующий уровень, или, если это возможно, процедура может удовлетворить запрос, не передавая его дальше по цепочке.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

2.6. Контрольные вопросы

4. Что является единицей работы в многонитевых операционных системах?
5. Поясните употребление терминов *программа*, *процесс*, *задача*, *поток* и *нить*.
6. В чем состоит принципиальное отличие состояний *ожидание* и *готовности*?
7. Чем объясняется потенциально более высокая надежность операционных систем, в которых реализована вытесняющая многозадачность?
8. Можно ли задачу планирования процессов целиком возложить на приложения?
9. Чем ограничивается максимальный размер физической памяти, которую можно установить в компьютере определенной модели?
10. Чем ограничивается максимальный размер виртуального адресного пространства, доступного приложению?
11. В каких случаях транслятор создает объектный код программы не в виртуальных, а в физических адресах?
12. Как величина файла подкачки влияет на производительность системы?
13. На что влияет размер страницы? Каковы преимущества и недостатки большого размера страницы?
14. Где хранятся таблицы страниц и таблицы сегментов?
15. Чем определяется количество таблиц сегментов, имеющихся в операционной системе в произвольный момент времени?
16. Какие характеристики содержит таблица сегментов и таблица страниц при сегментно-страничной организации памяти?
17. Почему загрузка и выгрузка данных из кэш-памяти производится блоками?
18. Как обеспечивается согласование данных в кэше с помощью методов обратной и сквозной записи?
19. Какие функции выполняет подсистема ввода-вывода?
20. Какой набор методов организации файловой системы подходит для накопителя на магнитной ленте?

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

3. Управление распределенными ресурсами

3.1. Базовые примитивы передачи сообщений в распределенных системах

Единственным по-настоящему важным отличием распределенных систем от централизованных является межпроцессная взаимосвязь. В централизованных системах связь между процессами, как правило, предполагает наличие разделяемой памяти. Типичный пример — проблема "поставщик-потребитель", в этом случае один процесс пишет в разделяемый буфер, а другой — читает из него. Даже наиболее простая форма синхронизации — семафор — требует, чтобы хотя бы одно слово (переменная самого семафора) было разделяемым. В распределенных системах нет какой бы то ни было разделяемой памяти, таким образом, вся природа межпроцессных коммуникаций должна быть продумана заново.

Основой этого взаимодействия может служить только передача по сети сообщений. В самом простом случае системные средства обеспечения связи могут быть сведены к двум основным системным вызовам (примитивам): один — для посылки сообщения, другой — для получения сообщения. В дальнейшем на их базе могут быть построены более мощные средства сетевых коммуникаций, такие как распределенная файловая система или вызов удаленных процедур, которые, в свою очередь, также могут служить основой для построения других сетевых сервисов.

Несмотря на концептуальную простоту этих системных вызовов — *послать* и *получить* — существуют различные варианты их реализации, от правильного выбора которых зависит эффективность работы сети. В частности, эффективность коммуникаций в сети зависит от способа задания адреса, от того, является ли системный вызов блокирующим или неблокирующим, какие выбраны способы буферизации сообщений и насколько надежным является протокол обмена сообщениями.

3.1.1. Способы адресации

Для того чтобы послать сообщение, необходимо указать адрес получателя. В очень простой сети адрес может задаваться в виде константы, но в более сложных сетях нужен и более изощренный способ адресации.

Одним из вариантов адресации на верхнем уровне является использование физических адресов сетевых адаптеров. Если в получающем компьютере выполняется только один процесс, то ядро будет знать, что делать с поступившим сообщением — передать его этому процессу. Однако, если на машине выполняется несколько процессов, то ядру неизвестно, какому из них предназначено сообщение, поэтому использование сетевого адреса адаптера в качестве адреса получателя приводит к очень серьезному ограничению: на каждой машине должен выполняться только один процесс.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Альтернативная адресная система использует имена назначения, состоящие из двух частей, определяющих номер машины и номер процесса. Однако адресация типа "машина-процесс" далека от идеала, в частности, она негибка и непрозрачна, так как пользователь должен явно задавать адрес машины-получателя. В этом случае, если в один прекрасный день машина, на которой работает сервер, отказывает, то программа, в которой жестко используется адрес сервера, не сможет работать с другим сервером, установленном на другой машине.

Другим вариантом могло бы быть назначение каждому процессу уникального адреса, который никак не связан с адресом машины. Одним из способов достижения этой цели является использование централизованного механизма распределения адресов процессов, который работает просто, как счетчик. При получении запроса на выделение адреса он просто возвращает текущее значение счетчика, а затем наращивает его на единицу. Недостатком этой схемы является то, что централизованные компоненты, подобные этому, не обеспечивают в достаточной степени расширяемость систем.

Еще один метод назначения процессам уникальных идентификаторов заключается в разрешении каждому процессу выбора своего собственного идентификатора из очень большого адресного пространства, такого как пространство 64-битных целых чисел. Вероятность выбора одного и того же числа двумя процессами является ничтожной, а система хорошо расширяется. Однако здесь имеется одна проблема: как процесс-отправитель может узнать номер машины процесса-получателя. В сети, которая поддерживает широковещательный режим (то есть в ней предусмотрен такой адрес, который принимают все сетевые адаптеры), отправитель может широковещательно передать специальный пакет, который содержит идентификатор процесса назначения. Все ядра получат эти сообщения, проверят адрес процесса и, если он совпадает с идентификатором одного из процессов этой машины, пошлют ответное сообщение "Я здесь", содержащее сетевой адрес машины.

Хотя эта схема и прозрачна, но широковещательные сообщения перегружают систему. Такой перегрузки можно избежать, выделив в сети специальную машину для отображения высокоуровневых символьных имен. При применении такой системы процессы адресуются с помощью символьных строк, и в программы вставляются эти строки, а не номера машин или процессов. Каждый раз перед первой попыткой связаться процесс должен послать запрос специальному отображающему процессу, обычно называемому сервером имен, запрашивая номер машины, на которой работает процесс-получатель.

Совершенно иной подход — использование специальной аппаратуры. Пусть процессы выбирают свои адреса случайно, а конструкция сетевых адаптеров позволяет хранить эти адреса. Теперь адреса процессов не

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

обнаруживаются путем широковещательной передачи, а непосредственно указываются в кадрах, заменяя там адреса сетевых адаптеров.

3.1.2. Блокирующие и неблокирующие примитивы

Примитивы бывают блокирующими и неблокирующими, иногда они называются соответственно синхронными и асинхронными. При использовании блокирующего примитива процесс, выдавший запрос на его выполнение, приостанавливается до полного завершения примитива. Например, вызов примитива *получить* приостанавливает вызывающий процесс до получения сообщения.

При использовании неблокирующего примитива управление возвращается вызывающему процессу немедленно, еще до того, как требуемая работа будет выполнена. Преимуществом этой схемы является параллельное выполнение вызывающего процесса и процесса передачи сообщения. Обычно в ОС имеется один из двух видов примитивов и очень редко — оба. Однако выигрыш в производительности при использовании неблокирующих примитивов компенсируется серьезным недостатком: отправитель не может модифицировать буфер сообщения, пока сообщение не отправлено, а узнать, отправлено ли сообщение, отправитель не может. Отсюда сложности в построении программ, которые передают последовательность сообщений с помощью неблокирующих примитивов.

Имеется два возможных выхода. Первое решение — это заставить ядро копировать сообщение в свой внутренний буфер, а затем разрешить процессу продолжить выполнение. С точки зрения процесса эта схема ничем не отличается от схемы блокирующего вызова: как только процесс снова получает управление, он может повторно использовать буфер.

Второе решение заключается в прерывании процесса-отправителя после отправки сообщения, чтобы проинформировать его, что буфер снова доступен. Здесь не требуется копирование, что экономит время, но прерывание пользовательского уровня делает программирование запутанным, сложным, может привести к возникновению гонок.

Вопросом, тесно связанным с блокирующими и неблокирующими вызовами, является вопрос тайм-аутов. В системе с блокирующим вызовом *послать* при отсутствии ответа вызывающий процесс может заблокироваться навсегда. Для предотвращения такой ситуации в некоторых системах вызывающий процесс может задать временной интервал, в течение которого он ждет ответ. Если за это время сообщение не поступает, вызов *послать* завершается с кодом ошибки.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

3.1.3. Буферизуемые и небуферизуемые примитивы

Примитивы, которые были описаны, являются небуферизуемыми примитивами. Это означает, что вызов *получить* сообщает ядру машины, на которой он выполняется, адрес буфера, в который следует поместить прибывающее для него сообщение.

Эта схема работает прекрасно при условии, что получатель выполняет вызов *получить* раньше, чем отправитель выполняет вызов *послать*. Вызов *получить* сообщает ядру машины, на которой выполняется, по какому адресу должно поступить ожидаемое сообщение и в какую область памяти необходимо его поместить. Проблема возникает тогда, когда вызов *послать* сделан раньше вызова *получить*. Каким образом сможет узнать ядро на машине получателя, какому процессу адресовано вновь поступившее сообщение, если их несколько? И как оно узнает, куда его скопировать?

Один из вариантов — просто отказаться от сообщения, позволить отправителю взять тайм-аут и надеяться, что получатель все-таки выполнит вызов *получить* перед повторной передачей сообщения. Этот подход несложен в реализации, но, к сожалению, отправитель (или, скорее, ядро его машины) может сделать несколько таких безуспешных попыток. Еще хуже то, что после достаточно большого числа безуспешных попыток ядро отправителя может сделать неправильный вывод об аварии на машине получателя или о неправильности использованного адреса.

Второй подход к этой проблеме заключается в том, чтобы хранить, хотя бы некоторое время, поступающие сообщения в ядре получателя на тот случай, что вскоре будет выполнен соответствующий вызов *получить*. Каждый раз, когда поступает такое "не ожидаемое" сообщение, включается таймер. Если заданный временной интервал истекает раньше, чем происходит соответствующий вызов *получить*, то сообщение теряется.

Хотя этот метод и уменьшает вероятность потери сообщений, он порождает проблему хранения и управления преждевременно поступившими сообщениями. Необходимы буферы, которые следует где-то размещать, освобождать, в общем, которыми нужно управлять. Концептуально простым способом управления буферами является определение новой структуры данных, называемой почтовым ящиком.

Процесс, который заинтересован в получении сообщений, обращается к ядру с запросом о создании для него почтового ящика и сообщает адрес, по которому ему могут поступать сетевые пакеты, после чего все сообщения с данным адресом будут помещены в его почтовый ящик. Такой способ часто называют буферизуемым примитивом.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

3.1.4. Надежные и ненадежные примитивы

Ранее подразумевалось, что когда отправитель посылает сообщение, адресат его обязательно получает. Но реально сообщения могут теряться. Предположим, что используются блокирующие примитивы. Когда отправитель посылает сообщение, то он приостанавливает свою работу до тех пор, пока сообщение не будет послано. Однако нет никаких гарантий, что после того, как он возобновит свою работу, сообщение будет доставлено адресату.

Для решения этой проблемы существует три подхода. Первый заключается в том, что система не берет на себя никаких обязательств по поводу доставки сообщений. Реализация надежного взаимодействия становится целиком заботой пользователя.

Второй подход заключается в том, что ядро принимающей машины посылает квитанцию-подтверждение ядру отправляющей машины на каждое сообщение. Посылающее ядро разблокирует пользовательский процесс только после получения этого подтверждения. Подтверждение передается от ядра к ядру. Ни отправитель, ни получатель его не видят.

Третий подход заключается в использовании ответа в качестве подтверждения в тех системах, в которых запрос всегда сопровождается ответом. Отправитель остается заблокированным до получения ответа. Если ответа нет слишком долго, то посылающее ядро может переслать запрос специальной службе предотвращения потери сообщений.

3.2. Вызов удаленных процедур

3.2.1. Концепция удаленного вызова процедур

Идея вызова удаленных процедур *(Remote Procedure Call — RPC)* состоит в расширении хорошо известного и понятного механизма передачи управления и данных внутри программы, выполняющейся на одной машине, на передачу управления и данных через сеть. Средства удаленного вызова процедур предназначены для облегчения организации распределенных вычислений. Наибольшая эффективность использования RPC достигается в тех приложениях, в которых существует интерактивная связь между удаленными компонентами с небольшим временем ответов и относительно малым количеством передаваемых данных. Такие приложения называются RPC-ориентированными.

Характерными чертами вызова локальных процедур являются:

- асимметричность, то есть одна из взаимодействующих сторон является инициатором;
- синхронность, то есть выполнение вызывающей процедуры приостанавливается с момента выдачи запроса и возобновляется только после возврата из вызываемой процедуры.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Реализация удаленных вызовов существенно сложнее реализации вызовов локальных процедур. Начнем с того, что поскольку вызывающая и вызываемая процедуры выполняются на разных машинах, то они имеют разные адресные пространства, и это создает проблемы при передаче параметров и результатов, особенно если машины не идентичны. Так как RPC не может рассчитывать на разделяемую память, то это означает, что параметры RPC не должны содержать указатели на ячейки нестековой памяти и что значения параметров должны копироваться с одного компьютера на другой. Следующим отличием RPC от локального вызова является то, что он обязательно использует нижележащую систему связи, однако это не должно быть явно видно ни в определении процедур, ни в самих процедурах. Удаленность вносит дополнительные проблемы. Выполнение вызывающей программы и вызываемой локальной процедуры в одной машине реализуется в рамках единого процесса. Но в реализации RPC участвуют как минимум два процесса — по одному в каждой машине. В случае, если один из них аварийно завершится, могут возникнуть следующие ситуации: при аварии вызывающей процедуры удаленно вызванные процедуры станут "осиротевшими", а при аварийном завершении удаленных процедур станут "обездоленными родителями" вызывающие процедуры, которые будут безрезультатно ожидать ответа от удаленных процедур.

Кроме того, существует ряд проблем, связанных с неоднородностью языков программирования и операционных сред: структуры данных и структуры вызова процедур, поддерживаемые в каком-либо одном языке программирования, не поддерживаются точно так же во всех других языках.

Эти и некоторые другие проблемы решает широко распространенная технология RPC, лежащая в основе многих распределенных операционных систем.

3.2.2. Базовые операции удаленного вызова процедур

Чтобы понять работу RPC, рассмотрим вначале выполнение вызова локальной процедуры в обычной машине, работающей автономно. Пусть это, например, будет системный вызов

count=read (fd,buf,nbytes);

где fd — целое число;
buf — массив символов;
nbytes — целое число.

Чтобы осуществить вызов, вызывающая процедура заталкивает параметры в стек в обратном порядке (рис. 3.1). После того как вызов read выполнен, он помещает возвращаемое значение в регистр, перемещает адрес возврата

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

и возвращает управление вызывающей процедуре, которая выбирает параметры из стека, возвращая его в исходное состояние. Заметим, что в языке С параметры могут вызываться или по ссылке (by name), или по значению (by value). По отношению к вызываемой процедуре параметры-значения являются инициализируемыми локальными переменными. Вызываемая процедура может изменить их, и это не повлияет на значение оригиналов этих переменных в вызывающей процедуре.

Если в вызываемую процедуру передается указатель на переменную, то изменение значения этой переменной вызываемой процедурой влечет изменение значения этой переменной и для вызывающей процедуры. Этот факт весьма существенен для RPC.

Существует также другой механизм передачи параметров, который не используется в языке С. Он называется call-by-copy/restore и состоит в необходимости копирования вызывающей программой переменных в стек в виде значений, а затем копирования назад после выполнения вызова поверх оригинальных значений вызывающей процедуры.

Решение о том, какой механизм передачи параметров использовать, принимается разработчиками языка. Иногда это зависит от типа передаваемых данных. В языке С, например, целые и другие скалярные данные всегда передаются по значению, а массивы — по ссылке.

Идея, положенная в основу RPC, состоит в том, чтобы сделать вызов удаленной процедуры выглядящим по возможности так же, как и вызов локальной процедуры. Другими словами — сделать RPC прозрачным: вызывающей процедуре не требуется знать, что вызываемая процедура находится на другой машине, и наоборот.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

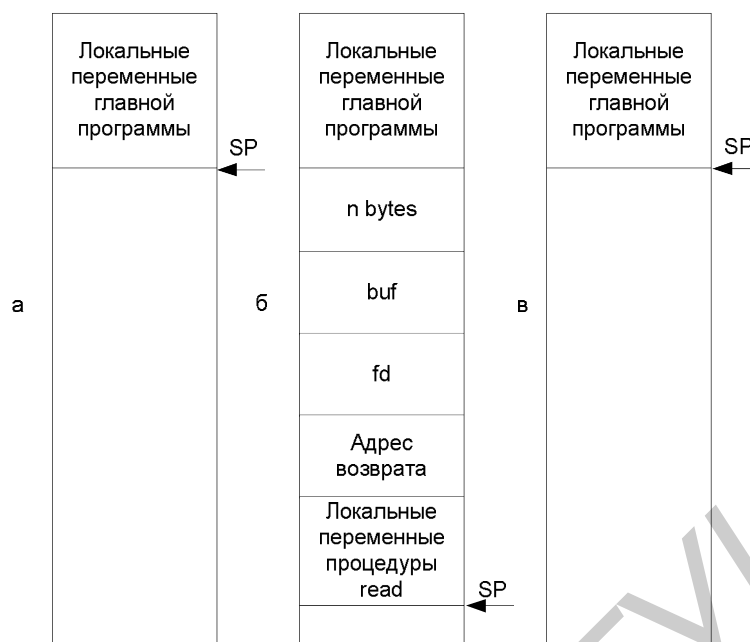


Рис. 3.1. Стек до выполнения вызова read (а); стек во время выполнения процедуры (б);
стек после возврата в вызывающую программу (в)

RPC достигает прозрачности следующим путем. Когда вызываемая процедура действительно является удаленной, в библиотеку помещается вместо локальной процедуры другая версия процедуры, называемая клиентским стабом (stub — заглушка). Подобно оригинальной процедуре, стаб вызывается с использованием вызывающей последовательности (как на рис. 3.1), так же происходит прерывание при обращении к ядру. Только в отличие от оригинальной процедуры он не помещает параметры в регистры и не запрашивает у ядра данные, вместо этого он формирует сообщение для отправки ядру удаленной машины.

3.2.3. Этапы выполнения удаленного вызова процедур

Взаимодействие программных компонентов при выполнении удаленного вызова процедуры иллюстрируется рис. 3.2. После того как клиентский стаб был вызван программой-клиентом, его первой задачей является заполнение буфера отправляемым сообщением. В некоторых системах клиентский стаб имеет единственный буфер фиксированной длины, заполняемый каждый раз с самого начала при поступлении каждого нового запроса. В других системах буфер сообщения представляет собой пул буферов для отдельных полей сообщения, причем некоторые из этих буферов уже заполнены. Этот метод особенно подходит для тех случаев, когда пакет имеет формат, состоящий из большого числа полей, но значения многих из этих полей не меняются от вызова к вызову.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Затем параметры должны быть преобразованы в соответствующий формат и вставлены в буфер сообщения. К этому моменту сообщение готово к передаче, поэтому выполняется прерывание по вызову ядра.

Когда ядро получает управление, оно переключает контексты, сохраняет регистры процессора и карту памяти (дескрипторы страниц), устанавливает новую карту памяти, которая будет использоваться для работы в режиме ядра. Поскольку контексты ядра и пользователя различаются, ядро должно точно скопировать сообщение в свое собственное адресное пространство, так чтобы иметь к нему доступ, запомнить адрес назначения (а, возможно, и другие поля заголовка), а также передать его сетевому интерфейсу. На этом завершается работа на клиентской стороне. Включается таймер передачи, и ядро может либо выполнять циклический опрос наличия ответа, либо передать управление планировщику, который выберет какой-либо другой процесс на выполнение. В первом случае ускоряется выполнение запроса, но отсутствует мультипрограммирование.

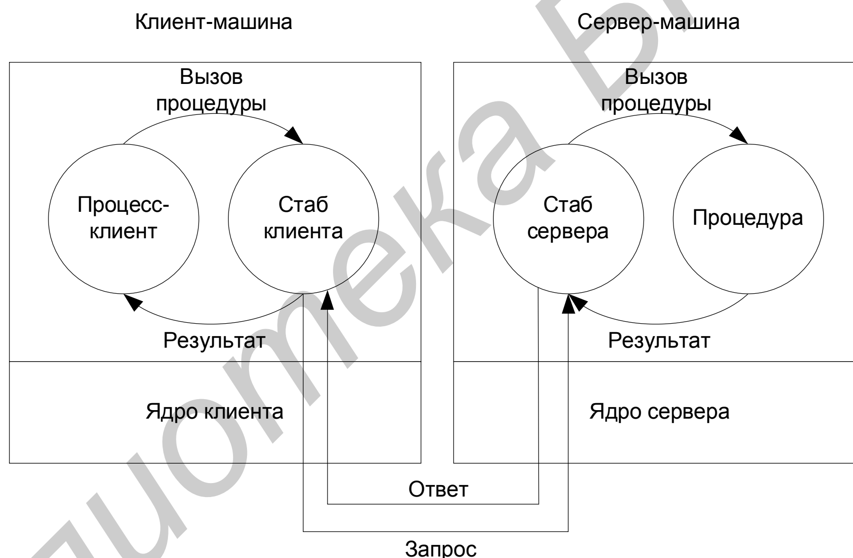


Рис. 3.2. Remote Procedure Call

На стороне сервера поступающие биты помещаются принимающей аппаратурой либо во встроенный буфер, либо в оперативную память. Когда вся информация будет получена, генерируется прерывание. Обработчик прерывания проверяет правильность данных пакета и определяет, какому стабу следует их передать. Если ни один из стабов не ожидает этот пакет, обработчик должен либо поместить его в буфер, либо вообще отказаться от него. Если имеется ожидающий стаб, то сообщение копируется ему. Наконец, выполняется переключение контекстов, в результате чего восстанавливаются регистры и карта памяти, принимая те значения, которые они имели в момент, когда стаб сделал вызов receive.

Теперь начинает работу серверный стаб. Он распаковывает параметры и помещает их соответствующим образом в стек. Когда все готово,

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

выполняется вызов сервера. После выполнения процедуры сервер передает результаты клиенту. Для этого выполняются все описанные выше этапы, только в обратном порядке.

Рис. 3.3 показывает последовательность команд, которую необходимо выполнить для каждого RPC-вызова, а рис. 3.4 — какая доля общего времени выполнения RPC тратится на выполнение каждого их описанных 14 этапов. Исследования были проведены на мультипроцессорной рабочей станции DEC Firefly, и, хотя наличие пяти процессоров обязательно повлияло на результаты измерений, приведенная на рисунке гистограмма дает общее представление о процессе выполнения RPC.

Рис. 3.3. Этапы выполнения процедуры RPC

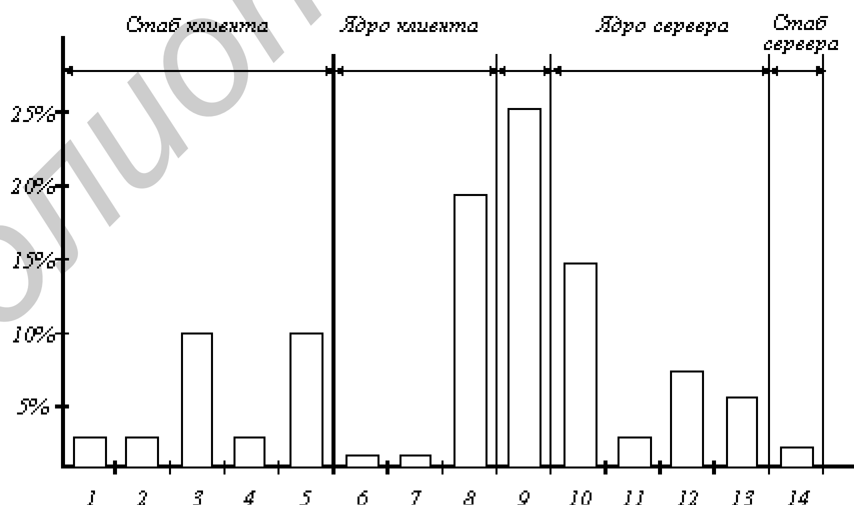


Рис. 3.4. Распределение времени между 14 этапами выполнения RPC

21. Вызов стаба.
22. Подготовка буфера.
23. Упаковка параметров.
24. Заполнение поля заголовка
25. Вычисление контрольной суммы в сообщении.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

26. Прерывание к ядру.
27. Очередь пакета на выполнение.
28. Передача сообщения контроллеру по шине QBUS.
29. Время передачи по сети ethernet.
30. Получение пакета от контроллера.
31. Процедура обработки прерывания.
32. Вычисление контрольной суммы.
33. Переключение контекста в пространство пользователя.
34. Выполнение серверного стаба.

3.2.4. Динамическое связывание

Рассмотрим вопрос о том, как клиент задает месторасположение сервера. Одним из методов решения этой проблемы является непосредственное использование сетевого адреса сервера в клиентской программе. Недостаток такого подхода — его чрезвычайная негибкость: при перемещении сервера, или при увеличении числа серверов, или при изменении интерфейса во всех этих и многих других случаях необходимо перекомпилировать все программы, которые использовали жесткое задание адреса сервера. Для того чтобы избежать всех этих проблем, в некоторых распределенных системах используется так называемое динамическое связывание.

Начальным моментом для динамического связывания является формальное определение (спецификация) сервера. Спецификация содержит имя файл-сервера, номер версии и список процедур-услуг, предоставляемых данным сервером для клиентов. Для каждой процедуры дается описание ее параметров с указанием того, является ли данный параметр входным или выходным относительно сервера. Некоторые параметры могут быть одновременно входными и выходными — например, некоторый массив, который посылается клиентом на сервер, модифицируется там, а затем возвращается обратно клиенту (операция copy/ restore).

Формальная спецификация сервера используется в качестве исходных данных для программы-генератора стабов, которая создает как клиентские, так и серверные стабы. Затем они помещаются в соответствующие библиотеки. Когда пользовательская (клиентская) программа вызывает любую процедуру, определенную в спецификации сервера, соответствующая стаб-процедура связывается с двоичным кодом программы. Аналогично, когда компилируется сервер, с ним связываются серверные стабы.

При запуске сервера самым первым его действием является передача своего серверного интерфейса специальной программе, называемой binder'ом. Этот процесс, известный как процесс регистрации сервера, включает передачу сервером своего имени, номера версии, уникального идентификатора и описателя местонахождения сервера. Описатель системно независим и может представлять собой IP, Ethernet, X.500 или

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

еще какой-либо адрес. Кроме того, он может содержать и другую информацию, например, относящуюся к аутентификации.

Когда клиент вызывает одну из удаленных процедур первый раз, например read, клиентский стаб видит, что он еще не подсоединен к серверу, и посылает сообщение binder-программе с просьбой об импорте интерфейса нужной версии нужного сервера. Если такой сервер существует, то binder передает описатель и уникальный идентификатор клиентскому стабу.

Клиентский стаб при посылке сообщения с запросом использует в качестве адреса описатель. В сообщении содержатся параметры и уникальный идентификатор, который ядро сервера использует для того, чтобы направить поступившее сообщение в нужный сервер в случае, если их несколько на этой машине.

Этот метод, заключающийся в импорте/экспорте интерфейсов, обладает высокой гибкостью. Например, может быть несколько серверов, поддерживающих один и тот же интерфейс, и клиенты распределяются по серверам случайным образом. В рамках этого метода становится возможным периодический опрос серверов, анализ их работоспособности и, в случае отказа, автоматическое отключение, что повышает общую отказоустойчивость системы. Этот метод может также поддерживать аутентификацию клиента. Например, сервер может определить, что он может быть использован только клиентами из определенного списка.

Однако у динамического связывания имеются недостатки, например дополнительные накладные расходы (временные затраты) на экспорт и импорт интерфейсов. Величина этих затрат может быть значительна, так как многие клиентские процессы существуют короткое время, а при каждом старте процесса процедура импорта интерфейса должна быть снова выполнена. Кроме того, в больших распределенных системах может стать узким местом программа binder, а создание нескольких программ аналогичного назначения также увеличивает накладные расходы на создание и синхронизацию процессов.

3.2.5. Семантика удаленного вызова процедур в случае отказов

В идеале RPC должен функционировать правильно и в случае отказов. Рассмотрим следующие классы отказов:

35. Клиент не может определить местонахождение сервера, например, в случае отказа нужного сервера, или из-за того, что программа клиента была скомпилирована давно и использовала старую версию интерфейса сервера. В этом случае в ответ на запрос клиента поступает сообщение, содержащее код ошибки.
36. Потерян запрос от клиента к серверу. Самое простое решение — через определенное время повторить запрос.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

37. Потеряно ответное сообщение от сервера клиенту. Этот вариант сложнее предыдущего, так как некоторые процедуры не являются идемпотентными. Идемпотентной называется процедура, запрос на выполнение которой можно повторить несколько раз и результат при этом не изменится. Примером такой процедуры может служить чтение файла. Но вот процедура снятия некоторой суммы с банковского счета не является идемпотентной, и в случае потери ответа повторный запрос может существенно изменить состояние счета клиента. Одним из возможных решений является приведение всех процедур к идемпотентному виду. Однако на практике это не всегда удается, поэтому может быть использован другой метод — последовательная нумерация всех запросов клиентским ядром. Ядро сервера запоминает номер самого последнего запроса от каждого из клиентов и при получении каждого запроса выполняет анализ — является ли этот запрос первичным или повторным.
38. Сервер потерпел аварию после получения запроса. Здесь также важно свойство идемпотентности, но, к сожалению, не может быть применен подход с нумерацией запросов. В данном случае имеет значение, когда произошел отказ — до или после выполнения операции. Но клиентское ядро не может распознать эти ситуации, для него известно только то, что время ответа истекло. Существует три подхода к этой проблеме:
   - ждать до тех пор, пока сервер не перезагрузится, и пытаться выполнить операцию снова. Этот подход гарантирует, что RPC был выполнен до конца по крайней мере один раз, а возможно, и более;
   - сразу сообщить приложению об ошибке. Этот подход гарантирует, что RPC был выполнен не более одного раза;
   - третий подход не гарантирует ничего. Когда сервер отказывает, клиенту не оказывается никакой поддержки. RPC может быть или не выполнен вообще, или выполнен много раз. Во всяком случае этот способ очень легко реализовать.

Ни один из этих подходов не является очень привлекательным. А идеальный вариант, который бы гарантировал ровно одно выполнение RPC, в общем случае не может быть реализован по принципиальным соображениям. Пусть, например, удаленной операцией является печать некоторого текста, которая включает загрузку буфера принтера и установку одного бита в некотором управляющем регистре принтера, в результате которой принтер стартует. Авария сервера может произойти как за микросекунду до, так и за микросекунду после установки управляющего бита. Момент сбоя целиком определяет процедуру восстановления, но клиент о моменте сбоя узнать не может. Короче говоря, возможность аварии сервера радикально меняет природу RPC и ясно отражает разницу между централизованной и распределенной системой. В первом случае крах сервера ведет к краху клиента и восстановление невозможно. Во втором случае действия по восстановлению системы выполнить и возможно, и необходимо.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Клиент потерпел аварию после отсылки запроса. В этом случае выполняются вычисления результатов, которых никто не ожидает. Такие вычисления называют "сиротами". Наличие сирот может вызвать различные проблемы: непроизводительные затраты процессорного времени, блокирование ресурсов, подмена ответа на текущий запрос ответом на запрос, который был выдан клиентской машиной еще до перезапуска системы.

Как поступать с сиротами? Рассмотрим 4 возможных решения.

- *Уничтожение.* До того как клиентский стаб посылает RPC-сообщение, он делает отметку в журнале, оповещая о том, что он будет сейчас делать. Журнал хранится на диске или в другой памяти, устойчивой к сбоям. После аварии система перезагружается, журнал анализируется и сироты ликвидируются. К недостаткам такого подхода относятся, во-первых, повышенные затраты, связанные с записью о каждом RPC на диск, а во-вторых, возможная неэффективность из-за появления сирот второго поколения, порожденных RPC-вызовами, выданными сиротами первого поколения.
- *Перевоплощение*. В этом случае все проблемы решаются без использования записи на диск. Метод состоит в делении времени на последовательно пронумерованные периоды. Когда клиент перезагружается, он передает широковещательное сообщение всем машинам о начале нового периода. После приема этого сообщения все удаленные вычисления ликвидируются. Конечно, если сеть сегментированная, то некоторые сироты могут и уцелеть.
- *Мягкое перевоплощение* аналогично предыдущему случаю, за исключением того что отыскиваются и уничтожаются не все удаленные вычисления, а только вычисления перезагружающегося клиента.
- *Истечение срока*. Каждому запросу отводится стандартный отрезок времени Т, в течение которого он должен быть выполнен. Если запрос не выполняется за отведенное время, то выделяется дополнительный квант. Хотя это и требует дополнительной работы, но если после аварии клиента сервер ждет в течение интервала Т до перезагрузки клиента, то все сироты обязательно уничтожаются.

На практике ни один из этих подходов не желателен, более того, уничтожение сирот может усугубить ситуацию. Например, пусть сирота заблокировал один или более файлов базы данных. Если сирота будет вдруг уничтожен, то эти блокировки останутся, кроме того, уничтоженные сироты могут остаться стоять в различных системных очередях, в будущем они могут вызвать выполнение новых процессов и т.п.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

3.3. Синхронизация в распределенных системах

К вопросам связи процессов, реализуемой путем передачи сообщений или вызовов RPC, тесно примыкают и вопросы синхронизации процессов. Синхронизация необходима процессам для организации совместного использования ресурсов, таких как файлы или устройства, а также для обмена данными.

В однопроцессорных системах решение задач взаимного исключения, критических областей и других проблем синхронизации осуществлялось с использованием общих методов, таких как семафоры и мониторы. Однако эти методы не совсем подходят для распределенных систем, так как все они базируются на использовании разделяемой оперативной памяти. Например, два процесса, которые взаимодействуют, используя семафор, должны иметь доступ к нему. Если оба процесса выполняются на одной и той же машине, они могут иметь совместный доступ к семафору, хранящемуся, например, в ядре, делая системные вызовы. Однако, если процессы выполняются на разных машинах, то этот метод неприменим, для распределенных систем нужны новые подходы.

3.3.1. Алгоритм синхронизации логических часов

В централизованной однопроцессорной системе, как правило, важно только относительное время и не важна точность часов. В распределенной системе, где каждый процессор имеет собственные часы со своей точностью хода, ситуация резко меняется: программы, использующие время (например, программы, подобные команде make в UNIX, которые используют время создания файлов, или программы, для которых важно время прибытия сообщений и т.п.), становятся зависимыми от того, часами какого компьютера они пользуются. В распределенных системах синхронизация физических часов (показывающих реальное время) является сложной проблемой, но, с другой стороны, очень часто в этом нет никакой необходимости. То есть процессам не нужно, чтобы во всех машинах было правильное время, для них важно, чтобы оно было везде одинаковое, более того, для некоторых процессов важен только правильный порядок событий. В этом случае мы имеем дело с логическими часами.

Введем для двух произвольных событий отношение "случилось до". Выражение a ® b читается "a случилось до b" и означает, что все процессы в системе считают, что сначала произошло событие a, а потом — событие b. Отношение "случилось до" обладает свойством транзитивности: если выражения a ® b и b ® c истинны, то справедливо и выражение a ® c. Для двух событий одного и того же процесса всегда можно установить отношение "случилось до", аналогично может быть установлено это отношение и для событий передачи сообщения одним процессом и приема его другим, так как прием не может произойти раньше отправки. Однако, если два произвольных события случились в разных процессах на разных машинах, и эти процессы не имеют между собой никакой связи (даже

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

косвенной через третьи процессы), то нельзя сказать с полной определенностью, какое из событий произошло раньше, а какое позже.

Ставится задача создания такого механизма ведения времени, который бы для каждого события a мог указать значение времени T(a), с которым бы были согласны все процессы в системе. При этом должно выполняться условие: если a ® b , то T(a) < T(b). Кроме того, время может только увеличиваться и, следовательно, любые корректировки времени могут выполняться только путем добавления положительных значений и никогда — путем вычитания.

Рассмотрим алгоритм решения этой задачи, который предложил Lamport. Для отметок времени в нем используются события. На рис. 3.5 показаны три процесса, выполняющихся на разных машинах, каждая из которых имеет свои часы, идущие со своей скоростью. Как видно из рисунка, когда часы процесса 0 показали время 6, в процессе 1 часы показывали 8, а в процессе 2 — 10. Предполагается, что все эти часы идут с постоянной для себя скоростью.

В момент времени 6 процесс 0 посылает сообщение A процессу 1. Это сообщение приходит к процессу 1 в момент времени 16 по его часам. В логическом смысле это вполне возможно, так как 6<16. Аналогично, сообщение B, посланное процессом 1 процессу 2 пришло к последнему в момент времени 40, то есть его передача заняла 16 единиц времени, что также является правдоподобным.

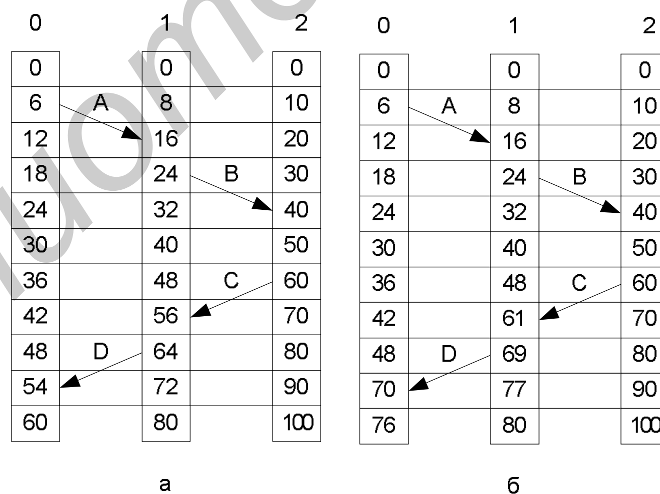


Рис. 3.5. Синхронизация логических часов:
а — три процесса, каждый со своими собственными часами;
б — алгоритм синхронизации логических часов

Ну а далее начинаются весьма странные вещи. Сообщение C от процесса 2 к процессу 1 было отправлено в момент времени 64, а поступило в место назначения в момент времени 54. Очевидно, что это невозможно. Такие ситуации необходимо предотвращать. Решение Lamport'a вытекает непосредственно из отношений "случилось до". Так как C было отправлено

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

в момент 60, то оно должно дойти в момент 61 или позже. Следовательно, каждое сообщение должно нести с собой время своего отправления по часам машины-отправителя. Если в машине, получившей сообщение, часы показывают время, которое меньше времени отправления, то эти часы переводятся вперед, так, чтобы они показали время, большее времени отправления сообщения. На рис. 3.5, б видно, что С поступило в момент 61, а сообщение D — в 70.

Этот алгоритм удовлетворяет сформулированным выше требованиям.

3.3.2. Алгоритмы взаимного исключения

Системы, состоящие из нескольких процессов, часто легче программировать, используя так называемые критические секции. Когда процессу нужно читать или модифицировать некоторые разделяемые структуры данных, он прежде всего входит в критическую секцию для того, чтобы обеспечить себе исключительное право использования этих данных, при этом он уверен, что никакой процесс не будет иметь доступа к этому ресурсу одновременно с ним. Это называется взаимным исключением. В однопроцессорных системах критические секции защищаются семафорами, мониторами и другими аналогичными конструкциями. Рассмотрим, какие алгоритмы могут быть использованы в распределенных системах.

3.3.3. Централизованный алгоритм

Наиболее очевидный и простой путь реализации взаимного исключения в распределенных системах — это применение тех же методов, которые используются в однопроцессорных системах. Один из процессов выбирается в качестве координатора (например, процесс, выполняющийся на машине, имеющей наибольшее значение сетевого адреса). Когда какой-либо процесс хочет войти в критическую секцию, он посылает сообщение с запросом к координатору, оповещая его о том, в какую критическую секцию он хочет войти, и ждет от координатора разрешение. Если в этот момент ни один из процессов не находится в критической секции, то координатор посылает ответ с разрешением. Если же некоторый процесс уже выполняет критическую секцию, связанную с данным ресурсом, то никакой ответ не посылается; запрашивавший процесс ставится в очередь, и после освобождения критической секции ему отправляется ответ-разрешение. Этот алгоритм гарантирует взаимное исключение, но вследствие своей централизованной природы обладает низкой отказоустойчивостью.

3.3.4. Распределенный алгоритм

Когда процесс хочет войти в критическую секцию, он формирует сообщение, содержащее имя нужной ему критической секции, номер процесса и текущее значение времени. Затем он посылает это сообщение

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

всем другим процессам. Предполагается, что передача сообщения надежна, то есть получение каждого сообщения сопровождается подтверждением. Когда процесс получает сообщение такого рода, его действия зависят от того, в каком состоянии по отношению к указанной в сообщении критической секции он находится. Имеют место три ситуации:

39. Если получатель не находится и не собирается входить в критическую секцию в данный момент, то он отсылает назад процессу-отправителю сообщение с разрешением.
40. Если получатель уже находится в критической секции, то он не отправляет никакого ответа, а ставит запрос в очередь.
41. Если получатель хочет войти в критическую секцию, но еще не сделал этого, то он сравнивает временную отметку поступившего сообщения со значением времени, которое содержится в его собственном сообщении, разосланном всем другим процессам. Если время в поступившем к нему сообщении меньше, то есть его собственный запрос возник позже, то он посылает сообщение-разрешение, в обратном случае он не посылает ничего и ставит поступившее сообщение-запрос в очередь.

Процесс может войти в критическую секцию только в том случае, если он получил ответные сообщения-разрешения от всех остальных процессов. Когда процесс покидает критическую секцию, он посылает разрешение всем процессам из своей очереди и исключает их из очереди.

3.3.5. Алгоритм Token Ring

Совершенно другой подход к достижению взаимного исключения в распределенных системах иллюстрируется рис. 3.6. Все процессы системы образуют логическое кольцо, то есть каждый процесс знает номер своей позиции в кольце, а также номер ближайшего к нему следующего процесса. Когда кольцо инициализируется, процессу 0 передается так называемый токен. Токен циркулирует по кольцу. Он переходит от процесса n к процессу n+1 путем передачи сообщения по типу "точка-точка". Когда процесс получает токен от своего соседа, он анализирует, не требуется ли ему самому войти в критическую секцию. Если да, то процесс входит в критическую секцию. После того как процесс выйдет из критической секции, он передает токен дальше по кольцу. Если же процесс, принявший токен от своего соседа, не заинтересован во вхождении в критическую секцию, то он сразу отправляет токен в кольцо. Следовательно, если ни один из процессов не желает входить в критическую секцию, то в этом случае токен просто циркулирует по кольцу с высокой скоростью.

Сравним эти три алгоритма взаимного исключения. Централизованный алгоритм является наиболее простым и наиболее эффективным. При его использовании требуется только три сообщения для того, чтобы процесс

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

вошел и покинул критическую секцию: запрос и сообщение-разрешение для входа и сообщение об освобождении ресурса при выходе. При использовании распределенного алгоритма для одного использования критической секции требуется послать (n-1) сообщений-запросов (где n — число процессов) — по одному на каждый процесс и получить (n-1) сообщений-разрешений, то есть всего необходимо 2(n-1) сообщений. В алгоритме Token Ring число сообщений переменно: от 1 в случае, если каждый процесс входил в критическую секцию, до бесконечно большого числа при циркуляции токена по кольцу, в котором ни один процесс не входил в критическую секцию.

К сожалению, все эти три алгоритма плохо защищены от отказов. В первом случае к краху приводит отказ координатора, во втором — отказ любого процесса (парадоксально, но распределенный алгоритм оказывается менее отказоустойчивым, чем централизованный), а в третьем — потеря токена или отказ процесса.

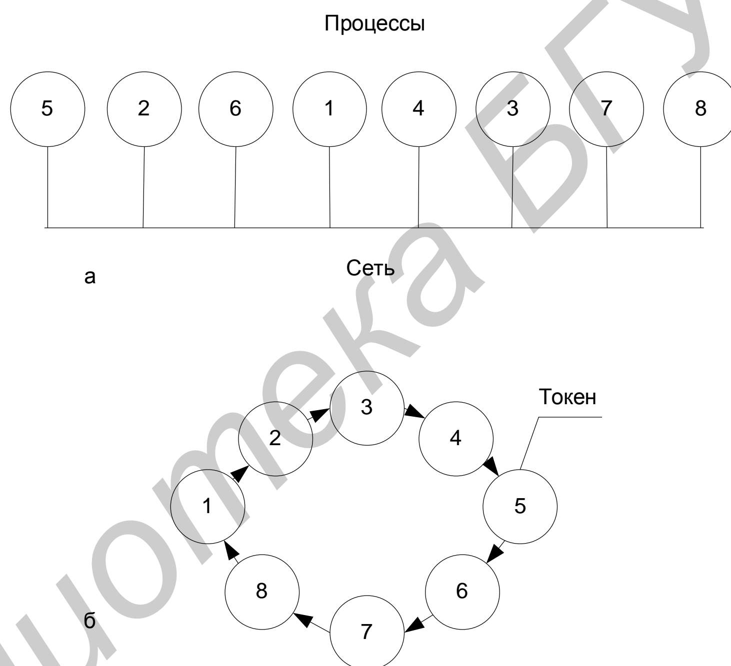


Рис. 3.6. Средства взаимного исключения в распределенных системах:
а — неупорядоченная группа процессов в сети;
б — логическое кольцо, образованное программным обеспечением

3.3.6. Неделимые транзакции

Все средства синхронизации, которые были рассмотрены ранее, относятся к нижнему уровню, например, семафоры. Они требуют от программиста детального знания алгоритмов взаимного исключения, управления критическими секциями, умения предотвращать клинчи (взаимные блокировки), а также владения средствами восстановления после краха. Однако существуют средства синхронизации более высокого уровня, которые освобождают программиста от необходимости вникать во все эти подробности и позволяют ему сконцентрировать свое внимание на логике

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

алгоритмов и организации параллельных вычислений. Таким средством является неделимая транзакция.

Модель неделимой транзакции пришла из бизнеса. Представьте себе переговорный процесс двух фирм о продаже-покупке некоторого товара. В процессе переговоров условия договора могут многократно меняться, уточняться. Пока договор еще не подписан обеими сторонами, каждая из них может от него отказаться. Но после подписания контракта сделка (transaction) должна быть выполнена.

Компьютерная транзакция полностью аналогична. Один процесс объявляет, что он хочет начать транзакцию с одним или более процессами. Они могут некоторое время создавать и уничтожать разные объекты, выполнять какие-либо операции. Затем инициатор объявляет, что он хочет завершить транзакцию. Если все с ним соглашаются, то результат фиксируется. Если один или более процессов отказываются (или они потерпели крах еще до выработки согласия), тогда измененные объекты возвращаются точно к тому состоянию, в котором они находились до начала выполнения транзакции. Такое свойство "все-или-ничего" облегчает работу программиста.

Для программирования с использованием транзакций требуется некоторый набор примитивов, которые должны быть предоставлены программисту либо операционной системой, либо языком программирования.

Первые два примитива используются для определения границ транзакции. Операции между ними представляют собой тело транзакции. Либо все они должны быть выполнены, либо ни одна из них. Это может быть системный вызов, библиотечная процедура или группа операторов языка программирования, заключенная в скобки.

Транзакции обладают следующими свойствами: упорядочиваемостью, неделимостью, постоянством.

- *Упорядочиваемость* гарантирует, что если две или более транзакции выполняются в одно и то же время, то конечный результат выглядит так, как если бы все транзакции выполнялись последовательно в некотором (в зависимости от системы) порядке.
- *Неделимость* означает, что когда транзакция находится в процессе выполнения, то никакой другой процесс не видит ее промежуточных результатов.
- *Постоянство* означает, что после фиксации транзакции никакой сбой не может отменить результатов ее выполнения.

Если программное обеспечение гарантирует вышеперечисленные свойства, то это означает, что в системе поддерживается механизм транзакций.

Рассмотрим некоторые подходы к реализации механизма транзакций.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

В соответствии с первым подходом, когда процесс начинает транзакцию, то он работает в индивидуальном рабочем пространстве, содержащем все файлы и другие объекты, к которым он имеет доступ. Пока транзакция не зафиксируется или не прервется, все изменения данных происходят в этом рабочем пространстве, а не в "реальном", под которым мы понимаем обычную файловую систему. Главная проблема этого подхода состоит в больших накладных расходах по копированию большого объема данных в индивидуальное рабочее пространство, хотя и имеются несколько приемов уменьшения этих расходов.

Второй общий подход к реализации механизма транзакций называется списком намерений. Этот метод заключается в том, что модифицируются сами файлы, а не их копии, но перед изменением любого блока производится запись в специальный файл — журнал регистрации, где отмечается, какая транзакция делает изменения, какие файл и блок изменяются и каковы старое и новое значения изменяемого блока. Только после успешной записи в журнал регистрации делаются изменения в исходном файле. Если транзакция фиксируется, то и об этом делается запись в журнал регистрации, но старые значения измененных данных сохраняются. Если транзакция прерывается, то информация журнала регистрации используется для приведения файла в исходное состояние, и это действие называется откатом.

В распределенных системах фиксация транзакций может потребовать взаимодействия нескольких процессов на разных машинах, каждая из которых хранит некоторые переменные, файлы, базы данных. Для достижения свойства неделимости транзакций в распределенных системах используется специальный протокол, называемый протоколом двухфазной фиксации транзакций. Хотя он и не является единственным протоколом такого рода, но наиболее широко используется.

Суть этого протокола состоит в следующем. Один из процессов выполняет функции координатора (рис. 3.7). Координатор начинает транзакцию, делая запись об этом в своем журнале регистрации, затем он посылает всем подчиненным процессам, также выполняющим эту транзакцию, сообщение "подготовиться к фиксации". Когда подчиненные процессы получают это сообщение, то они проверяют, готовы ли они к фиксации, делают запись в своем журнале и посылают координатору сообщение-ответ "готов к фиксации". После этого подчиненные процессы остаются в состоянии готовности и ждут от координатора команду фиксации. Если хотя бы один из подчиненных процессов не откликнулся, то координатор откатывает подчиненные транзакции, включая и те, которые подготовились к фиксации.

Выполнение второй фазы заключается в том, что координатор посылает команду "фиксировать" (commit) всем подчиненным процессам. Выполняя эту команду, последние фиксируют изменения и завершают подчиненные транзакции. В результате гарантируется одновременное синхронное завершение (удачное или неудачное) распределенной транзакции.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Рис. 3.7. Двухфазный протокол фиксации транзакции

3.4. Процессы и нити в распределенных системах

3.4.1. Понятие "нить"

В традиционных ОС понятие нити тождественно понятию процесса. В действительности желательно иметь несколько нитей управления, разделяющих единое адресное пространство, но выполняющихся квазипараллельно.

Предположим, например, что файл-сервер блокируется, ожидания выполнения операции с диском. Если сервер имеет несколько нитей управления, вторая нить может выполняться, пока первая нить находится в состоянии ожидания. Это повышает пропускную способность и производительность. Эта цель не достигается путем создания двух независимых серверных процессов, потому что они должны разделять общий буфер кэша, который требуется им, чтобы быть в одном адресном пространстве.

На рис. 3.8, а показана машина с тремя процессами. Каждый процесс имеет собственный программный счетчик, собственный стек, собственный набор регистров и собственное адресное пространство. Каждый процесс не должен ничего делать с остальными, за исключением того что они могут взаимодействовать посредством системных примитивов связи, таких, как семафоры, мониторы, сообщения. На рис. 3.8, б показана другая машина с одним процессом. Этот процесс состоит из нескольких нитей управления, обычно называемых просто нитями или иногда облегченными процессами. Во многих отношениях нити подобны мини-процессам. Каждая нить выполняется строго последовательно и имеет свой собственный программный счетчик и стек. Нити разделяют процессор так, как это

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

делают процессы (разделение времени). Только на многопроцессорной системе они действительно выполняются параллельно. Нити могут, например, порождать нити-потомки, могут переходить в состояние ожидания до завершения системного вызова, как обычные процессы; пока одна нить заблокирована, другая нить того же процесса может выполняться.

Нити делают возможным сохранение идеи последовательных процессов, которые выполняют блокирующие системные вызовы (например RPC для обращения к диску) и в то же время позволяют достичь параллелизма вычислений. Блокирующие системные вызовы делают проще программирование, а параллелизм повышает производительность.

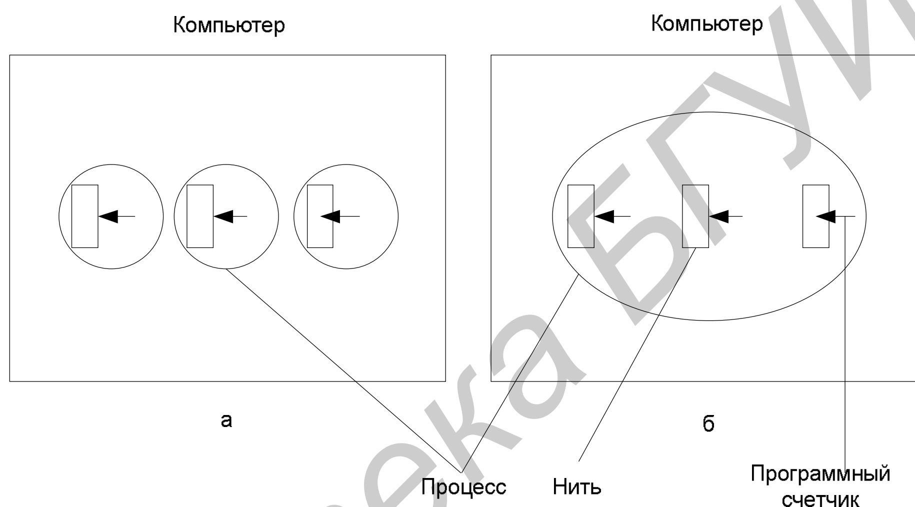


Рис. 3.8. Три процесса с одной нитью каждый (а); один процесс с тремя нитями (б)

3.4.2. Различные способы организации вычислительного процесса с использованием нитей

Один из возможных способов организации вычислительного процесса показан на рис. 3.9, а. Здесь нить-диспетчер читает приходящие запросы на работу из почтового ящика системы. После проверки запроса диспетчер выбирает простаивающую (то есть блокированную) рабочую нить, передает ей запрос и активизирует ее, устанавливая, например, семафор, который она ожидает.

Когда рабочая нить активизируется, она проверяет, может ли быть выполнен запрос с данными разделяемого блока кэша, к которому имеют отношение все нити. Если нет, она посылает сообщение к диску, чтобы получить нужный блок (предположим, это READ), и переходит в состояние блокировки, ожидая завершения дисковой операции. В этот момент происходит обращение к планировщику, в результате работы которого активизируется другая нить, возможно, нить-диспетчер или некоторая рабочая нить, готовая к выполнению.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Структура с диспетчером – не единственный путь организации многонитевой обработки. В модели "команда" все нити эквивалентны, каждая получает и обрабатывает свои собственные запросы. Иногда работы приходят, а нужная нить занята, особенно если каждая нить специализируется на выполнении особого вида работ. В этом случае может создаваться очередь незавершенных работ. При такой организации нити должны вначале просматривать очередь работ, а затем почтовый ящик.

Нити могут быть также организованы в виде конвейера. В этом случае первая нить порождает некоторые данные и передает их для обработки следующей нити и т.д. Хотя эта организация и не подходит для файл-сервера, для других задач, например задач типа "производитель-потребитель", это хорошее решение.

Нити часто полезны и для клиентов. Например, если клиент хочет растиражировать файл на много серверов, он может создать по одной нити для копирования на каждом сервере. Другое использование нитей клиентами — это управление сигналами, такими как прерывание с клавиатуры (del или break). Вместо обработки сигнала прерывания одна нить назначается для постоянного ожидания поступления сигналов. Таким образом, использование нитей может сократить необходимое количество прерываний пользовательского уровня.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

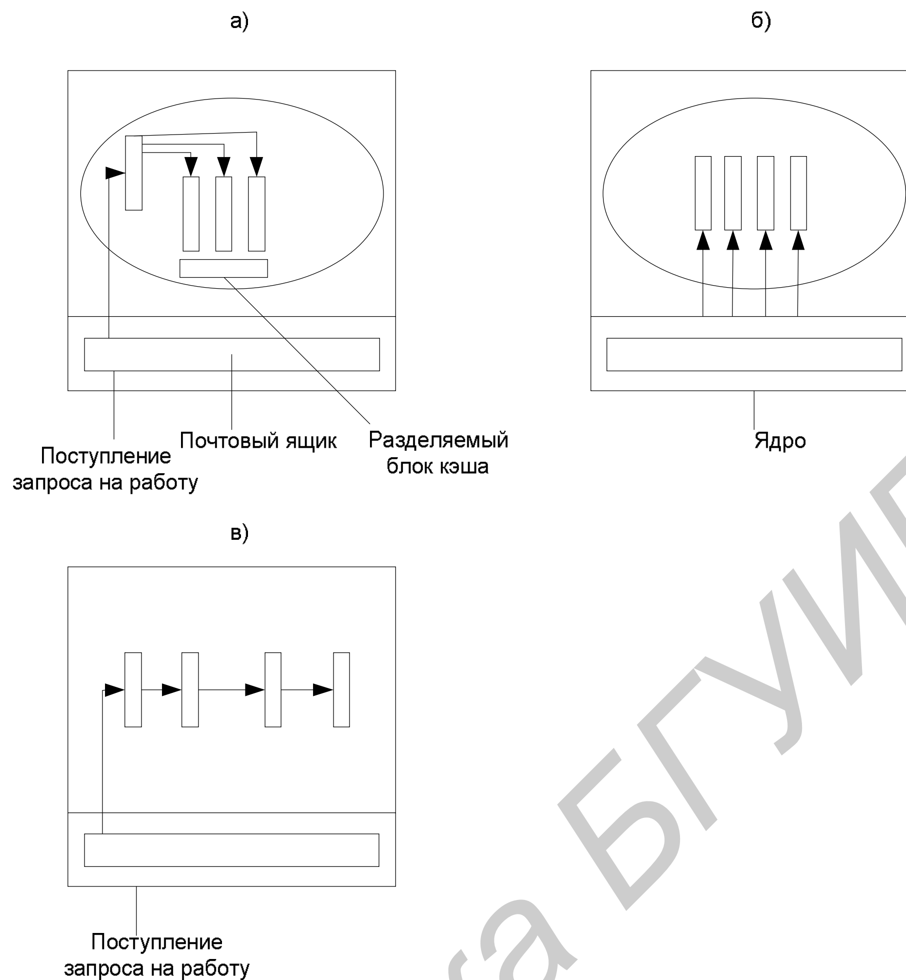


Рис. 3.9. Три способа организации нитей в процессе:
а — модель “диспетчер/рабочие нити”; б — модель “команда”; в — модель конвейера

Другой аргумент в пользу нитей не имеет отношения ни к удаленным вызовам, ни к коммуникациям. Некоторые прикладные задачи легче программировать, используя параллелизм, например задачи типа "производитель-потребитель". Параллельное выполнение не столько важно, сколько ясность программы. А поскольку они разделяют общий буфер, не стоит их делать отдельными процессами.

Наконец, в многопроцессорных системах нити из одного адресного пространства могут выполняться параллельно на разных процессорах. С другой стороны, правильно сконструированные программы, которые используют нити, должны работать одинаково хорошо на однопроцессорной машине в режиме разделения времени между нитями и на настоящем мультипроцессоре.

3.4.3. Вопросы реализации нитей

Существует два подхода к управлению нитями: статический и динамический. При статическом подходе вопрос, сколько будет нитей, решается уже на стадии написания программы или на стадии компиляции. Каждой нити назначается фиксированный стек. Этот подход простой, но

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

негибкий. Более общим является динамический подход, который позволяет создавать и удалять нити оперативно по ходу выполнения. Системный вызов для создания нити обычно содержится в нити главной программы в виде указателя на процедуру с указанием размера стека, а также других параметров, например диспетчерского приоритета. Вызов обычно возвращает идентификатор нити, который можно использовать в последующих вызовах, связанных с этой нитью. В этой модели процесс начинается с одной нити, но может создавать их еще, когда необходимо.

Завершаться нити могут одним из двух способов: по своей инициативе, когда завершается работа, и извне. Во многих случаях, например при конвейерной модели, нити создаются сразу же после старта процесса и никогда не уничтожаются.

Поскольку нити разделяют общую память, они могут (и, как правило, делают это) использовать ее для сохранения данных, которые совместно используются множеством нитей, таких, например, как буфер в системе "производитель-потребитель". Доступ к разделяемым данным обычно программируется с использованием критических секций, предотвращающих попытки сразу нескольких нитей обратиться к одним и тем же данным в одно и то же время. Критическая секция наиболее легко реализуется с использованием семафоров, мониторов и аналогичных конструкций.

Нити могут быть реализованы как в пользовательском пространстве, так и в пространстве ядра. В первом случае нити работают на базе прикладной системы, управляющей всеми операциями с нитями. Первым преимуществом такого способа является то, что можно реализовать нити в операционной системе, которая их не поддерживает. Операционная система прикладная среда, управляющая нитями, кажется одним процессом. Все вызовы (приостановить, проверить семафор и т. д.) обрабатываются как вызовы функций этой прикладной среды. Она сохраняет регистры и переключает указатели счетчика команд и стека. В этом случае переключение происходит быстрее, чем с помощью ядра. Такая реализация имеет еще одно преимущество: для каждого процесса можно организовать свою схему планирования. Однако этот подход связан с некоторыми проблемами, одна из которых состоит в следующем. При выполнении блокирующих системных вызовов приостанавливается весь набор нитей, принадлежащих этому процессу. Чтобы избежать этого, можно сделать все системные вызовы неблокирующими, но это требует изменений в ОС, что нежелательно, так как одной из целей реализации нитей в пользовательском пространстве является их работа в существующих операционных системах.

Такой проблемы не существует при реализации нитей в пространстве ядра. Преимущество заключается также и в том, что ядро может при диспетчеризации выбирать нить из другого процесса. Однако хотя

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

механизм управления нитями аналогичен первому случаю, временные затраты на переключение нитей выше, так как тратится время на переключение из режима пользователя в режим ядра.

3.4.4. Нити и удаленный вызов процедур

Обычно в распределенных системах используются как RPC, так и нити. Так как нити были введены как дешевая альтернатива стандартным процессам, то естественно, что исследователи обратили особое внимание в этом контексте на RPC: нельзя ли их также сделать облегченными. Было замечено, что в распределенных системах значительное количество RPC обрабатывается на той же машине, на которой они были вызваны (локально), например, вызовы к менеджеру окон. Поэтому была предложена новая схема, которая делает возможным для нити одного процесса вызвать нить другого процесса на этой же машине более эффективно, чем обычным способом.

Идея заключается в следующем. Когда стартует серверная нить S, то она экспортирует свой интерфейс, сообщая о нем ядру. Интерфейс определяет, какие процедуры могут быть вызваны, каковы их параметры и т.п. Когда стартует клиентская нить C, то она импортирует интерфейс из ядра в том случае, если собирается вызвать S, и ей дается специальный идентификатор для выполнения определенного вызова. Ядро теперь знает, что C собирается позже вызвать S и создает специальные структуры данных для подготовки к вызову.

Одна из этих структур данных является стеком аргументов, который разделяется нитями C и S и отображается в оба адресных пространства для чтения и записи. Для вызова сервера нить C помещает аргументы в разделяемый стек, используя обычную процедуру передачи параметров, а затем прерывает ядро, помещая данный ей идентификатор в регистр. По этому идентификатору ядро видит, что вызов является локальным. (Если бы он был удаленным, то ядро обработало бы его обычным способом для удаленных вызовов.) Затем ядро выполняет переключение из адресного пространства клиента в адресное пространство нити-сервера и запускает в рамках клиентской нити требуемую процедуру сервера. При таком способе вызова аргументы уже загружены в нужное место, так что копирование или перегруппировка аргументов не требуется. Главный результат — локальный вызов RPC — будет выполнен этим способом гораздо быстрее.

Другой прием широко используется для ускорения удаленных RPC. Идея основана на следующем наблюдении: когда нить-сервер блокируется, ожидая нового запроса, ее контекст почти всегда не содержит важной информации. Следовательно, когда нить завершает обработку запроса, то ее просто удаляют. При поступлении на сервер нового сообщения ядро создает новую нить для обслуживания этого запроса. Кроме того, ядро помещает сообщение в адресное пространство сервера и устанавливает новый стек нити для доступа к сообщению. Эту схему иногда называют неявным вызовом.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Этот метод имеет несколько преимуществ по сравнению с обычным RPC. Во-первых, нити не должны блокироваться, ожидая новую работу, следовательно, контекст не нужно сохранять, во-вторых, создание новой нити проще, чем активизация существующей приостановленной, так как не нужно восстанавливать контекст.

3.5. Распределенные файловые системы

Ключевым компонентом любой распределенной системы является файловая система. Как и в централизованных системах, в распределенной системе функцией файловой системы является хранение программ и данных и предоставление доступа к ним по мере необходимости. Файловая система поддерживается одной или более машинами, называемыми файл-серверами. Файл-серверы перехватывают запросы на чтение или запись файлов, поступающие от других машин (не серверов). Эти другие машины называются клиентами. Каждый посланный запрос проверяется и выполняется, а ответ отсылается обратно. Файл-серверы обычно содержат иерархические файловые системы, каждая из которых имеет корневой каталог и каталоги более низких уровней. Рабочая станция может подсоединять и монтировать эти файловые системы к своим локальным файловым системам. При этом монтируемые файловые системы остаются на серверах.

Важно понимать различие между файловым сервисом и файловым сервером. Файловый сервис — это описание функций, которые файловая система предлагает своим пользователям. Это описание включает имеющиеся примитивы, их параметры и функции, которые они выполняют. С точки зрения пользователей файловый сервис определяет то, с чем пользователи могут работать, но ничего не говорит о том, как все это реализовано. В сущности, файловый сервис определяет интерфейс файловой системы с клиентами.

Файловый сервер — это процесс, который выполняется на отдельной машине и помогает реализовывать файловый сервис. В системе может быть один файловый сервер или несколько, но в хорошо организованной распределенной системе пользователи не знают, как реализована файловая система. В частности, они не знают количество файловых серверов, их месторасположение и функции. Они только знают, что если процедура определена в файловом сервисе, то требуемая работа каким-то образом выполняется и им возвращаются требуемые результаты. Более того, пользователи даже не должны знать, что файловый сервис является распределенным. В идеале он должен выглядеть так же, как и в централизованной файловой системе.

Так как обычно файловый сервер — это просто пользовательский процесс (или иногда процесс ядра), выполняющийся на некоторой машине, в системе может быть несколько файловых серверов, каждый из которых

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

предлагает различный файловый сервис. Например, в распределенной системе может быть два сервера, которые обеспечивают файловые сервисы систем UNIX и MS-DOS соответственно, и любой пользовательский процесс пользуется подходящим сервисом.

Файловый сервис в распределенных файловых системах (впрочем как и в централизованных) имеет две функционально различные части: собственно файловый сервис и сервис каталогов. Первый имеет дело с операциями над отдельными файлами, такими, как чтение, запись или добавление, а второй — с созданием каталогов и управлением ими, добавлением и удалением файлов из каталогов и т.п.

3.5.1. Интерфейс файлового сервиса

Для любого файлового сервиса, независимо от того, централизован он или распределен, самым главным является вопрос, что такое файл. Во многих системах, таких как UNIX и MS DOS, файл — это неинтерпретируемая последовательность байтов. Значение и структура информации в файле является заботой прикладных программ, операционную систему это не интересует.

В ОС мейнфреймов поддерживаются разные типы логической организации файлов, каждый с различными свойствами. Файл может быть организован как последовательность записей, и у операционной системы имеются вызовы, которые позволяют работать на уровне этих записей. Большинство современных распределенных файловых систем поддерживают определение файла как последовательности байтов, а не последовательности записей. Файл характеризуется атрибутами: именем, размером, датой создания, идентификатором владельца, адресом и др.

Важным аспектом файловой модели является возможность модификации файла после его создания. Обычно файлы могут модифицироваться, но в некоторых распределенных системах единственными операциями с файлами являются *создать* и *прочитать*. Такие файлы называются неизменяемыми. Для неизменяемых файлов намного легче осуществить кэширование файла и его репликацию (тиражирование), так как исключается все проблемы, связанные с обновлением всех копий файла при его изменении.

Файловый сервис может быть разделен на два типа в зависимости от того, поддерживает ли он модель загрузки-выгрузки или модель удаленного доступа. В модели загрузки-выгрузки пользователю предлагаются средства чтения или записи файла целиком. Эта модель предполагает следующую схему обработки файла: чтение файла с сервера на машину клиента, обработка файла на машине клиента и запись обновленного файла на сервер. Преимуществом этой модели является ее концептуальная простота. Кроме того, передача файла целиком очень эффективна. Главным недостатком этой модели являются высокие требования к дискам клиентов.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Кроме того, неэффективно перемещать весь файл, если нужна его маленькая часть.

Другой тип файлового сервиса соответствует модели удаленного доступа, которая предполагает поддержку большого количества операций над файлами: открытие и закрытие файлов, чтение и запись частей файла, позиционирование в файле, проверка и изменение атрибутов файла и т.д. В то время как в модели загрузки-выгрузки файловый сервер обеспечивал только хранение и перемещение файлов, в данном случае вся файловая система выполняется на серверах, а не на клиентских машинах. Преимуществом такого подхода являются низкие требования к дисковому пространству на клиентских машинах, а также исключение необходимости передачи целого файла, когда нужна только его часть.

3.5.2. Интерфейс сервиса каталогов

Природа сервиса каталогов не зависит от типа используемой модели файлового сервиса. В распределенных системах используются те же принципы организации каталогов, что и в централизованных, в том числе многоуровневая организация каталогов.

Принципиальной проблемой, связанной со способами именования файлов, является обеспечение прозрачности. В данном контексте прозрачность понимается в двух слабо различимых смыслах. Первый — прозрачность расположения — означает, что имена не дают возможности определить месторасположение файла. Например, имя */server1/dir1/ dir2/x* говорит, что файл x расположен на сервере 1, но не указывает, где расположен этот сервер. Сервер может перемещаться по сети, а полное имя файла при этом не меняется. Следовательно, такая система обладает прозрачностью расположения.

Предположим, что файл x очень большой, а на сервере 1 мало места, предположим далее, что на сервере 2 места много. Система может захотеть переместить автоматически файл x на сервер 2. К сожалению, когда первый компонент всех имен — это имя сервера, система не может переместить файл на другой сервер автоматически, даже если каталоги *dir1* и *dir2* находятся на обоих серверах. Программы, имеющие встроенные строки имен файлов, не будут правильно работать в этом случае. Система, в которой файлы могут перемещаться без изменения имен, обладает свойством независимости от расположения. Распределенная система, которая включает имена серверов или машин непосредственно в имена файлов, не является независимой от расположения. Система, базирующаяся на удаленном монтировании, также не обладает этим свойством, так как в ней невозможно переместить файл из одной группы файлов в другую и продолжать после этого пользоваться старыми именами. Независимости от расположения трудно достичь, но это желаемое свойство распределенной системы.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Большинство распределенных систем используют какую-либо форму двухуровневого именования: на одном уровне файлы имеют символические имена, такие как prog.c, предназначенные для использования людьми, а на другом — внутренние, двоичные имена, для использования самой системой. Каталоги обеспечивают отображение между двумя этими уровнями имен. Отличием распределенных систем от централизованных является возможность соответствия одному символьному имени нескольких двоичных имен. Обычно это используется для представления оригинального файла и его архивных копий. Имея несколько двоичных имен, можно при недоступности одной из копий файла получить доступ к другой. Этот метод обеспечивает отказоустойчивость за счет избыточности.

3.5.3. Семантика разделения файлов

Когда два или более пользователей разделяют один файл, необходимо точно определить семантику чтения и записи, чтобы избежать проблем. В централизованных системах, разрешающих разделение файлов, таких как UNIX, обычно определяется, что когда операция *чтение* следует за операцией *запись*, то читается только что обновленный файл. Аналогично, когда операция чтения следует за двумя операциями записи, то читается файл, измененный последней операцией записи. Тем самым система придерживается абсолютного временного упорядочивания всех операций и всегда возвращает самое последнее значение. Будем называть эту модель семантикой UNIX. В централизованной системе (и даже на мультипроцессоре с разделяемой памятью) ее легко и понять, и реализовать.

Семантика UNIX может быть обеспечена и в распределенных системах, но только если в ней имеется лишь один файловый сервер и клиенты не кэшируют файлы. Для этого все операции чтения и записи направляются на файловый сервер, который обрабатывает их строго последовательно. На практике, однако, производительность распределенной системы, в которой все запросы к файлам идут на один сервер, часто становится неудовлетворительной. Эта проблема иногда решается путем разрешения клиентам обрабатывать локальные копии часто используемых файлов в своих личных кэшах. Если клиент сделает локальную копию файла в своем локальном кэше и начнет ее модифицировать, а вскоре после этого другой клиент прочитает этот файл с сервера, то он получит неверную копию файла. Одним из способов устранения этого недостатка является немедленный возврат всех изменений в кэшированном файле на сервер. Такой подход хотя и концептуально прост, но не эффективен.

Другим решением является введение так называемой сессионной семантики, в соответствии с которой изменения в открытом файле сначала видны только процессу, который модифицирует файл, и только после закрытия файла эти изменения могут видеть другие процессы. При использовании сессионной семантики возникает проблема одновременного использования одного и того же файла двумя или более клиентами. Одним

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

из решений этой проблемы является принятие правила, в соответствии с которым окончательным является тот вариант, который был закрыт последним. Менее эффективным, но гораздо более простым в реализации является вариант, при котором окончательным результирующим файлом на сервере может оказаться любой из этих файлов.

Следующий подход к разделению файлов заключается в том, чтобы сделать все файлы неизменяемыми. Тогда файл нельзя открыть для записи, а можно выполнять только операции *создать* и *читать*. Тогда для изменения файла остается только возможность создать полностью новый файл и поместить его в каталог под именем старого файла. Следовательно, хотя файл и нельзя модифицировать, его можно заменить (автоматически) новым файлом. Другими словами, хотя файлы и нельзя обновлять, но каталоги обновлять можно. Таким образом, проблема, связанная с одновременным использованием файла, просто исчезнет.

Четвертый способ работы с разделяемыми файлами в распределенных системах — это использование механизма неделимых транзакций.

Итак, было рассмотрено четыре различных подхода к работе с разделяемыми файлами в распределенных системах.

- **Семантика UNIX**. Каждая операция над файлом немедленно становится видимой для всех процессов.
- **Сессионная семантика**. Изменения не видны до тех пор, пока файл не закрывается.
- **Неизменяемые файлы**. Модификации невозможны, разделение файлов и репликация упрощаются.
- **Транзакции**. Все изменения делаются по принципу "все или ничего".

3.5.4. Вопросы разработки структуры файловой системы

Рассмотрим прежде всего вопрос о распределении серверной и клиентской частей между машинами. В некоторых системах (например NFS) нет разницы между клиентом и сервером, на всех машинах работает одно и то же базовое программное обеспечение, так что любая машина, которая хочет предложить файловый сервис, свободно может это сделать. Для этого ей достаточно экспортировать имена выбранных каталогов, чтобы другие машины могли иметь к ним доступ.

В других системах файловый сервер — это только пользовательская программа, так что система может быть сконфигурирована как клиент, как сервер или как клиент и сервер одновременно. Третьим, крайним случаем является система, в которой клиенты и серверы — это принципиально различные машины, как в терминах аппаратуры, так и в терминах программного обеспечения. Серверы могут даже работать под управлением другой операционной системы.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Вторым важным вопросом реализации файловой системы является структуризация сервиса файлов и каталогов. Один подход заключается в комбинировании этих двух сервисов на одном сервере. При другом подходе эти сервисы разделяются. В последнем случае при открытии файла требуется обращение к серверу каталогов, который отображает символьное имя в двоичное, а затем обращение к файловому серверу с двоичным именем для действительного чтения или записи файла.

Аргументом в пользу разделения сервисов является тот факт, что они на самом деле слабо связаны, поэтому их раздельная реализация более гибкая. Например, можно реализовать сервер каталогов MS-DOS и сервер каталогов UNIX, которые будут использовать один и тот же файловый сервер для физического хранения файлов. Разделение этих функций также упрощает программное обеспечение. Недостатком является то, что использование двух серверов увеличивает интенсивность сетевого обмена.

Постоянный поиск имен, особенно при использовании нескольких серверов каталогов, может приводить к большим накладным расходам. В некоторых системах делается попытка улучшить производительность за счет кэширования имен. При открытии файла кэш проверяется на наличие в нем нужного имени. Если оно там есть, то этап поиска, выполняемый сервером каталогов, пропускается и двоичный адрес извлекается из кэша.

Последний рассматриваемый здесь структурный вопрос связан с хранением на серверах информации о состоянии клиентов. Существует две конкурирующие точки зрения.

Первая состоит в том, что сервер не должен хранить такую информацию (сервер stateless). Другими словами, когда клиент посылает запрос на сервер, сервер его выполняет, отсылает ответ, а затем удаляет из своих внутренних таблиц всю информацию о запросе. Между запросами на сервере не хранится никакой текущей информации о состоянии клиента. Другая точка зрения состоит в том, что сервер должен хранить такую информацию (сервер statefull).

Рассмотрим эту проблему на примере файлового сервера, имеющего команды *открыть*, *прочитать*, *записать* и *закрыть* файл. Открывая файлы, statefull-сервер должен запоминать, какие файлы открыл каждый пользователь. Обычно при открытии файла пользователю дается дескриптор файла или другое число, которое используется при последующих вызовах для его идентификации. При поступлении вызова сервер использует дескриптор файла для определения, какой файл нужен. Таблица, отображающая дескрипторы файлов на сами файлы, является информацией о состоянии клиентов.

Для сервера stateless каждый запрос должен содержать исчерпывающую информацию (полное имя файла, смещение в файле и т.п.), необходимую серверу для выполнения требуемой операции. Очевидно, что эта информация увеличивает длину сообщения.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Однако при отказе statefull-сервера теряются все его таблицы, и после перезагрузки неизвестно, какие файлы открыл каждый пользователь. Последовательные попытки провести операции чтения или записи с открытыми файлами будут безуспешными. Stateless-серверы в этом плане являются более отказоустойчивыми, и это аргумент в их пользу.

Преимущества обоих подходов можно обобщить следующим образом:

42. *Stateless-серверы:*

- отказоустойчивы;
- не нужны вызовы OPEN/CLOSE;
- меньше памяти сервера расходуется на таблицы;
- нет ограничений на число открытых файлов;
- отказ клиента не создает проблем для сервера.

43. *Statefull-серверы:*

- более короткие сообщения при запросах;
- лучше производительность;
- возможно опережающее чтение;
- легче достичь идемпотентности;
- возможна блокировка файлов.

3.5.5. Кэширование

В системах, состоящих из клиентов и серверов, потенциально имеется четыре различных места для хранения файлов и их частей: диск сервера, память сервера, диск клиента (если имеется) и память клиента. Наиболее подходящим местом для хранения всех файлов является диск сервера. Он обычно имеет большую емкость, и файлы становятся доступными всем клиентам. Кроме того, поскольку в этом случае существует только одна копия каждого файла, то не возникает проблемы согласования состояний копий.

Проблемой при использовании диска сервера является производительность. Перед тем как клиент сможет прочитать файл, файл должен быть переписан с диска сервера в его оперативную память, а затем передан по сети в память клиента. Обе передачи занимают время.

Значительное увеличение производительности может быть достигнуто за счет кэширования файлов в памяти сервера. Требуются алгоритмы для определения, какие файлы или их части следует хранить в кэш-памяти.

При выборе алгоритма должны решаться две задачи. Во-первых, какими единицами оперирует кэш. Этими единицами могут быть или дисковые блоки, или целые файлы. Если это целые файлы, то они могут храниться на диске непрерывными областями (по крайней мере в виде больших

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

участков), при этом уменьшается число обменов между памятью и диском, а следовательно, обеспечивается высокая производительность. Кэширование блоков диска позволяет более эффективно использовать память кэша и дисковое пространство.

Во-вторых, необходимо определить правило замены данных при заполнении кэш-памяти. Здесь можно использовать любой стандартный алгоритм кэширования, например алгоритм LRU (least recently used), в соответствии с которым вытесняется блок, к которому дольше всего не было обращения.

Кэш-память на сервере легко реализуется и совершенно прозрачна для клиента. Так как сервер может синхронизировать работу памяти и диска, с точки зрения клиентов существует только одна копия каждого файла, так что проблема согласования не возникает.

Хотя кэширование на сервере исключает обмен с диском при каждом доступе, все еще остается обмен по сети. Существует только один путь избавиться от обмена по сети — это кэширование на стороне клиента, которое, однако, порождает много сложностей.

Так как в большинстве систем используется кэширование в памяти клиента, а не на его диске, то мы рассмотрим только этот случай. При проектировании такого варианта имеется три возможности размещения кэша (рис. 3.10). Самый простой состоит в кэшировании файлов непосредственно внутри адресного пространства каждого пользовательского процесса. Обычно кэш управляется с помощью библиотеки системных вызовов. По мере того как файлы открываются, закрываются, читаются и пишутся, библиотека просто сохраняет наиболее часто используемые файлы. Когда процесс завершается, все модифицированные файлы записываются назад на сервер. Хотя эта схема реализуется с чрезвычайно низкими издержками, она эффективна только тогда, когда отдельные процессы часто повторно открывают и закрывают файлы. Таким является процесс менеджера базы данных, но обычные программы чаще всего читают каждый файл однократно, так что кэширование с помощью библиотеки в этом случае не дает выигрыша.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

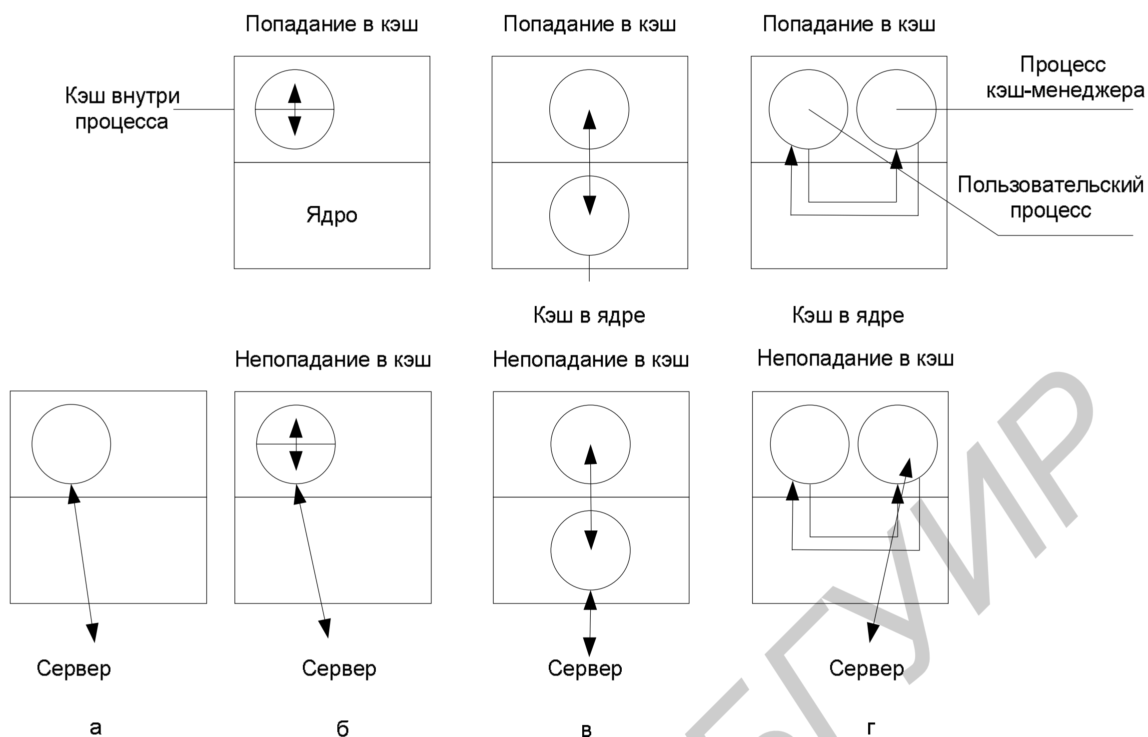


Рис. 3.10. Различные способы выполнения кэша в клиентской памяти: а — без кэширования; б — кэширование внутри каждого процесса; в — кэширование в ядре;
г — кэш-менеджер как пользовательский процесс

Другим местом кэширования является ядро. Недостаток этого варианта заключается в том, что во всех случаях требуется выполнять системные вызовы, даже в случае успешного обращения к кэш-памяти (файл оказался в кэше). Но преимуществом является то, что файлы остаются в кэше и после завершения процессов. Например, предположим, что двухпроходный компилятор выполняется как два процесса. Первый проход записывает промежуточный файл, который читается вторым проходом. На рисунке 3.10, в показано, что после завершения процесса первого прохода промежуточный файл, вероятно, будет находиться в кэше, так что вызов сервера не потребуется.

Третьим вариантом организации кэша является создание отдельного процесса пользовательского уровня — кэш-менеджера. Преимущество этого подхода заключается в том, что ядро освобождается от кода файловой системы и тем самым реализуются все достоинства микроядер.

С другой стороны, когда ядро управляет кэшем, оно может динамически решить, сколько памяти выделить для программ, а сколько для кэша. Когда же кэш-менеджер пользовательского уровня работает на машине с виртуальной памятью, то понятно, что ядро может решить выгрузить некоторые или даже все страницы кэша на диск, так что для так называемого "попадания в кэш" требуется подкачка одной или более страниц. Нечего и говорить, что это полностью дискредитирует идею

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

кэширования. Однако, если в системе имеется возможность фиксировать некоторые страницы в памяти, то такая парадоксальная ситуация может быть исключена.

Как и везде, нельзя получить что-либо, не заплатив чем-то за это. Кэширование на стороне клиента вносит в систему проблему несогласованности данных.

Одним из путей решения проблемы согласования является использование алгоритма сквозной записи. Когда кэшируемый элемент (файл или блок) модифицируется, новое значение записывается в кэш и одновременно посылается на сервер. Теперь другой процесс, читающий этот файл, получает самую последнюю версию.

Один из недостатков алгоритма сквозной записи состоит в том, что он уменьшает интенсивность сетевого обмена только при чтении, при записи интенсивность сетевого обмена та же самая, что и без кэширования. Многие разработчики систем находят это неприемлемым и предлагают следующий алгоритм, использующий отложенную запись: вместо того чтобы выполнять запись на сервер, клиент просто помечает, что файл изменен. Примерно каждые 30 секунд все изменения в файлах собираются вместе и отсылаются на сервер за один прием. Одна большая запись обычно более эффективна, чем много маленьких.

Следующим шагом в этом направлении является принятие сессионной семантики, в соответствии с которой запись файла на сервер производится только после его закрытия. Этот алгоритм называется "запись-по-закрытию". Как мы видели раньше, этот путь приводит к тому, что если две копии одного файла кэшируются на разных машинах и последовательно записываются на сервер, то вторая записывается поверх первой. Однако это не так уж плохо, как кажется на первый взгляд. В однопроцессорной системе два процесса могут открыть и читать файл, модифицировать его в своих адресных пространствах, а затем записать его назад. Следовательно, алгоритм "запись-по-закрытию", основанный на сессионной семантике, не намного хуже варианта, уже используемого в однопроцессорной системе.

Совершенно отличный от других подход к проблеме согласования — это использование алгоритма централизованного управления (подход соответствует семантике UNIX). Когда файл открыт, машина, открывшая его, посылает сообщение файловому серверу, чтобы оповестить его об этом факте. Файл-сервер сохраняет информацию о том, и кто какой файл открыл, и о том, открыт ли он для чтения, для записи, или для того и другого. Если файл открыт для чтения, то нет никаких препятствий для разрешения другим процессам открыть его для чтения, но открытие его для записи должно быть запрещено. Аналогично, если некоторый процесс открыл файл для записи, то все другие виды доступа должны быть предотвращены. При закрытии файла также необходимо оповестить файл-сервер для того, чтобы он обновил свои таблицы, содержащие данные об

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

открытых файлах. Модифицированный файл также может быть выгружен на сервер в такой момент.

Четыре алгоритма управления кэшированием обобщаются следующим образом:

44. *Сквозная запись.* Этот метод эффективен частично, так как уменьшает интенсивность только операций чтения, а интенсивность операций записи остается неизменной.
45. *Отложенная запись.* Производительность лучше, но результат чтения кэшированного файла не всегда однозначен.
46. *"Запись-по-закрытию".* Удовлетворяет сессионной семантике.
47. *Централизованное управление.* Ненадежен вследствие своей централизованной природы.

Подводя итоги обсуждения проблемы кэширования, нужно отметить, что кэширование на сервере несложно реализуется и почти всегда дает эффект, независимо от того, реализовано кэширование у клиента или нет. Кэширование на сервере не влияет на семантику файловой системы, видимую клиентом. Кэширование у клиента, напротив, дает увеличение производительности, но увеличивает и сложность семантики.

3.5.6. Репликация

Распределенные системы часто обеспечивают репликацию (тиражирование) файлов в качестве одной из услуг, предоставляемых клиентам. Репликация — это асинхронный перенос изменений данных исходной файловой системы в файловые системы, принадлежащие различным узлам распределенной файловой системы. Другими словами, система оперирует несколькими копиями файлов, причем каждая копия находится на отдельном файловом сервере. Имеется несколько причин для предоставления этого сервиса, главными из которых являются:

- увеличение надежности за счет наличия независимых копий каждого файла на разных файл-серверах;
- распределение нагрузки между несколькими серверами.

Как обычно, ключевым вопросом, связанным с репликацией, является прозрачность. До какой степени пользователи должны быть в курсе того, что некоторые файлы реплицируются? Должны ли они играть какую-либо роль в процессе репликации или репликация должна выполняться полностью автоматически? В одних системах пользователи полностью вовлечены в этот процесс, в других система все делает без их ведома. В последнем случае говорят, что система репликационно прозрачна.

На рис. 3.11 показаны три возможных способа репликации. При использовании первого способа (а) программист сам управляет всем процессом репликации. Когда процесс создает файл, он делает это на одном определенном сервере. Затем, если пожелает, он может сделать

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

дополнительные копии на других серверах. Если сервер каталогов разрешает сделать несколько копий файла, то сетевые адреса всех копий могут быть ассоциированы с именем файла, как показано на рисунке снизу, и когда имя найдено, это означает, что найдены все копии. Чтобы сделать концепцию репликации более понятной, рассмотрим, как может быть реализована репликация в системах, основанных на удаленном монтировании, типа UNIX. Предположим, что рабочий каталог программиста имеет имя */machine1/usr/ast*. После создания файла, например */machine1/usr/ast/xyz*, программист, процесс или библиотека могут использовать команду копирования, для того чтобы сделать копии */machine2/usr/ast/xyz* и *machine3/usr/ast/xyz*. Возможно программа использует в качестве аргумента строку */usr/ast/xyz* и последовательно попытается открывать копии, пока не достигнет успеха. Эта схема хотя и работает, но имеет много недостатков, и по этим причинам ее не стоит использовать в распределенных системах.

На рис. 3.11, б показан альтернативный подход — ленивая репликация. Здесь создается только одна копия каждого файла на некотором сервере. Позже сервер сам автоматически выполнит репликации на другие серверы без участия программиста. Эта система должна быть достаточно быстрой, для того чтобы обновлять все эти копии, если потребуется.

Последним рассмотрим метод, использующий групповые связи (рис. 3.11, в). В этом методе все системные вызовы *записать* передаются одновременно на все серверы, таким образом, копии создаются одновременно с созданием оригинала. Имеется два принципиальных различия в использовании групповых связей и ленивой репликации. Во-первых, при ленивой репликации адресуется один сервер, а не группа. Во-вторых, ленивая репликация происходит в фоновом режиме, когда сервер имеет промежуток свободного времени, а при групповой репликации все копии создаются в одно и то же время.

Рассмотрим, как могут быть изменены существующие реплицированные файлы. Существует два хорошо известных алгоритма решения этой проблемы.

Первый алгоритм, называемый "репликация первой копии", требует, чтобы один сервер был выделен как первичный. Остальные серверы являются вторичными. Когда реплицированный файл модифицируется, изменение посылается на первичный сервер, который выполняет изменения локально, а затем посылает изменения на вторичные серверы.

Чтобы предотвратить ситуацию, когда из-за отказа первичный сервер не успевает оповестить об изменениях все вторичные серверы, изменения должны быть сохранены в постоянном запоминающем устройстве еще до изменения первичной копии. В этом случае после перезагрузки сервера есть возможность проверить, не проводились ли какие-нибудь обновления в момент краха. Недостаток этого алгоритма типичен для централизованных систем — пониженная надежность. Чтобы избежать его, используется метод, предложенный Гиффордом и известный как "голосование". Пусть имеется n копий, тогда изменения должны быть внесены в любые W копий. При этом серверы, на которых хранятся копии,

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

должны отслеживать порядковые номера их версий. В случае, когда какой-либо сервер выполняет операцию чтения, он обращается с запросом к любым R серверам. Если R+W > n, то хотя бы один сервер содержит последнюю версию, которую можно определить по максимальному номеру.

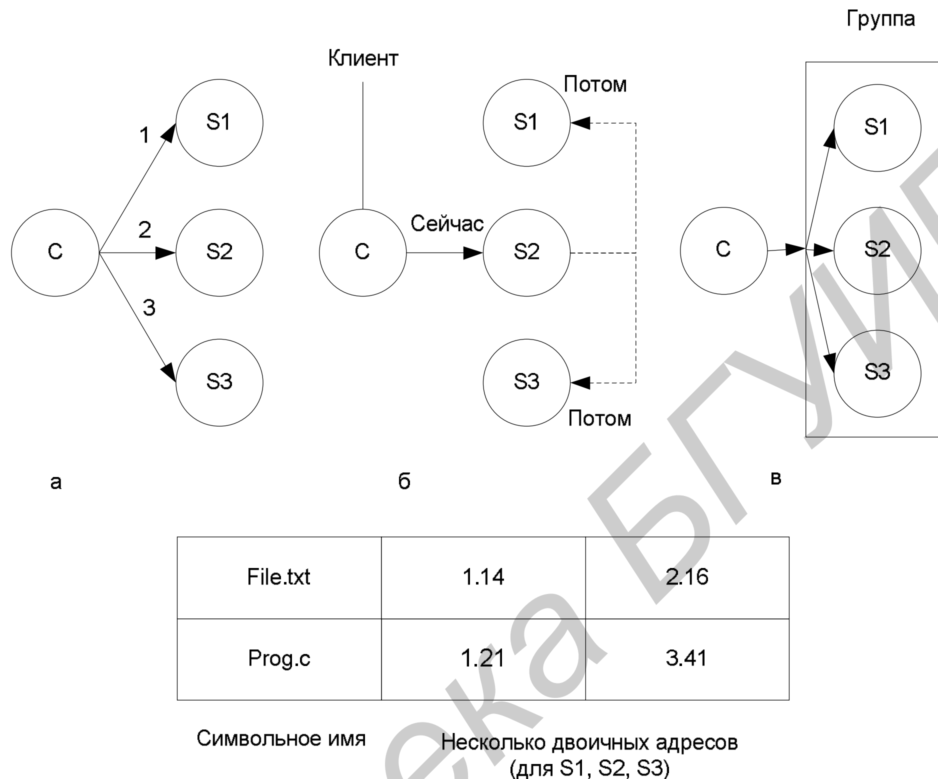


Рис. 3.11. Точная репликация файла (а); Ленивая репликация файла (б); Репликация файла, использующая группу (в)

Интересной модификацией этого алгоритма является алгоритм "голосование с приведениями". В большинстве приложений операции чтения встречаются гораздо чаще, чем операции записи, поэтому R обычно делают небольшим, а W — близким к N. При этом выход из строя нескольких серверов приводит к отсутствию кворума для записи. Голосование с приведениями решает эту проблему путем создания фиктивного сервера без дисков для каждого отказавшего или отключенного сервера. Фиктивный сервер не участвует в кворуме чтения (прежде всего, у него нет файлов), но он может присоединиться к кворуму записи, причем он просто записывает в никуда передаваемый ему файл. Запись только тогда успешна, когда хотя бы один сервер настоящий.

Когда отказавший сервер перезапускается, то он должен получить кворум чтения для обнаружения последней версии, которую он копирует к себе перед тем, как начать обычные операции. В остальном этот алгоритм подобен основному.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

3.6. Контрольные вопросы

48. Чем отличается взаимодействие процессов в рамках одного компьютера от их взаимодействия по сети?
49. В каких случаях целесообразно использовать ненадежные примитивы передачи сообщений?
50. В чем состоит основное назначение механизма RPC?
51. Почему в процедурах RPC не используются глобальные перменные?
52. Сравните два метода кэширования (на стороне клиента и на стороне сервера), используемые в сетевой файловой службе. Приведите достоинства и недостатки каждого метода.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

4. Операционная среда графодинамических ассоциативных машин

4.1. Графодинамические ассоциативные машины

Определение 4.1. Абстрактная машина обработки информации C задается парой:

$C = \langle M, O \rangle$,

где M – память абстрактной машины обработки информации, которая задается синтаксисом, семантикой и множеством начальных конструкций языка представления информации;

O – множество операций абстрактной машины обработки информации, которые определены над памятью.

Существуют множество различных абстрактных машин обработки информации, которые используются для описания формальной модели обработки информации, например:

- абстрактная машина Тьюринга;
- абстрактная машина Поста;
- MIX – абстрактная машина для описания алгоритмов обработки различных структур данных ***(Кнут Д.2000кн-ИскусП)***;
- РАМ (равнодоступная адресная машина) – для описания алгоритмов обработки графовых структур ***(Касьянов В.Н..2003кн-ГрафВП)***.

Определение 4.2. Графодинамические ассоциативные машины (ГАМ) – это класс абстрактных машин обработки информации, представленной в виде графодинамических структур. Память таких машин называется графодинамической ассоциативной памятью.

Определение 4.3. Графодинамическая структура – это абстрактный тип данных со следующими свойствами:

- элементами графодинамической структуры являются дуга или узел;
- связь между элементами графодинамической структуры не фиксирована и может меняться;
- значение элемента графодинамической структуры также может меняться.

4.2. Архитектура операционной среды графодинамических ассоциативных машин

Операционная среда графодинамических ассоциативных машин разбивается на конечное множество подсистем, выделяются следующие (рис.4.1) основные подсистемы:

- подсистема управления памятью;
- подсистема управления процессами;

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

- подсистема управления внешними устройствами;
- подсистема управления внешней памятью.

В операционной среде графодинамической ассоциативной машины выделяется набор традиционных подсистем (перечисленные выше четыре подсистемы) и навигационно-поисковая подсистема. Необходимость выделения такой подсистемы обусловлена спецификой применения графодинамических ассоциативных машин – обработка знаний, представленных в виде семантических сетей. Основу операционной среды графодинамической ассоциативной машины составляет модель графодинамической ассоциативной памяти.

При распределенной реализации операционной среды на каждом компьютере располагается некоторое множество подсистем. Основу этого множества составляют подсистемы управления памятью и подсистема коммуникации.

Множество подсистем, находящихся на выделенном компьютере, будем называть **модулем**. Модули классифицируются по роли, которые они выполняют. В разработанной операционной среде выделяют два типа модулей: процессорный и терминальный. Процессорный модуль исполняет роль вычислителя, а терминальный модуль – роль пользовательского терминала.

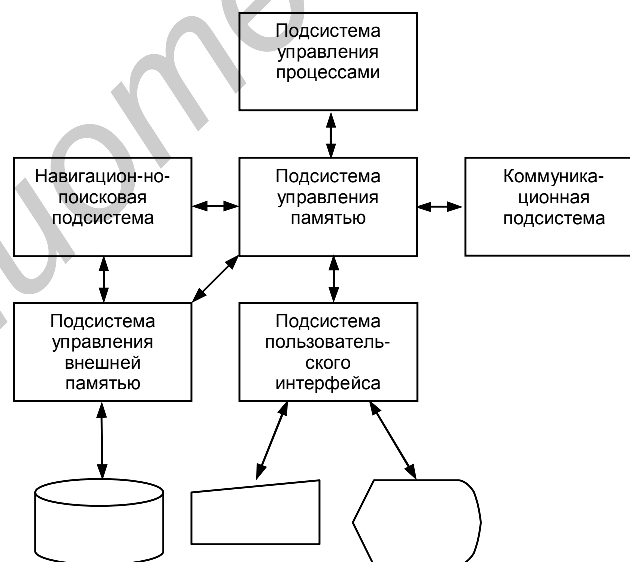


Рис. 4.1. Обобщенная схема взаимодействия подсистем ОС ГАМ

На рис.4.2 показано распределение конкретных подсистем на модули при распределенной реализации.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

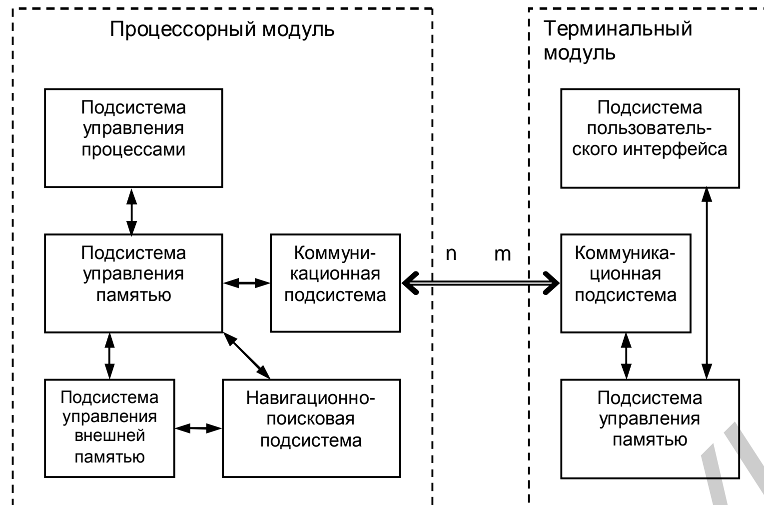


Рис. 4.2. Общая схема взаимодействия подсистем при распределенной реализации

Соотношение процессорного модуля к терминальному модулю n к m. Разработанная архитектура позволяет реализовать многопользовательскую распределенную операционную среду.

Ниже рассмотрим основные подсистемы операционной среды графодинамической ассоциативной машины и разработанные способы, которые применяются в этих подсистемах.

4.3. Подсистема управления памятью графодинамических ассоциативных машин

Подсистема управления памятью графодинамических ассоциативных машин выполняет аналогичные функции, как и подсистема управления памятью традиционных компьютеров. Для начала необходимо выделить основные операции над графодинамической ассоциативной памятью.

Выделяются следующие базовые операции над графодинамической ассоциативной памятью:

- генерация узлов (константных, переменных и метапеременных);
- генерация дуг (позитивных, негативных и нечетких; константных, переменных и метапеременных);
- генерация элементов неопределенного типа (константных, переменных и метапеременных);
- изменение идентификатора узла, дуги или элемента неопределенного типа;
- установка, изменение и удаление содержимого узла;
- получение списка выходящих дуг из узла;
- получения списка входящих дуг в узел, дугу или элемент неопределенного типа;

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

- проверка типа узла, дуги или элемента неопределенного типа;
- арифметические и логические операции над содержимым узла;
- установка обработчика событий в графодинамической ассоциативной памяти.

Перечисленный выше список является базовым. Все остальные действия над графодинамической ассоциативной памятью состоят из этих базовых операций.

4.4. Подсистема управления процессами в графодинамических ассоциативных машинах

Определение 4.4. Под процессом в графодинамических ассоциативных машинах понимается абстракция, описывающая текущее состояние выполнения операций графодинамических ассоциативных машин.

4.4.1. Описание процесса в графодинамической ассоциативной машине

Определение 4.5. Контекст процесса в ГАМ описывается следующим отношением:

process -> qi ={ active_ : opi , program_ : pi , fatherProcess_ : qj , fatherOp_ : opj , varValue_ : ri , segment_ : sseg , blkG_ : sbg , blkS_ : sbs , blkGE_ : sbge , blkSE_ : sbse } ; (4.1)

где opi – узел текущего выполняемого оператора программы pi;
pi – узел программы, которую выполняет данный процесс qi;
qj – узел процесса, который породил процесс qi (этот элемент может отсутствовать);
opj – узел оператора, результатом выполнения которого было создание процесса qi (этот элемент может отсутствовать);
ri – множество значений переменных, вычисленных в рамках процесса qi;
sseg – множество открытых сегментов в рамках процесса qi;
sbg – множество g-блокированных sc-элементов в рамках процесса qi;
sbs – множество s-блокированных sc-элементов в рамках процесса qi;
sbge – множество ge-блокированных sc-элементов в рамках процесса qi;
sbse – множество se-блокированных sc-элементов в рамках процесса qi.

В каждый момент времени в ГАМ может существовать неограниченное количество одновременно выполняемых процессов. Процесс может находиться в одном из следующих состояний:

- в активном состоянии;
- в состоянии ожидания какого-либо ресурса ГАМ;

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

- в состоянии ошибки;
- в состоянии прерывания.

При создании процесса указывается его приоритет, возможно три значения приоритета: ***kernel***, ***system***, ***user***. Для каждого значения приоритета существует своя очередь. В разработанном методе смена приоритета процессом не предусматривается.

4.4.2. Алгоритм управления процессами в графодинамических ассоциативных машинах

В подсистеме управления процессами за основу взят алгоритм невытесняющей многозадачности. Его достоинством является простота реализации. Также применяется алгоритм, учитывающий при планировании выделяемый квант времени и приоритет процесса. Идея этого алгоритма управления процессами такова: для определения следующего на выполнение процесса используется приоритет, активную роль по смене процессов играет сам процесс (невытесняющая многозадачность), но процесс сам следит за продолжительностью своего выполнения (квантование). Такой алгоритм ориентируется только на применение в программной модели операционной среды.

Рассмотрим алгоритм планирования процессов в графодинамической ассоциативной машине.

Шаг 1. При создании процесса ему присваивается приоритет (***kernel***, ***system***, ***user***).

Шаг 2. Созданный процесс включается в очередь согласно своему приоритету.

Шаг 3. Выбор следующего на выполнение процесса осуществляется по принципу: ***kernel*** – ***kernel*** – ***system*** – ***kernel*** - ***system*** - ***user***. Соотношение частоты выполнения процессов из разных очередей получается 3/2/1.

Шаг 4. Выбранному процессу передается, как параметр, число, которое обозначает квант времени.

Шаг 5. После выполнения очередного шага процесс проверяет, есть ли у него еще время, и если нет, то передает управление планировщику (переход к шагу 3).

Шаг 6. Если процесс завершился, то управление передается планировщику (переход к шагу 3).

4.4.3. Организация взаимодействия процессов в графодинамических ассоциативных машинах

Основу организации взаимодействия процессов в ГАМ составляют события и блокировки. Первый механизм позволяет обрабатывать

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

различные события в памяти (создание, удаление и изменения элемента), второй позволяет реализовать механизм транзакций.

Существует возможность обрабатывать следующие события:

- создание узла, дуги или элемента неопределенного типа в заданном сегменте;
- удаление узла, дуги или элемента из заданного сегмента;
- изменение идентификатора элемента в рамках заданного сегмента.

Определение 4.6. Сообщение – это конструкция, которая имеет следующий вид:

$$\begin{aligned} \textbf{\textit{message -> msg}} = \{ \quad & \textbf{\textit{src_ : vs ,}} \\ & \textbf{\textit{dst_ : vd ,}} \\ & \textbf{\textit{data_ : vdt}} \\ \} ; & \end{aligned} \qquad (4.2)$$

где ***vs*** – узел, обозначающий адрес источника сообщения;
vd – узел, обозначающий адрес приемника сообщения;
vdt – узел, обозначающий данные сообщения.

Адрес в сообщении описывается двумя элементами: узлом, обозначающим почтовый ящик, и узлом, обозначающим модуль, где находится этот почтовый ящик.

Определение 4.7. Почтовый ящик содержит входящую и выходящую очередь сообщения и имеет следующий вид:

$$\begin{aligned} \textbf{\textit{mailbox -> mb}} = \{ \quad & \textbf{\textit{incomming_ : vqi ,}} \\ & \textbf{\textit{outgoing_ : vqo ,}} \\ & \textbf{\textit{handler_ : ph}} \\ \} ; & \end{aligned} \qquad (4.3)$$

где ***vqi*** – узел, обозначающий очередь входящих сообщений;
vqo – узел, обозначающий очередь выходящих сообщений;
ph – узел, обозначающий программу, которая обрабатывает событие "появление сообщений во входящей очереди".

Метод организации взаимодействия процессов на основе сообщений и почтовых ящиков позволяет взаимодействовать распределенным процессам.

4.5. Подсистема управления внешними устройствами графодинамических ассоциативных машин

Внешним устройством графодинамических ассоциативных машин является любое программное обеспечение, которое обеспечивает ввод, хранение или переработку информации. Внешним устройством может быть СУБД, анализатор сетевого трафика и т.д.

Для организации взаимодействия с внешним устройством используется такое понятие, как драйвер графодинамических ассоциативных машин.

PDF created with pdfFactory Pro trial version www.pdffactory.com

Драйвер обеспечивает преобразование запросов из ГАМ в формат внешнего устройства и обратно. Это необходимо для того, чтобы программы ГАМ могли взаимодействовать с внешними устройствами посредством базовых операций над графодинамической ассоциативной памятью (создать узел, удалить узел, найти множество дуг и т.д.). Общая схема взаимодействия ГАМ с внешними устройствами изображена на рис.4.3.

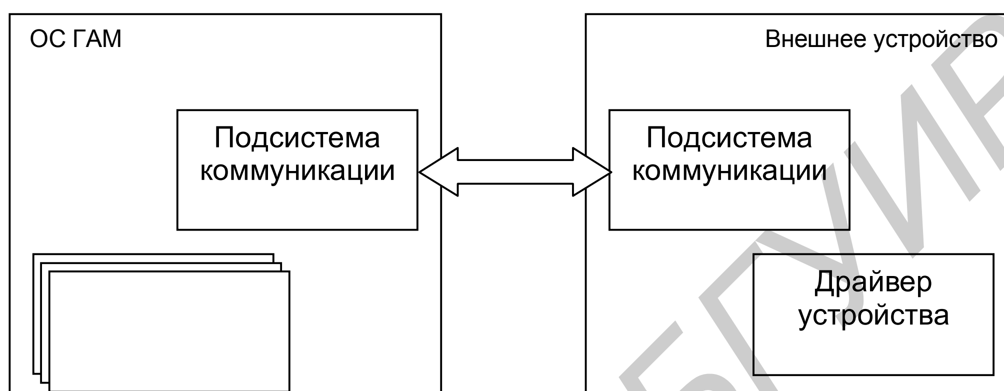


Рис. 4.3. Общая схема взаимодействия с внешними устройствами

Между модулем ОС ГАМ и внешним устройством устанавливается коммуникационная связь, передача информации по ней осуществляется в формате TGF.

Возможен также другой метод организации взаимодействия с внешними устройствами. Суть этого метода заключается в использовании виртуального представления. Виртуальное представление – это сегмент графодинамической ассоциативной памяти особого вида при работе, с которым все запросы на изменения переадресуются внешнему устройству. Такой метод обеспечивает высокую степень абстрагирования от конкретного представления информации.

В операционной среде применяется два метода хранения графодинамической ассоциативной памяти: на файловой системе и в таблицах реляционных СУБД.

Первый метод осуществляет хранение графодинамической ассоциативной памяти на любой файловой системе, это достигается использованием стандартных библиотек ввода-вывода. В отдельном файле хранится сегмент графодинамической ассоциативной памяти в формате TGF. Каталог соответствует сегменту-каталогу графодинамической ассоциативной памяти.

Все сегменты ГАП делятся на временные и постоянные сегменты. Постоянные хранятся на внешних носителях и при обращении к элементам

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

в этих сегментах подгружаются в память. Временные сегменты всегда находятся в оперативной памяти.

Рассмотренный метод прост в реализации, но неэффективен для операций поиска. Идея второго метода – использовать методы и средства реляционных баз данных для хранения больших баз знаний.

4.6. Навигационно-поисковая подсистема графодинамической ассоциативной машины

Навигационно-поисковая подсистема предназначена для навигации и поиска по текущему состоянию базы знаний. Эффективность работы этой подсистемы влияет на эффективность всей ОС ГАМ, так как основу любой прикладной системы составляют операции формирование и поиска информации.

Основу навигационно-поисковой подсистемы составляет язык запросов, пример запроса на этом языке и соответствующего результата показан на рис.4.4.

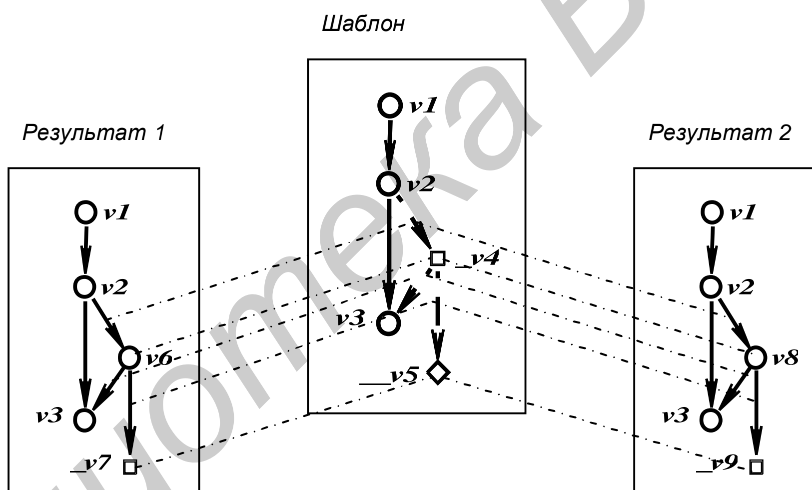


Рис. 4.4. Общая схема взаимодействия с внешними устройствами

Более подробно навигационно-поисковая подсистема графодинамической ассоциативной машины рассмотрена в работе [6].

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

5. Базовый язык программирования для графодинамических ассоциативных машин

Данный раздел посвящен рассмотрению языка системного программирования для графодинамических ассоциативных машин – SCP (Semantic Code Programming). Язык SCP относится к классу графовых языков программирования. Особенностью этого языка является то, что не только данные, но и сами программы, написанные на языке SCP, представляются в виде графодинамических структур. Язык SCP является языком параллельного асинхронного программирования. Это необходимо для того, чтобы обеспечить адекватную интерпретацию не только последовательных, но и параллельных, не только синхронных, но и асинхронных формальных моделей.

5.1. Принципы языка SCP (Semantic Code Programming)

Язык SCP относится к классу процедурных языков параллельного программирования, ориентированных на переработку нечисловой информации.

Особенностями языка SCP по сравнению с другими языками указанного класса являются:

53. Приспособленность к переработке нечисловых структур, имеющих мощные выразительные возможности, т.е. обеспечивающих описание любых сложноструктурированных предметных областей.
54. Использование структурно перестраиваемой графодинамической ассоциативной памяти. Как тексты языка SCP (scp-процедуры), так и перерабатываемые данные являются графовыми конструкциями, хранимыми в графовой структурно перестраиваемой ассоциативной памяти (это означает, что язык SCP относится к классу графовых языков).
55. Ориентация на переработку непосредственно семантических сетей, а не структур (например списковых), с помощью которых семантические сети кодируются (это означает, что язык SCP описывает переработку информации непосредственно на уровне семантического кодирования).
56. Возможность описания (с помощью scp-процедур) механизмов решения задач (логических операций, операций вывода), имеющих различный уровень сложности и поддерживающих самые различные стратегии решения задач.
57. Высокий потенциал распараллеливания процессов переработки информации в графовой структурно перестраиваемой ассоциативной памяти (как на уровне реализации различных логических операций, так и на уровне реализации каждой из этих операций).
58. Возможность через понятие sc-узла интегрировать процедуры над sc-конструкциями с процедурами, написанными на традиционных языках и

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

описывающими переработку обычных информационных объектов. Это обеспечивается тем, что содержимым sc-узла может быть информационный объект любой природы.

Из всех вышеперечисленных отличительных особенностей языка SCP самой существенной является ориентация на принципиально иную модель памяти – ориентация на использование графодинамической ассоциативной памяти.

Ближайшими аналогами языка SCP являются:
59. Язык ВОЛНА (WAVE) П.С. Сапатого.
60. Язык передачи маркеров, используемый в качестве внутреннего языка параллельного компьютера маркерной обработки семантических сетей (marker-passing parallel computer).

Принципы языка SCP хорошо сочетаются с идеологией языков и моделей продукционного (ситуационного) программирования, к числу которых относятся нормальные алгоритмы Маркова, язык РЕФАЛ, всевозможные формальные грамматики.

Тест языка SCP с формальной точки зрения есть определенным образом устроенная sc-конструкция, хранимая в sc-памяти и описывающая некоторую систему параллельных взаимодействующих процессов переработки другой sc-конструкции, хранимой в той же sc-памяти.

Язык SCP разрабатывается и реализуется в нескольких модификациях, имеющих одинаковый набор непосредственно выполняемых операций, но различный синтаксис. В 1993 г. был реализован интерпретатор, обеспечивающий выполнение всех операций, включенных в текущую версию языка SCP. Для каждой из модификаций языка SCP дополнительно к указанному интерпретатору требуется реализовать конвертор из соответствующей формы представления scp-процедур в форму, поддерживаемую интерпретатором. Основная модификация языка SCP характеризуется графовым синтаксисом. Другая его модификация, называемая SCPas (Semantic Code Programming assembler), имеет синтаксис языка ассемблерного типа. Каждая процедура на языке SCPas представляет собой последовательность scpas-операторов, содержащих код операции и перечень операндов, причем каждой операции языка SCP, непосредственно выполняемой интерпретатором, в языке SCPas соответствует свой код операции. Конвертор из SCPas в форму scp-программ, поддерживаемую интерпретатором, имеет достаточно простой вид.

Другие модификации языка SCP характеризуются более удобным для программиста синтаксисом, что связано с относительно небольшим набором кодов операций и большим или меньшим (различным для разных модификаций) разнообразием операторных выражений. Конверторы из таких форм представления scp-процедур в форму, поддерживаемую интерпретатором языка SCP, имеют большое сходство с компиляторами традиционных и относительно простых алгоритмических языков высокого

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

уровня, таких как Бейсик или Паскаль. Из модификаций рассматриваемого класса следует упомянуть SCPbs (Semantic Code Programming basic – базовый SCP), а также SCPso (Semantic Code Programming source – исходный SCP). Язык SCPso отличается от всех остальных версий языка SCP более высокими выразительными возможностями, позволяющими создавать более компактные и наглядные программы.

5.2. Описание ядра языка SCP

5.2.1. Основные средства ядра языка SCP

Программы на языке SCP (scp-программы) представляются в виде специальным образом устроенных sc-конструкций, поэтому язык SCP следует считать подъязыком языка SC, т.е. одним из конкретных языков, построенных на базе SC.

Описание заголовка программы языка SCP

Язык представления данных для scp-программ полностью совпадает с языком SC. Данными scp-программ являются sc-конструкции произвольного вида, и в частности конструкции любого графового языка, построенного на базе языка SC, например, конструкции логического языка SCL и даже конструкции самого языка SCP. SCP-программа может входить в состав данных по отношению к какой-либо другой scp-программе либо по отношению к самой себе. SCP-программа представляет собой соединение sc-конструкции, которая является декларативной частью (заголовком) этой scp-программы (см. пример 5.1), а также sc-конструкций, являющихся операторами scp-программы. Соединение sc-конструкций предполагает склеивание синонимичных sc-элементов соединяемых sc-конструкций.

При представлении конструкций языка SCP (scp-операторов и заголовков scp-программ), как и при представлении конструкций любого другого графового языка, используется определенный набор ключевых узлов (см. подраздел 5.3), т.е. узлов, имеющих определенную семантику и обеспечивающих расшифровку произвольных конструкций соответствующего графового языка. Использование ключевых узлов в графовых языках программирования аналогично использованию ключевых слов в символьных языках программирования. Ключевыми узлами языка SCP являются также идентификаторы всех операторов этого языка.

Перейдем к рассмотрению синтаксиса языка SCP.

Заголовок scp-программы (обозначим его $\boldsymbol{pi}$) (см. пример 5.1) состоит из следующих элементов:

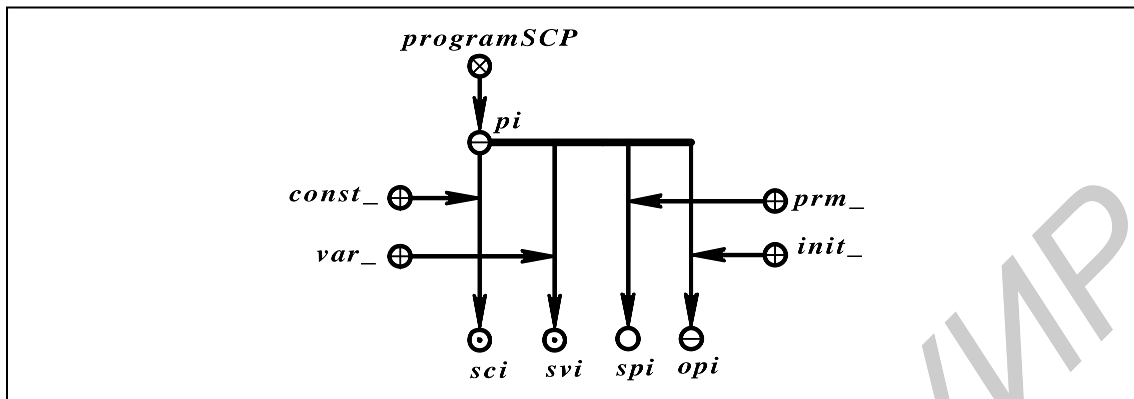
- знака множества параметров scp-программы (обозначим его $\boldsymbol{sy}$);

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

- знака множества констант scp-программы (обозначим его ***sc***);
- знака множества программных переменных scp-программы (обозначим его ***sv***);
- знака первого scp-оператора scp-программы (обозначим его ***opi***).

Пример 5.1. Заголовок scp-программы



Множество параметров scp-программы (***sv***) – это упорядоченное множество программных переменных этой scp-программы. Это множество упорядочено атрибутами ***1_***, ***2_***, ***3_*** и т.д. Параметры, значения которых scp-программа (***pi***) получает из вызвавшей её программы, уточняются атрибутом ***in_***. Параметры, значение которых scp-программа (***pi***) возвращает вызвавшей её программе, уточняется атрибутам ***out_***. Возвращение значений параметров происходит при вызове ***return***-операторов (см. п. 5.2.7).

Множество констант scp-программы (scp-константы) – это множество константных sc-элементов. Значением scp-константы является узел, обозначающий scp-константу.

Множество программных переменных scp-программы (scp-переменных) – это множество переменных sc-элементов. Значением переменной scp-программы является sc-элемент, связанный бинарным ориентированным отношением с sc-узлом, обозначающим scp-переменную.

Примечание. Значением переменного sc-узла может быть только sc-узел, значением переменной sc-дуги может быть только sc-дуга, значением переменного sc-элемента неопределенного типа может быть как sc-узел, так и sc-дуга.

Описание структуры оператора языка SCP

Операторы языка SCP (scp-операторы) разбиваются на типы. Множество типов scp-операторов в свою очередь разбивается на семейства типов scp-операторов. В ядре языка SCP выделено семь семейств типов scp-операторов. Перечислим идентификаторы ключевых узлов, каждый из которых обозначает соответствующее семейство scp-операторов:

- ***gen*** (семейство типов scp-операторов генерации sc-конструкций);
- ***erase*** (семейство типов scp-операторов удаления sc-конструкций);

Библиотека БГУИР

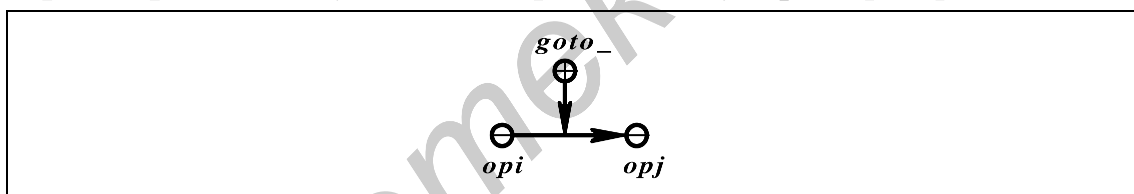
PDF created with pdfFactory Pro trial version www.pdffactory.com

- ***search*** (семейство типов scp-операторов ассоциативного поиска sc-конструкций);
- ***if*** (семейство типов scp-операторов проверки условий);
- ***change*** (семейство типов scp-операторов изменения свойств sc-элементов);
- ***manage*** (семейство типов scp-операторов управления scp-процессами).

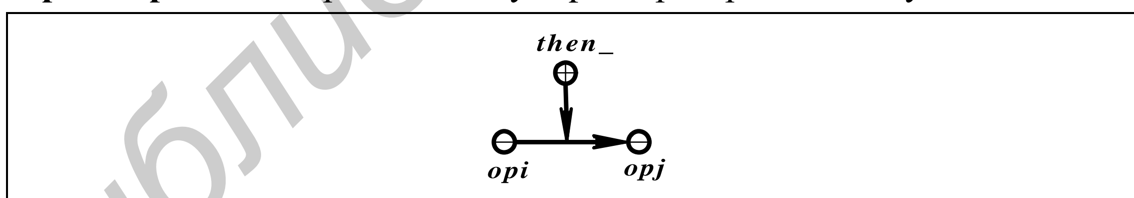
В языке SCP оператор формально определяется как кортеж операндов. Операторы языка SCP разных типов имеют разное количество операндов. В частности, в некоторых scp-операторах операнды могут отсутствовать. По характеру передачи управления scp-операторы делятся на три категории:

- операторы, при реализации которых осуществляется безусловная передача управления следующему реализуемому оператору (см. пример 5.2);
- операторы, при реализации которых осуществляется условная передача управления одному из двух операторов (см. пример 5.3, пример 5.4);
- операторы, завершающие реализацию scp-программ, т.е. операторы, при реализации которых не осуществляется передача управления одному из операторов этой же scp-программы.

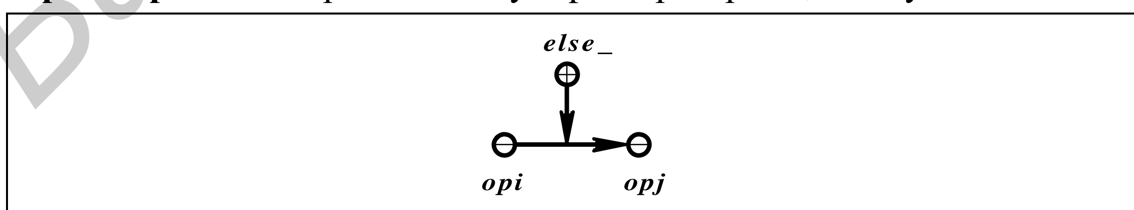
Пример 5.2. Безусловный переход между scp-операторами



Пример 5.3. Переход между scp-операторами, если условие истинно



Пример 5.4. Переход между scp-операторами, если условие ложно



Если рассмотреть scp-операторы, входящие в состав одной scp-программы, то их также можно разбить на следующие категории:

- scp-оператор, которому управление от других scp-операторов не передается, т.е. это scp-оператор, который инициируется первым сразу

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

после инициирования всей scp-программы. Напомним, что первый реализуемый scp-оператор (***opi***) scp-программы (***pi***) задается конструкцией

pi ; ***init_*** : ***opi*** ;

- scp-операторы, каждому из которых управление передается только от одного scp-оператора scp-программы;
- scp-операторы, каждому из которых управление передается от нескольких scp-операторов scp-программы. Наличие таких scp-операторов означает либо наличие циклов, либо слияние каких-либо альтернативных ветвей scp-программы.

Операнд scp-оператора scp-программы может быть или scp-константой, или scp-переменной этой scp-программы, т.е. входить в одно из этих sc-множеств. Операнд scp-оператора может уточняться одним или несколькими атрибутами.

Атрибуты операнда scp-оператора разбиваются на следующие группы:

- группа атрибутов порядка (***1_***, ***2_***, ***3_***, ***4_*** и т.д.), которые определяют порядок операндов;
- группа атрибутов уточнения значения операнда (***fixed_***, ***assign_***), которые определяют, известно ли значение операнда (***fixed_***) или же его надо сформировать (***assign_***);
- группа атрибутов уточнения типа значения операнда (***const_*** , ***var_*** , ***pos_***, ***neg_***, ***fuz_***, ***node_***, ***elem_***, ***arc_***, ***meta_***), которые задают тип значения операнда.

Ниже приведем перечень основных атрибутов операнда с их семантикой;

- ***fixed_*** – определяет, что значение операнда сформировано;
- ***assign_*** – определяет, что значение операнда не сформировано и его надо сформировать (за исключением тех случаев, когда операнд является scp-шаблоном);
- ***const_*** – уточняет, что значение операнда должно являться sc-константой;
- ***var_*** – уточняет, что значение операнда должно являться sc-переменной;
- ***pos_*** – уточняет, что значение операнда должно являться позитивной sc-дугой;
- ***neg_*** – уточняет, что значение операнда должно являться негативной sc-дугой;
- ***fuz_*** – уточняет, что значение операнда должно являться нечеткой sc-дугой;
- ***node_*** – уточняет, что значение операнда должно являться sc-узлом;
- ***elem_*** – уточняет, что значение операнда должно являться sc-элементом неопределенного типа;
- ***arc_*** – уточняет, что значение операнда должно являться sc-дугой;

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

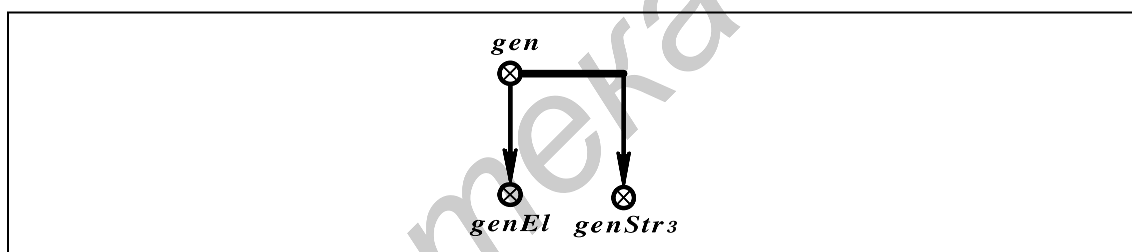
- ***meta_*** – уточняет, что значение операнда должно являться sc-метапеременной;
- ***f_*** – уточняет, что значение операнда должно быть удалено;

5.2.2. Операторы генерации конструкций языка SC

Семейство scp-операторов генерации sc-конструкций (такие операторы будем также называть gen-операторами) разбивается на следующие типы scp-операторов:

- операторы генерации sc-элемента (genEl-операторы);
- операторы генерации sc-конструкций, состоящей из трёх элементов (genStr3-операторы).

Множество genEl-операторов в языке SCP будем обозначать ключевым узлом, которому поставим в соответствие идентификатор ***"genEl"***. Множество genStr3-операторов в языке SCP будем обозначать ключевым узлом, которому поставим в соответствие идентификатор ***"genStr3"***. Семейство типов gen-операторов в языке SCP обозначается ключевым узлом с идентификаторов ***"gen"***. Соотношение между указанными ключевыми узлами задается следующей sc-конструкцией:



Перейдем к более подробному рассмотрению указанных выше типов gen-операторов.

Операторы генерации sc-элемента (genEl-операторы, операторы типа *genEl*)

Операторы данного типа осуществляют генерацию sc-узла или sc-элемента неопределенного типа.

Запись scp-операторов типа ***genEl*** на языке SCs имеет следующий вид (этот вид описан на метаязыке Бэкуса-Наура, см. соответствующее приложение):

запись genEl –оператора ::=

genEl ;

идентификатор

genEl-оператора ;

(* ***node_*** : | ***elem_*** : *) /* данные атрибуты

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

уточняют тип генерируемого sc-элемента по признаку "узел – элемент неопределенного типа" */

(* ***const_ :*** | ***var_ :*** | ***meta_ :*** *) /* данные атрибуты уточняют тип генерируемого sc-элемента по признаку "константа – переменная – метапеременная" */

□ ***операнд*** □ **,** /* операндом является программная переменная, значением которой становится генерируемый sc-элемент */

goto_ : □ ***идентификатор scp-оператора*** □ **;** /* здесь указывается идентификатор того scp-оператора, которому передается управление после реализации данного scp-оператора */

Результаты выполнения операторов типа ***genEl***:

61. Если значение операнда не вычислено (т.е. программной переменной, которая является операндом в текущий момент времени не присвоено никакого значения), то генерируется новый sc-элемент. Тип генерируемого элемента определяется атрибутами операнда:
    - ***node_*** – генерируется sc-узел;
    - ***elem_*** – генерируется sc-элемент неопределенного типа;
    - ***const_*** – генерируется константный sc-элемент;
    - ***var_*** – генерируется переменный sc-элемент;
    - ***meta_*** – генерируется метапеременный sc-элемент.

    Сгенерированный sc-элемент становится значением операнда (это означает, что проводится соответствующая бинарная ориентированная sc-дуга, соединяющая операнд с его значением).
62. Если значение операнда вычислено и тип sc-элемента, являющегося значением операнда, соответствует типу, задаваемому атрибутом операнда, то в результате реализации оператора изменения состояние памяти не происходит. Иначе генерируется новый sc-элемент требуемого типа (см. выше) и он становится значением операнда (sc-дуга, соединяющая операнд с его предыдущим значением, удаляется).

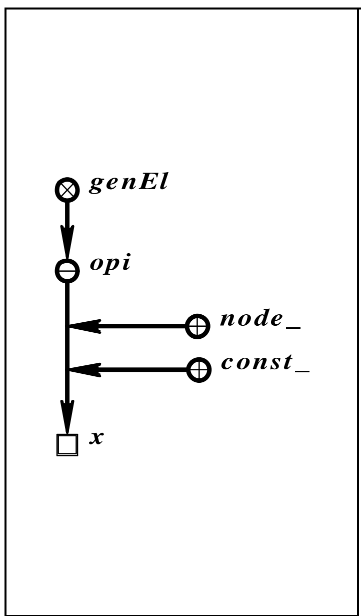
Примечание. В качестве операнда данного scp-оператора может использоваться только программная переменная.

Библиотека БГУИР

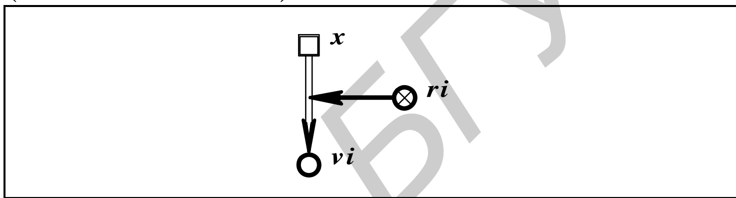
PDF created with pdfFactory Pro trial version www.pdffactory.com

Модификации scp-операторов типа ***genEl*** задаются комбинацией атрибутов, которыми помечен операнд. Всего для genEl-оператора существует 6 модификаций. Приведем некоторые из них:

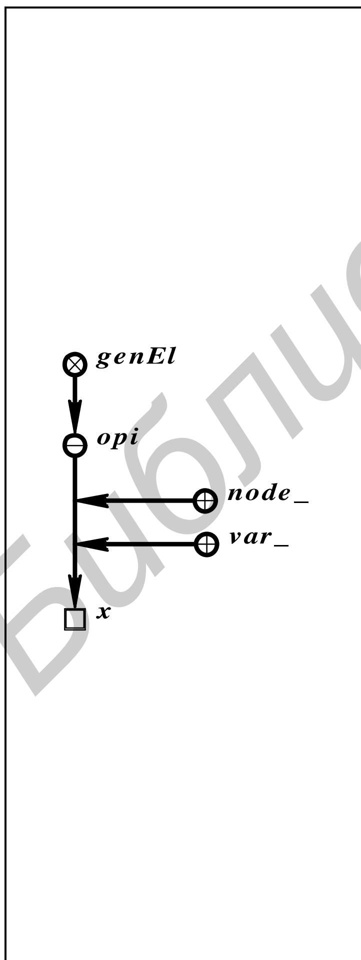
Пример 5.5



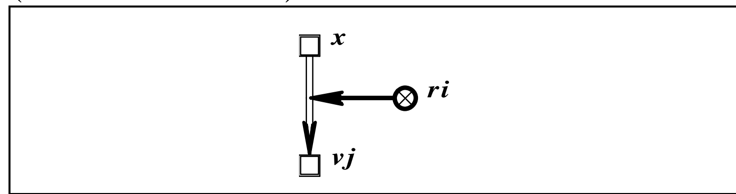
После успешного выполнения данного оператора (***opi***) будет сгенерирован константный sc-узел (***vi***), также будет сгенерирована дуга, связывающая операнд ***x*** с sc-узлом ***vi*** , и дуга принадлежности, связывающая указанную выше дугу со знаком соответствующего бинарного отношения (***ri***). Генерация этой дуги означает, что у операнда ***x*** сформировалось (было вычислено) значение:



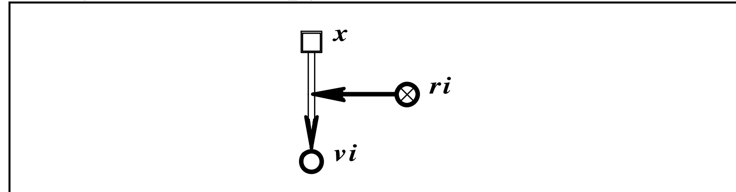
Пример 5.6



После успешного выполнения данного оператора (***opi***) будет сгенерирован переменный sc-узел (***vj***), также будет сгенерирована дуга, связывающая операнд ***x*** с sc-узлом ***vj*** , и дуга принадлежности, связывающая указанную выше дугу со знаком соответствующего бинарного отношения (***ri***). Генерация этой дуги означает, что у операнда ***x*** сформировалось (было вычислено) значение:



Если перед началом выполнения данного оператора (***opi***) в памяти находится следующая конструкция:

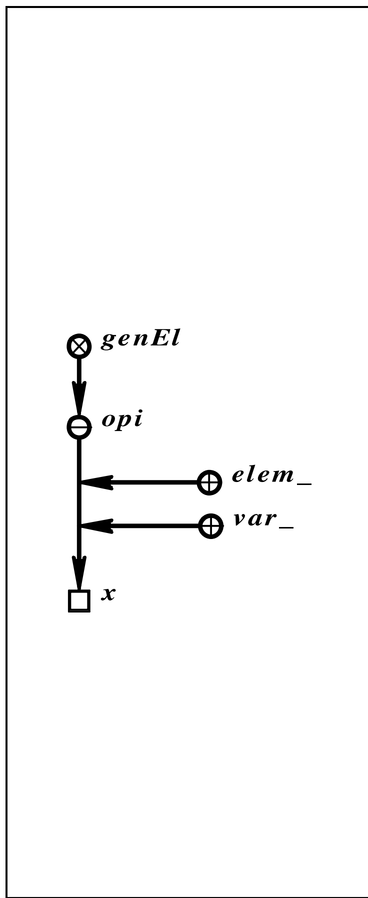


Библиотека БГУИР

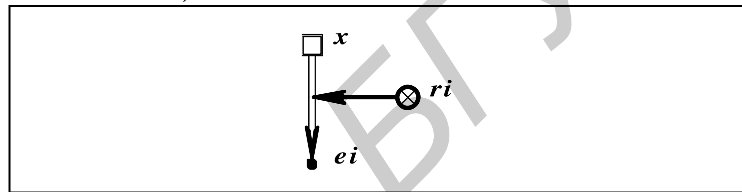
PDF created with pdfFactory Pro trial version www.pdffactory.com

То после успешного выполнения оператора ***opi*** дуга между операндом ***x*** и sc-узлом ***vi*** будет удалена.

Пример 5.7



После успешного выполнения данного оператора (***opi***) будет сгенерирован переменный sc-элемент (***ei***) неопределенного типа; также будет сгенерирована дуга, связывающая операнд ***x*** с sc-элементом ***ei*** , и дуга принадлежности, связывающая указанную выше дугу со знаком соответствующего бинарного отношения (***ri***). Генерация этой дуги означает, что у операнда ***x*** сформировалось (было вычислено) значение:



Здесь и далее ***ri*** – бинарное ориентированное отношение, связывающее программные переменные scp-программы, в которую входит оператор ***opi***, с их текущими значениями.

Перечислим все модификации genEl-операторов:

- операторы генерации константного sc-узла;
- операторы генерации переменного sc-узла;
- операторы генерации метапеременного sc-узла;
- операторы генерации константного sc-элемента неопределенного типа;
- операторы генерации переменного sc-элемента неопределенного типа;
- операторы генерации метапеременного sc-элемента неопределенного типа.

Операторы генерации sc-конструкции, состоящей из трёх элементов (genStr3-операторы, операторы типа *genStr3*)

Операторы данного типа осуществляют генерацию sc-конструкции, состоящей из трех элементов, или каких-либо элементов, входящих в состав этой конструкции.

Запись scp-операторов типа ***genStr3*** на языке SCs имеет следующий вид:

□ ***запись genStr3 –оператора*** □

::=

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

genStr3 ;

□ ***идентификатор genStr3-оператора*** □ ;

(* ***fixed_*** : | ***assign_*** : — /* данные атрибуты уточняют, известно (вычислено) ли значение операнда или необходимо сгенерировать sc-узел и установить значение операнда данного scp-оператора */

(* ***const_*** : | ***var_*** : | ***meta_*** : *) *) — /* данные атрибуты уточняют тип генерируемого sc-элемента по признаку "константа – переменная – метапеременная" */

1_ : □ ***операнд x1*** □ , — /* операндом является scp-переменная, если указан атрибут ***assign_***, значением которой становится генерируемый sc-узел. Если указан атрибут ***fixed_***, то операндом может быть и scp-переменная и scp-константа */

(* ***fixed_*** : | ***assign_*** : — /* данные атрибуты уточняют, известно (вычислено) ли значение операнда или необходимо сгенерировать sc-дугу и установить значение операнда данного scp-оператора */

(* ***const_*** : | ***var_*** : | ***meta_*** : *) *) — /* данные атрибуты уточняют тип генерируемого sc-элемента по признаку "константа – переменная – метапеременная" */

(* ***pos_*** : | ***neg_*** : | ***fuz_*** : *) — /* данные атрибуты уточняют тип генерируемой sc-дуги по признаку "позитивная дуга – негативная дуга – нечеткая дуга" */

2_ : □ ***операнд x2*** □ , — /* операндом является scp-переменная, если указан

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

	атрибут ***assign_***, значением, которой становится генерируемая sc-дуга. Если указан атрибут ***fixed_***, то операндом должна быть scp-переменная, а её значением sc-элемент – неопределенного типа */
(* ***fixed_ :*** | ***assign_ :***	/* данные атрибуты уточняют, известно (вычислено) ли значение операнда или необходимо сгенерировать sc-элемент и установить значение операнда данного оператора */
(* ***node_ :*** | ***elem_ :*** *)	/* данные атрибуты уточняют тип генерируемого sc-элемента по признаку "узел – элемент неопределенного типа" */
(* ***const_ :*** | ***var_ :*** | ***meta_ :*** *) *)	/* данные атрибуты уточняют тип генерируемого sc-элемента по признаку "константа – переменная – метапеременная" */
3_ : □ ***операнд x3*** □ ,	/* операндом является scp-переменная, если указан атрибут ***assign_***, значением которой становится генерируемый sc-элемент. Если указан атрибут ***fixed_***, то операндом может быть и scp-переменная, и scp-константа */
goto_ : □ ***идентификатор scp-оператора*** □ ;	/* здесь указывается идентификатор того scp-оператора, которому передается управление после реализации данного scp-оператора */

Результаты выполнения операторов типа ***genStr3***:

63. Если первый операнд помечен атрибутом assign_, то будет сгенерирован новый sc-узел. Тип сгенерированного sc-узла определяется следующими атрибутами:
    - const_ – генерируется константный sc-узел;
    - var_ – генерируется переменный sc-узел;

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

- meta_ – генерируется метапеременный sc-узел.

Сгенерированный sc-узел становится значением операнда (это означает, что проводится соответствующая бинарная ориентированная дуга, соединяющая операнд с его значением).

64. Если первый операнд помечен атрибутом fixed_ и его значением является sc-элемент неопределенного типа, то этот sc-элемент неопределенного типа будет преобразован в sc-узел. Тип sc-узла будет таким же, как тип sc-элемента неопределенного типа.
65. Если третий операнд помечен атрибутом assign_, то будет сгенерирован новый sc-элемент, тип которого определяется следующими атрибутами:
    - node_ – генерируется sc-узел;
    - elem_ – генерируется sc-элемент неопределенного типа;
    - const_, var_, meta_ – генерируется соответственно константный, переменный или метапеременный sc-элемент.

    Сгенерированный sc-элемент становится значением операнда (это означает, что проводится соответствующая бинарная ориентированная дуга, соединяющая операнд с его значением).
66. Если второй операнд помечен атрибутом assign_, то будет сгенерирована новая sc-дуга, выходящая из sc-узла, являющегося значением первого операнда, и входящая в sc-элемент, являющийся значением третьего операнда. Тип сгенерированной sc-дуги определяется следующими атрибутами:
    - pos_ – генерируется позитивная sc-дуга;
    - neg_ – генерируется негативная sc-дуга;
    - fuz_ – генерируется нечеткая sc-дуга;
    - const_, var_, meta_ – генерируется соответственно константная, переменная или метапеременная sc-дуга.

    Сгенерированная sc-дуга становится значением операнда (это означает, что проводится соответствующая бинарная ориентированная дуга, соединяющая операнд с его значением).
67. Если второй операнд помечен атрибутом fixed_ и его значением является sc-элемент неопределенного типа, то этот sc-элемент неопределенного типа будет преобразован в sc-дугу, выходящую из sc-узла, являющегося значением первого операнда, и входящую в sc-элемент, являющийся значением третьего операнда. Тип sc-дуги по признаку “константа – переменная – метапеременная” будет таким же, как и у sc-элемента неопределенного типа. Тип sc-дуги по признаку "позитивная дуга – негативная дуга – нечеткая дуга" определяется следующими атрибутами:
    - pos_ – генерируется позитивная sc-дуга;
    - neg_ – генерируется негативная sc-дуга;
    - fuz_ – генерируется нечеткая sc-дуга.

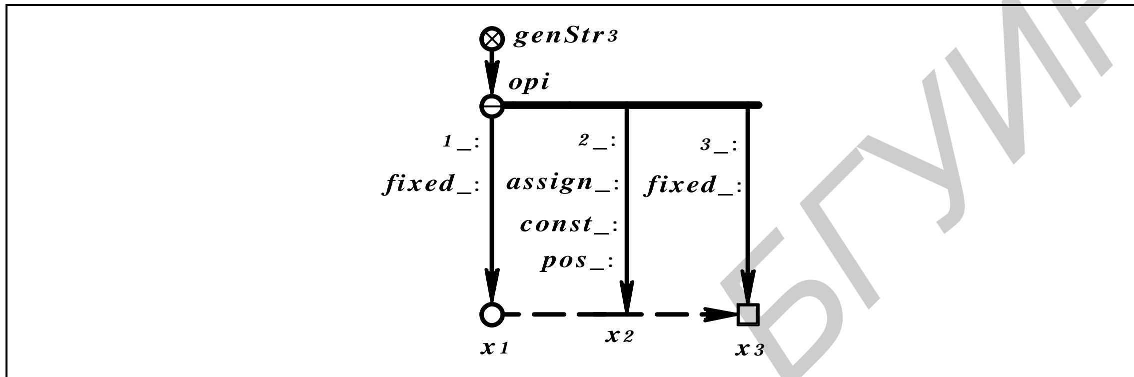
Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

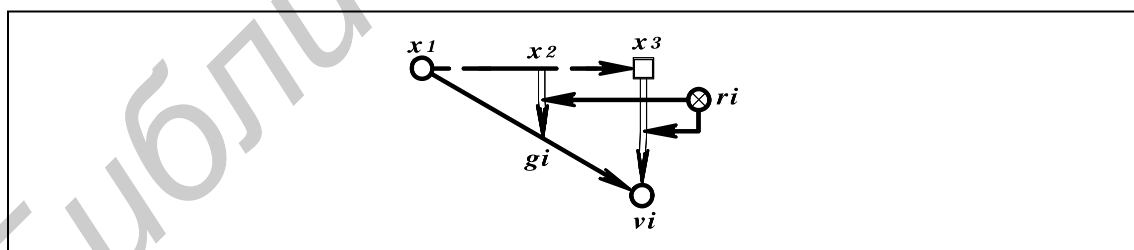
Примечание. Если первый операнд помечен атрибутом ***fixed_***, то его значением должен быть sc-узел или sc-элемент неопределенного типа. Если значением первого операнда является sc-элемент неопределенного типа, то после реализации данного scp-оператора этот sc-элемент неопределенного типа преобразуется в sc-узел. Если третий операнд помечен атрибутом ***fixed_***, то его значением может быть или sc-узел, или sc-дуга, или sc-элемент неопределенного типа.

Модификации scp-операторов типа ***genStr3*** задаются комбинацией атрибутов, которыми помечен операнд. Приведем некоторые из модификаций genStr3-оператора.

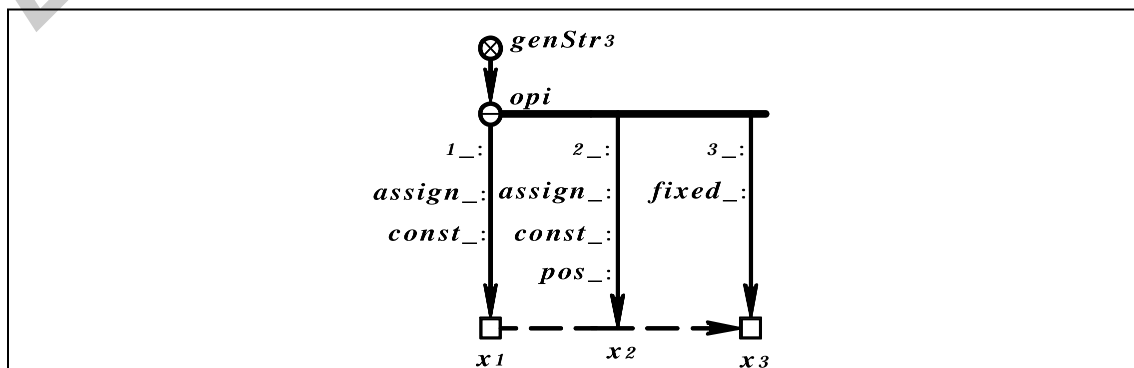
Пример 5.8



После успешного выполнения данного scp-оператора (***opi***) будет сгенерирована константная позитивная sc-дуга (***gi***), соединяющая значение операнда ***x1*** (так как операнд ***x1*** является scp-константой, то его значением является он сам) и sc-узел ***vi***, который является значением операнда ***x3***. Также будет сгенерирована sc-дуга, связывающая операнд ***x2*** с sc-дугой ***gi***, а также дуга принадлежности, связывающая указанную выше дугу со знаком соответствующего бинарного отношения (***ri***). Генерация этой дуги означает, что у операнда ***x2*** сформировалось (было вычислено) значение:



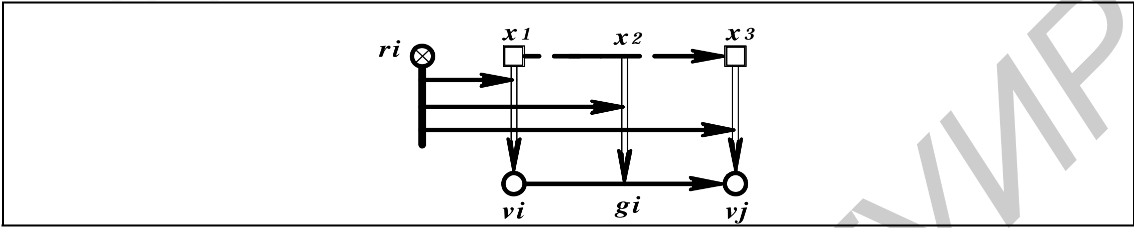
Пример 5.9



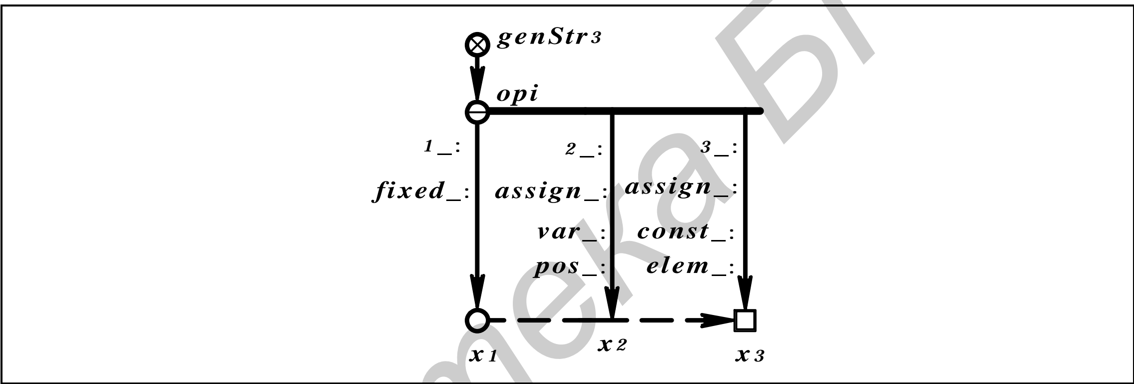
Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

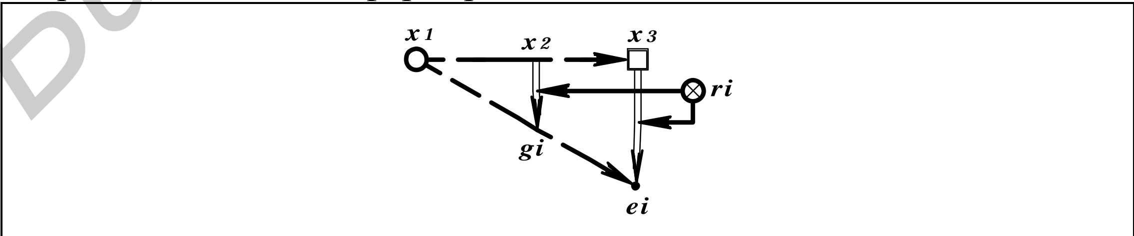
После успешного выполнения данного оператора (***opi***) будут сгенерированы два константных позитивных sc-узла (***vi*** и ***vj***) и соединяющая их дуга (***gi***). Также будут сгенерированы дуги, связывающие операнды ***x1, x2*** и ***x3*** с sc-элементами ***vi***, ***gi*** и ***vj*** соответственно, а также дуги принадлежности, связывающие указанные выше дуги со знаком соответствующего бинарного отношения (***ri***). Генерация этих дуг означает, что у операндов ***x1, x2*** и ***x3*** сформировались (вычислились) значения.



Пример 5.10



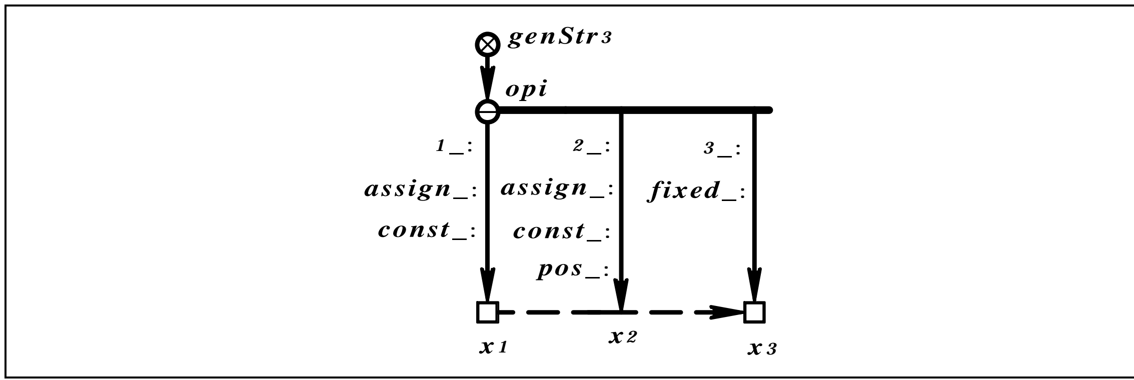
После успешного выполнения данного оператора (***opi***) будут сгенерированы константный sc-элемент неопределенного типа (***ei***) и переменная позитивная sc-дуга (***gi***), соединяющая операнд ***x1*** и сгенерированный sc-элемент ***ei***. Также будут сгенерированы дуга, связывающая операнд ***x2*** с sc-элементом ***gi***, дуга связывающая операнд ***x3*** с sc-элементом ***gi***, а также дуги принадлежности, связывающие указанные выше дуги со знаком соответствующего бинарного отношения (***ri***). Генерация этих дуг означает, что у операндов ***x2*** и ***x3*** сформировалось (вычислилось) значение:



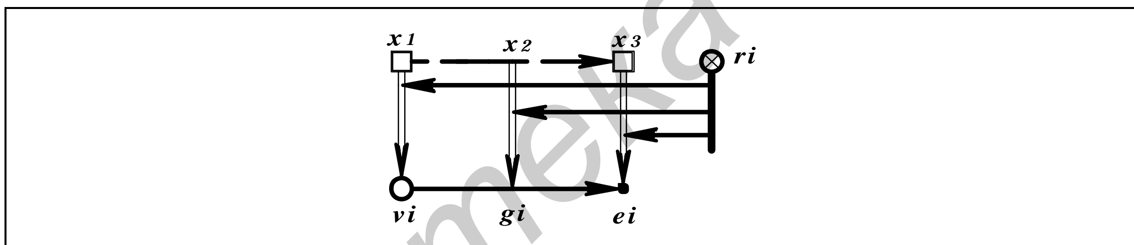
Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

П р и м е р 5.11



После успешного выполнения данного оператора (***opi***) будут сгенерированы константный sc-узел ***vi*** и константная позитивная sc-дуга (***gi***), соединяющая sc-узел ***vi*** и sc-элемент неопределенного типа ***ei***, который является значением операнда ***x3***. Также будут сгенерированы дуга связывающая операнд ***x2*** с sc-узлом ***vi***, дуга, связывающая операнд ***x2*** с sc-элементом ***gi***, а также дуги принадлежности, связывающие указанные выше дуги со знаком соответствующего бинарного отношения (***ri***). Генерация этих дуг означает, что у операндов ***x1*** и ***x2*** сформировалось (вычислилось) значение:



5.2.3. Операторы удаления конструкции языка SC

Семейство scp-операторов удаления sc-конструкций (такие операторы будем также называть erase-операторами) разбивается на следующие типы scp-операторов:

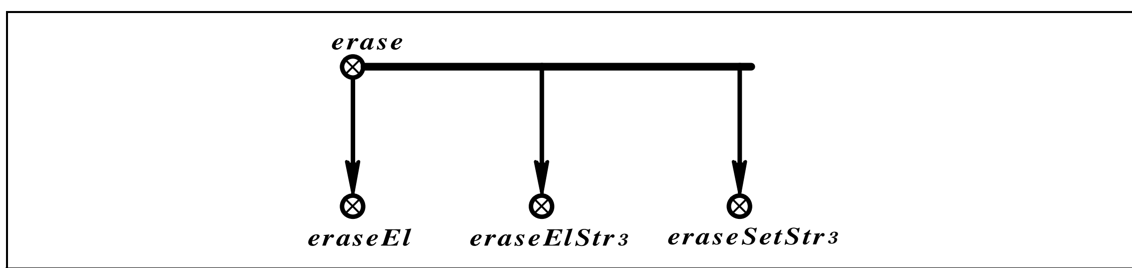
- операторы удаления указанного sc-элемента (eraseEl-операторы);
- операторы удаления указанных элементов sc-конструкции, состоящей из трех элементов (eraseElStr3-операторы);
- операторы удаления нескольких указанных элементов sc-конструкции, состоящей из трех элементов (eraseSetStr3-операторы).

Множество eraseEl-операторов в языке SCP будем обозначать ключевым узлом, которому поставим в соответствие идентификатор ***“eraseEl”***. Множество eraseElStr3-операторов в языке SCP будем обозначать ключевым узлом, которому поставим в соответствие идентификатор ***“eraseElStr3”***. Множество eraseSetStr3-операторов в языке SCP будем обозначать ключевым узлом, которому поставим в соответствие идентификатор ***“eraseSetStr3”***. Семейство типов erase-операторов в языке SCP обозначается ключевым узлом с идентификатором ***“erase”***.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Соотношение между указанными ключевыми узлами задается следующей sc-конструкцией:



Перейдем к более подробному рассмотрению указанных выше типов scp-операторов.

Операторы удаления указанного sc-элемента (eraseEl-операторы, операторы типа *eraseEl*)

Операторы данного типа осуществляют удаление sc-элемента.

Запись scp-операторов типа ***eraseEl*** на языке SCs имеет следующий вид:

 запись eraseEl -оператора **::=**

eraseEl **;**

 идентификатор eraseEl-оператора **;**

 операнд **,** /* операндом является программная переменная, значение которой удаляется */

goto_ **:** ***идентификатор scp-оператора*** **;** /* здесь указывается идентификатор того scp-оператора, которому передается управление после реализации данного scp-оператора */

Результат выполнения операторов типа ***eraseEl***:

- удаляется sc-элемент, который является значением операнда, также удаляются все sc-дуги, входящие в этот элемент и выходящие из него (если значением операнда является sc-узел).

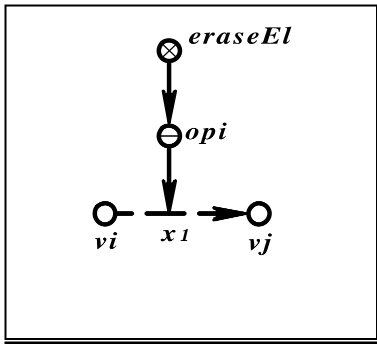
Примечание. При удалении sc-элемента также удаляется бинарное ориентированное отношение, связывающее этот sc-элемент с sc-узлом, обозначающим scp-переменную, которая является операндом данного scp-оператора. Таким образом, значение scp-переменной становится невычисленным.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

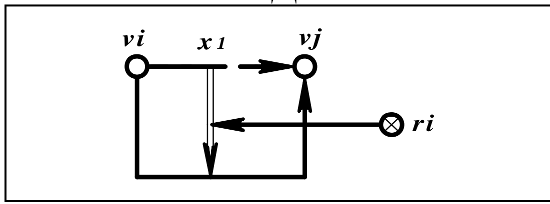
Существует только одна модификация scp-операторов типа ***eraseEl***. Приведем примеры использования данного оператора.

Пример 5.12

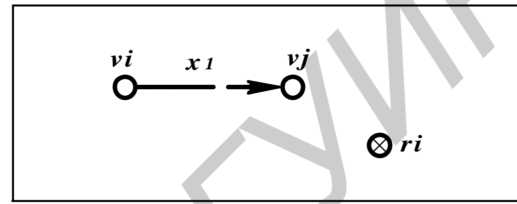


После успешного выполнения данного оператора (***opi***) будет удалена дуга, которая является значением операнда ***x2***. Также будут удалены: дуга, связывавшая операнд ***x2*** с его значением, и дуга принадлежности, связывавшая вышеупомянутую дугу со знаком соответствующего бинарного отношения (***ri***).

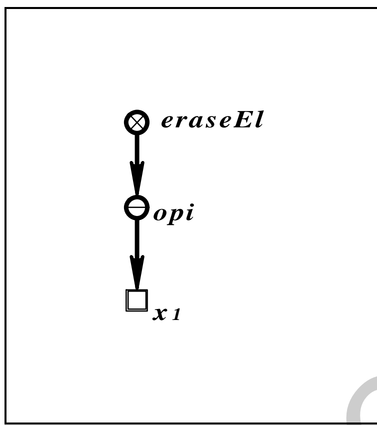
До



После

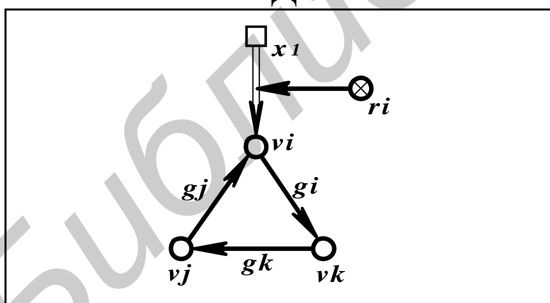


Пример 5.13

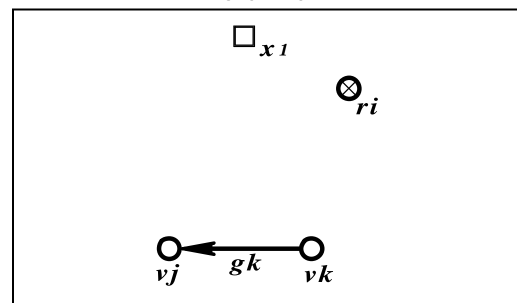


После успешного выполнения данного оператора (***opi***) будет удален узел ***vi***, который является значением операнда ***x1***. Также будут удалены входящие и выходящие из данного узла дуги (***gi*** и ***gj***), дуга, связывающая переменную ***x1*** с её значением, и дуга принадлежности, связывавшая эту дугу со знаком соответствующего бинарного отношения (***ri***).

До



После



Операторы удаления указанных элементов sc-конструкции, состоящей из трех элементов (eraseElStr3-операторы, операторы типа *eraseElStr3*)

Операторы данного типа осуществляют удаление sc-конструкции, состоящей из трех элементов, или каких-либо элементов, входящих в состав этой конструкции.

Запись scp-операторов типа ***eraseElStr3*** на языке SCs имеет следующий вид:

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

□ ***запись eraseElStr3 –оператора*** □
::=
eraseElStr3 ;
□ ***идентификатор eraseElStr3-оператора*** □ ;

(* ***fixed_ :*** |	/* данный атрибут уточняет, известно (вычислено) ли значение операнда данного scp-оператора */
[* (* ***const_ :*** | ***var_ :*** | ***meta_ :*** *) *] *)	/* данные атрибуты уточняют тип удаляемого (искомого) sc-узла по признаку «константа – переменная – метапеременная» */
[* ***f_ :*** *]	/* данный атрибут означает, что значение операнда данного scp-оператора или найденный sc-узел должен быть удален */
1_ : □ ***операнд x1*** □ ,	/* операндом является программная переменная, значение которой удаляется, если указан атрибут ***f_*** */
(* ***fixed_ :*** |	/* данный атрибут уточняет, известно (вычислено) ли значение операнда данного scp-оператора */
[* (* ***pos_ :*** | ***neg_ :*** | ***fuz_ :*** *) *]	/* данные атрибуты уточняют тип удаляемой (искомой) sc-дуги по признаку «позитивная дуга – негативная дуга – нечеткая дуга» */

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

[* (* ***const_ :*** | ***var_ :*** | ***meta_ :*** *) *] *)	/* данные атрибуты уточняют тип удаляемой (искомой) sc-дуги по признаку «константа – переменная – метапеременная» */
[* ***f_ :*** *]	/* данный атрибут означает, что значение операнда данного scp-оператора или найденная sc-дуга должны быть удалены */
2_ : □ ***операнд x2*** □ **,**	/* операндом является программная переменная, значение которой удаляется, если указан атрибут ***f_*** */
(* ***fixed_ :*** |	/* данный атрибут уточняет, известно (вычислено) ли значение операнда данного scp-оператора */
(* [* ***node_ :*** | ***elem_ :*** | ***arc_ :*** *]	/* данные атрибуты уточняют тип удаляемого (искомого) sc-элемента по признаку «узел – элемент неопределенного типа – дуга» */
[* (* ***pos_ :*** | ***neg_ :*** | ***fuz_ :*** *) *] . *)	/* данные атрибуты уточняют тип удаляемой (искомой) sc-дуги по признаку «позитивная дуга – негативная дуга – нечеткая дуга» */
[* (* ***const_ :*** | ***var_ :*** | ***meta_ :*** *) *] *)	/* данные атрибуты уточняют тип удаляемой (искомой) sc-дуги по признаку «константа – переменная – метапеременная» */
[* ***f_ :*** *]	/* данный атрибут означает, что значение операнда данного scp-оператора или найденная sc-дуга должна быть удалена */

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

3_* : □ *операнд x3* □ ,** /* операндом является программная переменная, значение которой удаляется, если указан атрибут ***f_ */

***goto_* : □ *идентификатор scp-оператора* □ ;** /* здесь указывается идентификатор того scp-оператора, которому передается управление после реализации данного scp-оператора */

Результаты выполнения операторов типа ***eraseElStr3***:

68. Если первый операнд не помечен атрибутом fixed_, то будет искаться sc-узел. Тип искомого sc-узла уточняется следующими атрибутами:
    - const_ – ищется константный sc-узел;
    - var_ – ищется переменный sc-узел;
    - meta_ – ищется метапеременный sc-узел.
69. Если искомый sc-узел найден и указан атрибут ***f_***, то этот sc-узел удаляется.
70. Если первый операнд помечен атрибутом ***fixed_***, а также атрибутом ***f_***, то значение операнда удаляется.
71. Если второй операнд не помечен атрибутом ***fixed_***, то будет искаться sc-дуга. Тип искомой sc-дуги уточняется следующими атрибутами:
    - ***pos_*** – ищется позитивная sc-дуга;
    - ***neg_*** – ищется негативная sc-дуга;
    - ***fuz_*** – ищется нечеткая sc-дуга;
    - ***const_, var_, meta_*** – ищется соответственно константная, переменная или метапеременная sc-дуга.
72. Если искомая sc-дуга найдена и указан атрибут ***f_***, то эта sc-дуга удаляется.
73. Если второй операнд помечен атрибутом ***fixed_***, а также атрибутом ***f_***, то значение операнда удаляется.
74. Если третий элемент не помечен атрибутом ***fixed_***, то будет искаться sc-элемент, тип которого уточняется следующими атрибутами:
    - ***node_*** – ищется sc-узел;
    - ***elem_*** – ищется sc-элемент неопределенного типа;
    - ***arc_*** – ищется sc-дуга. В этом случае для уточнения типа могут быть дополнительно использованы атрибуты ***pos_***, ***neg_***, ***fuz_*** (семантику смотри выше);

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

- ***const_***, ***var_***, ***meta_*** – ищется соответственно константный, переменный или метапеременный sc-элемент.

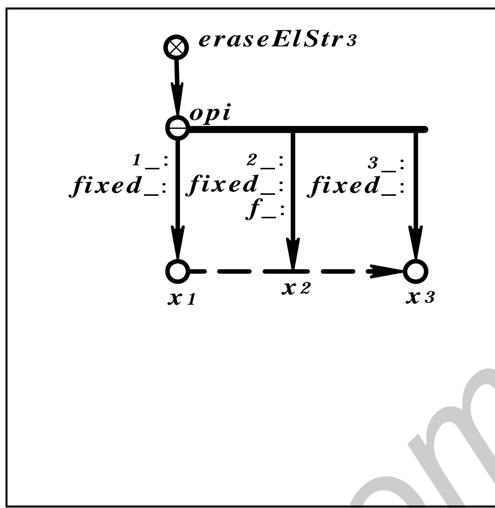
75. Если искомый sc-элемент найден и указан атрибут ***f_***, то этот sc-элемент удаляется.
76. Если третий операнд помечен атрибутом ***fixed_***, а также атрибутом ***f_***, то значение операнда удаляется.

Примечание 1. Хотя бы один операнд данного оператора должен быть помечен атрибутом ***fixed_***.

Примечание 2. Если ни один операнд scp-оператора типа ***eraseElStr3*** не помечен атрибутом ***f_***, то в результате реализации такого оператора изменения состояния памяти не происходит.

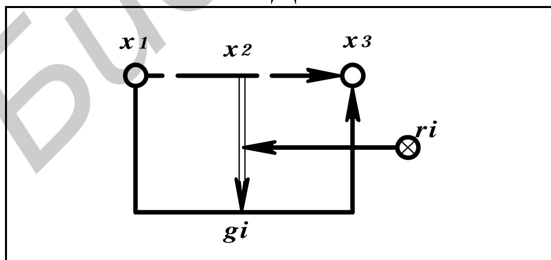
Модификации scp-операторов типа ***eraseElStr3*** задаются комбинацией атрибутов, которыми помечен операнд. Приведем некоторые модификации eraseElStr3-оператора:

Пример 5.14

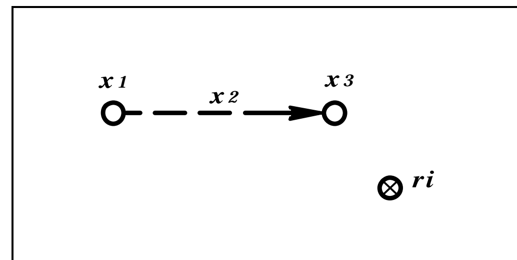


После успешного выполнения данного оператора (***opi***) будет удалена sc-дуга ***gi***, которая является значением операнда ***x2***. Также будут удалены: дуга, связывавшая операнд ***x2*** с его значением, и дуга принадлежности, связывавшая данную дугу со знаком соответствующего бинарного отношения (***ri***). Удаление этой дуги означает, что у операнда ***x2*** отсутствует (удалено) значение.

До



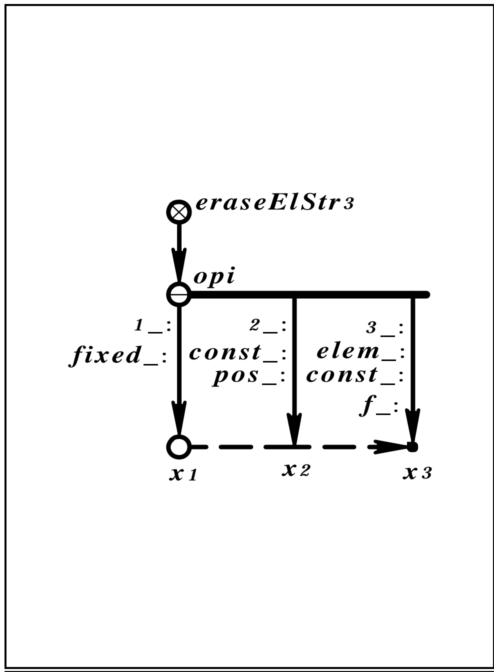
После



Библиотека БГУИР

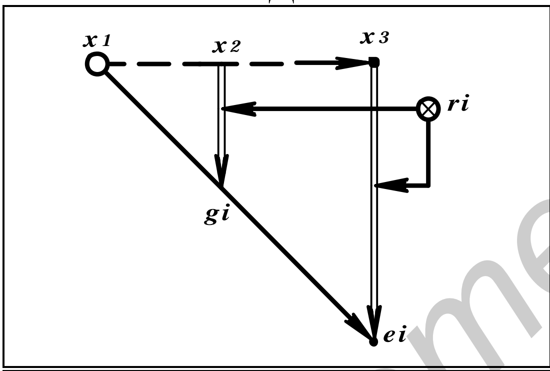
PDF created with pdfFactory Pro trial version www.pdffactory.com

Пример 5.15

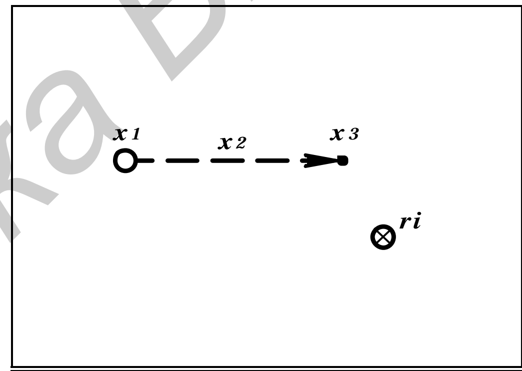


После успешного выполнения данного оператора (***opi***) будет удален константный sc-элемент неопределенного типа, который является значением операнда ***x3*** . Также будет удалена входящая в него константная позитивная sc-дуга, которая является значением операнда ***x2***; дуги, связывавшие операнды ***x2*** и ***x3*** с их значениями; дуги принадлежности, связывавшие данные дуги со знаком соответствующего бинарного отношения (***ri***). Удаление этих дуг означает, что у операндов ***x2*** и ***x3*** отсутствуют (удалены) значения.

До



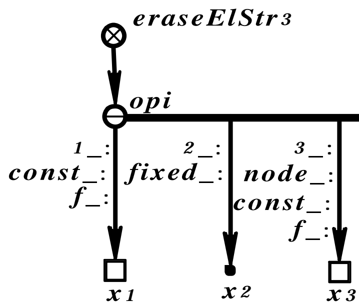
После



Библиотека БГУИР

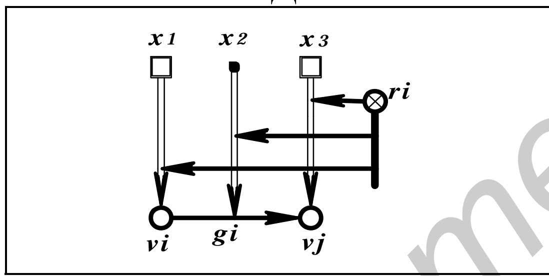
PDF created with pdfFactory Pro trial version www.pdffactory.com

Пример 5.16

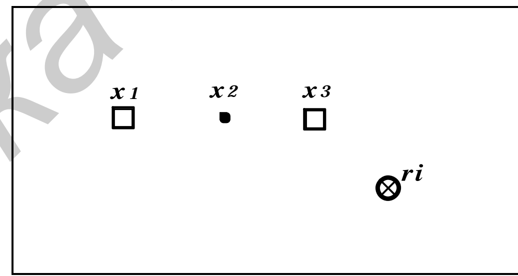


После успешного выполнения данного оператора (***opi***) будут удалены sc-элементы ***vi***, ***gi*** и ***vj***, которые являются значениями операндов ***x1***, ***x2***, ***x3*** соответственно (очевидно, что когда удаляется первый или третий элемент трехэлементной конструкции, второй элемент тоже будет удален и это можно не уточнять атрибутом ***f_***). Также будут удалены дуги, связывавшие операнды ***x1***, ***x2***, ***x3*** с их значениями, и дуги принадлежности, связывавшие вышеупомянутые дуги со знаком соответствующего бинарного отношения (***ri***). Удаление этих дуг означает, что у операндов ***x1***, ***x2***, ***x3*** отсутствуют (удалены) значения.

До



После



Операторы удаления нескольких указанных элементов sc-конструкции, состоящей из трех элементов (eraseSetStr3-операторы, операторы типа *eraseSetStr3*)

Операторы данного типа осуществляют удаление нескольких указанных элементов sc-конструкции, состоящей из трех элементов, или каких-либо элементов, входящих в состав этой конструкции.

Запись scp-операторов типа ***eraseSetStr3*** на языке SCs имеет следующий вид:

⟨***запись eraseSetStr3 –оператора***⟩
::=
eraseSetStr3 ;
⟨***идентификатор eraseSetStr3-оператора***⟩ ;
(* ***fixed_*** : |

/* данный атрибут уточняет, известно (вычислено) ли значение

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

операнда данного scp-оператора */

[* (* ***const_ :*** | ***var_ :*** | ***meta_ :*** *) *] *)

/* данные атрибуты уточняют тип удаляемого (искомого) sc-узла по признаку "константа – переменная – метапеременная" */

[* ***f_ :*** *]

/* данный атрибут означает, что найденный sc-узел должен быть удален */

1_ : □ ***операнд x1*** □ ,

/* операндом является программная переменная, или, если указан атрибут ***fixed_***, но не указан атрибут ***f_***, то операндом может быть и константа программы */

(* ***fixed_ :*** |

/* данный атрибут уточняет, известно (вычислено) ли значение операнда данного scp-оператора */

[* (* ***pos_ :*** | ***neg_ :*** | ***fuz_ :*** *) *]

/* данные атрибуты уточняют тип удаляемой (искомой) sc-дуги по признаку "позитивная дуга – негативная дуга – нечеткая дуга" */

[* (* ***const_ :*** | ***var_ :*** | ***meta_ :*** *) *] *)

/* данные атрибуты уточняют тип удаляемой (искомой) sc-дуги по признаку "константа – переменная – метапеременная" */

[* ***f_ :*** *]

/* данный атрибут означает, что найденная sc-дуга должна быть удалена */

2_ : □ ***операнд x2*** □ ,

/* операндом является программная переменная, или, если указан атрибут

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

	fixed_, но не указан атрибут ***f_***, то операндом может быть и константа программы */
(* ***fixed_ :*** |	/* данный атрибут уточняет, известно (вычислено) ли значение операнда данного scp-оператора */
(* [* ***node_ :*** | ***elem_ :*** | ***arc_ :*** *]	/* данные атрибуты уточняют тип удаляемого (искомого) sc-элемента по признаку "узел – элемент неопределенного типа – дуга" */
[* (* ***pos_ :*** | ***neg_ :*** | ***fuz_ :*** *) *] . *)	/* данные атрибуты уточняют тип удаляемой (искомой) sc-дуги по признаку "позитивная дуга – негативная дуга – нечеткая дуга" */
[* (* ***const_ :*** | ***var_ :*** | ***meta_ :*** *) *] *)	/* данные атрибуты уточняют тип удаляемого (искомого) sc-элемента по признаку "константа – переменная – метапеременная" */
[* ***f_ :*** *]	/* данный атрибут означает, что найденный sc-элемент должен быть удален */
3_ : □ ***операнд x3*** □ **,**	/* операндом является программная переменная, или, если указан атрибут ***fixed_***, но не указан атрибут ***f_***, то операндом может быть и константа программы */
[* ***set1_ :*** □ ***операнд xs1*** □ **,** *]	/* операнд, значением которого является множество "первых" элементов sc-конструкции */
[* ***set2_ :*** □ ***операнд xs2*** □ **,** *]	/* операнд, значением

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

	которого является множество “вторых” элементов sc-конструкции */
[* ***set3_ :*** □ ***операнд xs3*** □ **,** *]	/* операнд, значением которого является множество “третьих” элементов sc-конструкции */
goto_ : □ ***идентификатор scp-оператора*** □ **;**	/* здесь указывается идентификатор того scp-оператора, которому передается управление после реализации данного scp-оператора */

Результаты выполнения операторов типа ***eraseSetStr3***:

77. Если первый операнд не помечен атрибутом fixed_, то будет искаться sc-узел. Тип искомого sc-узла уточняется следующими атрибутами:
   - const_ – ищется константный sc-узел;
   - var_ – ищется переменный sc-узел;
   - meta_ – ищется метапеременный sc-узел.
78. Если искомый sc-узел найден и указан атрибут f_, то этот sc-узел удаляется.
79. Если первый операнд помечен атрибутом fixed_, а также атрибутом f_, то значение операнда удаляется.
80. Если второй операнд не помечен атрибутом fixed_, то будет искаться sc-дуга. Тип искомой sc-дуги уточняется следующими атрибутами:
   - pos_ – ищется позитивная sc-дуга;
   - neg_ – ищется негативная sc-дуга;
   - fuz_ – ищется нечеткая sc-дуга;
   - const_, var_, meta_ – ищется соответственно константная, переменная или метапеременная sc-дуга.
81. Если искомая sc-дуга найдена и указан атрибут f_, то эта sc-дуга удаляется.
82. Если второй операнд помечен атрибутом fixed_, а также атрибутом f_, то значение операнда удаляется.
83. Если третий элемент не помечен атрибутом fixed_, то будет искаться sc-элемент, тип которого уточняется следующими атрибутами:
   - node_ – ищется sc-узел;
   - elem_ – ищется sc-элемент неопределенного типа;

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

- arc_ – ищется sc-дуга. В этом случае для уточнения типа могут дополнительно использоваться атрибуты pos_, neg_, fuz_ (семантику смотри выше);
- const_, var_, meta_ – ищется соответственно константный, переменный или метапеременный sc-элемент.

84. Если искомый sc-элемент найден и указан атрибут f_, то этот sc-элемент удаляется.
85. Если третий операнд помечен атрибутом fixed_, а также атрибутом f_, то значение операнда удаляется.

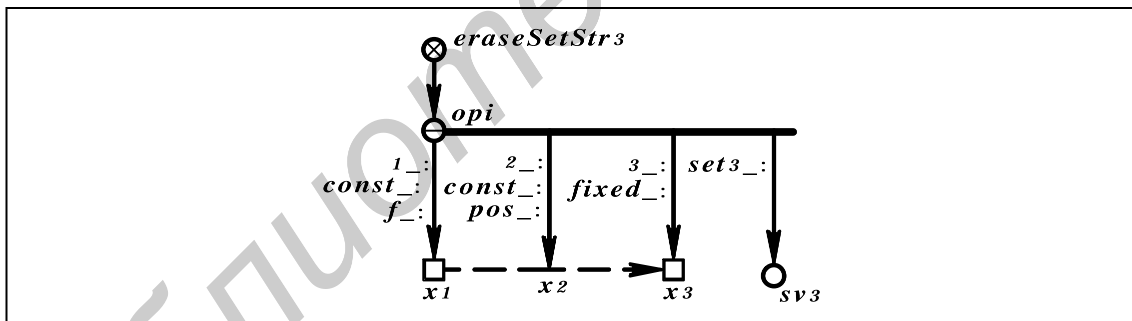
Примечание 1. Последние три операнда данного оператора (***xs1***, ***xs2*** и ***xs3***) задают множества, соответственно, “первых”, “вторых” и “третьих” элементов sc-конструкций. Эти множества определяют область поиска sc-конструкций. В найденных sc-конструкциях удаляются элементы, у операндов которых есть атрибут ***f_***.

Примечание 2. Операнды ***xs1***, ***xs2*** и ***xs3*** могут быть scp-переменными и scp-константами. В описании оператора обязательно должен присутствовать хотя бы один из операндов ***xs1***, ***xs2*** и ***xs3***.

Примечание 3. Возможно, что ни один операнд не помечен атрибутом ***fixed_***. Операнды ***xs1***, ***xs2*** и ***xs3*** могут совпадать.

Модификации scp-операторов типа ***eraseSetStr3*** задаются комбинацией атрибутов, которыми помечен операнд. Приведем некоторые модификации eraseSetStr3-оператора.

Пример 5.17

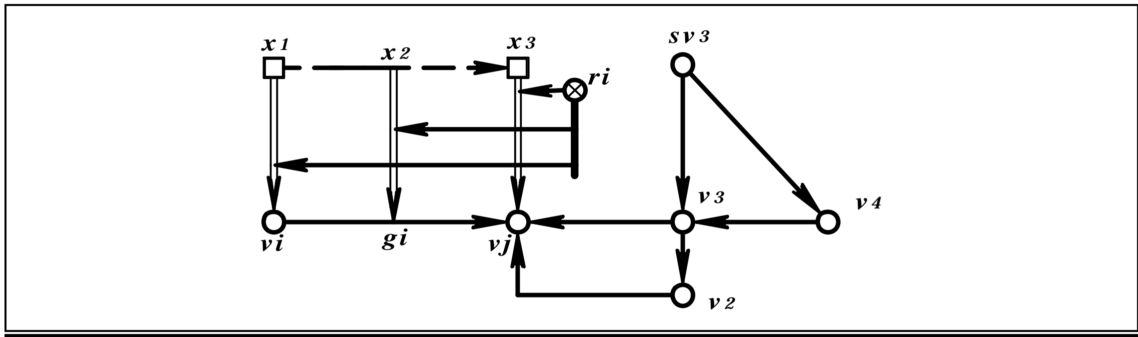


После успешного выполнения данного оператора (***opi***) будет удален sc-узел ***v2***, являющийся первым элементом трехэлементной конструкции, которая соответствует заданному шаблону и чей третий элемент ***vj*** входит в множество ***sv3***(состоит из ***vj*** и ***v3***). Также будут удалены дуги, входящие и выходящие из удаленного узла.

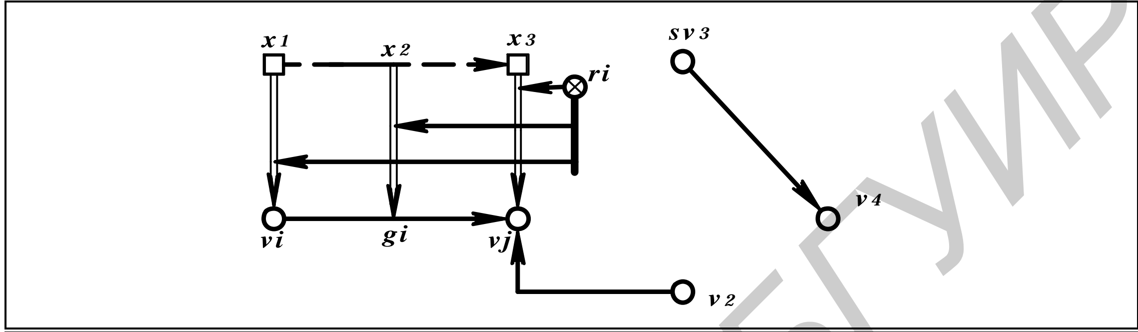
До

Библиотека БГУИР

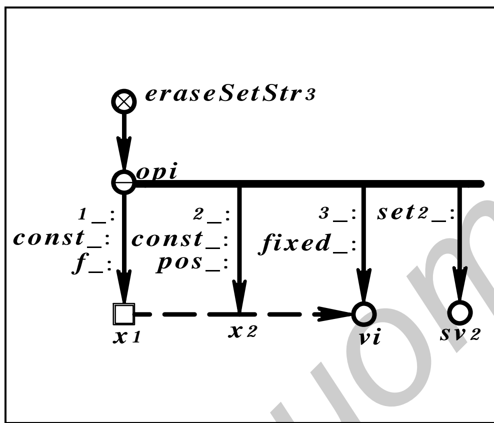
PDF created with pdfFactory Pro trial version www.pdffactory.com



После



Пример 5.18

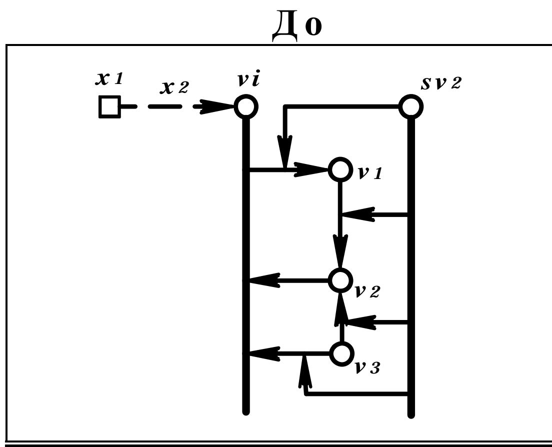


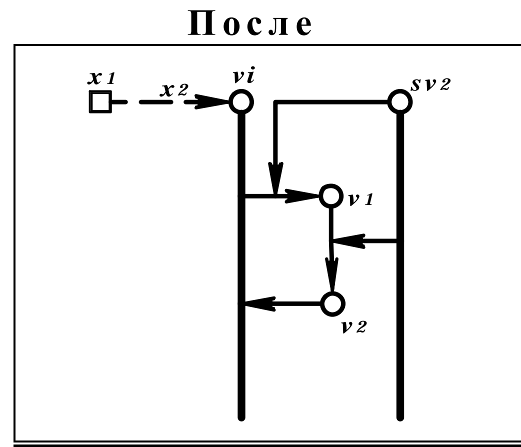
После успешного выполнения данного оператора (***opi***) будет удален sc-узел ***v3***, являющийся первым элементом трехэлементной конструкции, которая соответствует заданному шаблону и чей второй элемент входит в множество ***sv2***. Также будут удалены дуги, входящие и выходящие из удаленного узла.

Библиотека БГУИР

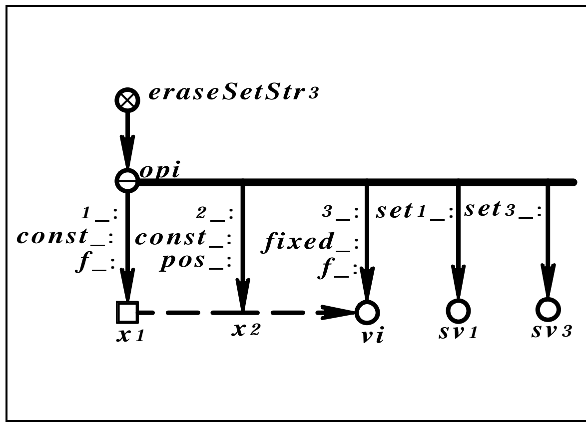
PDF created with pdfFactory Pro trial version www.pdffactory.com

До **После**



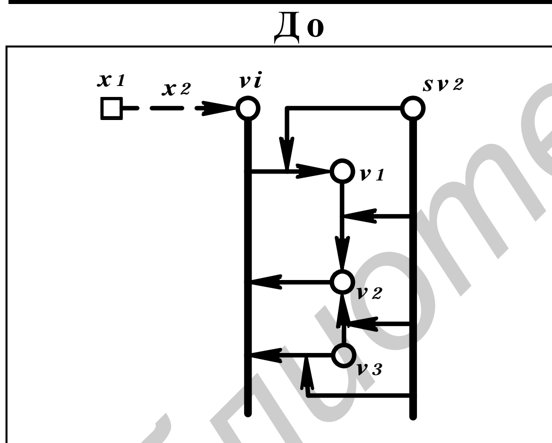


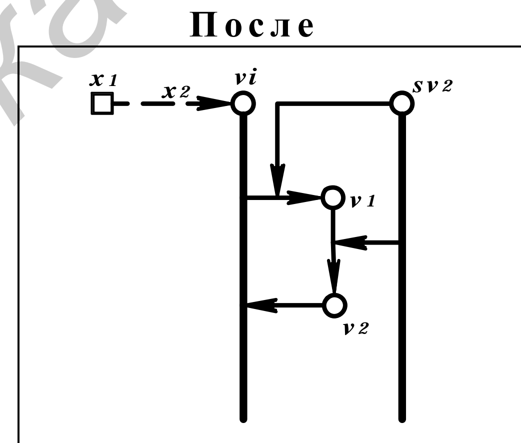
Пример 5.19



После выполнения данного оператора (***opi***) изменений в памяти не произойдет, так как во множествах ***sv1*** и ***sv3*** не содержится трехэлементной sc-конструкции, удовлетворяющей заданному шаблону (***vi*** не является третьим элементом ни одной из конструкций).

До **После**





5.2.4. Операторы ассоциативного поиска конструкций языка SC

Семейство scp-операторов ассоциативного поиска sc-конструкций (такие операторы будем также называть search-операторами) разбивается на следующие типы scp-операторов:

- операторы ассоциативного поиска произвольных элементов sc-конструкции, состоящей из трёх элементов (searchElStr3-операторы);
- операторы ассоциативного поиска произвольных элементов sc-конструкций, удовлетворяющих заданным условиям и состоящих из трех элементов, и формирования из них множества (searchSetStr3-операторы);
- операторы ассоциативного поиска произвольных элементов sc-конструкций, удовлетворяющих заданным условиям и состоящих из

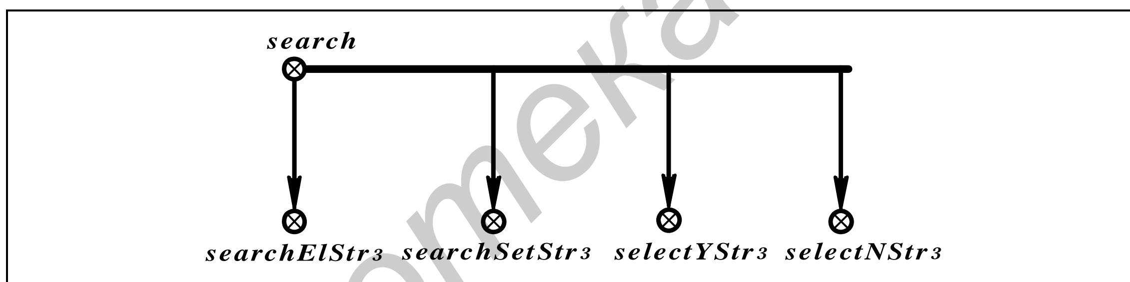
Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

трёх элементов, и формирования из них множества, элементами которого остаются только sc-элементы, удовлетворяющие заданным условиям (selectYStr3-операторы);

- операторы ассоциативного описка произвольных элементов sc-конструкций, удовлетворяющих заданным условиям и состоящих из трёх элементов, и формирования из них множества, элементами которого остаются только sc-элементы, не удовлетворяющие заданным условиям (selectNStr3-операторы).

Множество searchElStr3-операторов в языке SCP будем обозначать ключевым узлом, которому поставим в соответствие идентификатор ***“searchElStr3”***. Множество searchSetStr3-операторов в языке SCP будем обозначать ключевым узлом, которому поставим в соответствие идентификатор ***“searchSetStr3”***. Множество selectYStr3-операторов в языке SCP будем обозначать ключевым узлом, которому поставим в соответствие идентификатор ***“selectYStr3”***. Множество selectNStr3-операторов в языке SCP будем обозначать ключевым узлом, которому поставим в соответствие идентификатор ***“selectNStr3”***. Семейство типов search-операторов в языке SCP обозначается ключевым узлом с идентификатором ***“search”***. Соотношение между указанными ключевыми узлами задается следующей sc-конструкцией:



Перейдем к более подробному рассмотрению указанных выше типов scp-операторов.

Операторы ассоциативного поиска произвольных элементов sc-конструкции, состоящей из трёх элементов (searchElStr3-операторы, операторы типа *searchElStr3*)

Операторы данного типа осуществляют ассоциативный поиск произвольных элементов sc-конструкции, состоящей из трёх элементов.

Запись scp-операторов типа ***searchElStr3*** на языке SCs имеет следующий вид:

 запись searchElStr3 – оператора ::=

searchElStr3 ;

 идентификатор

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

searchElStr3-оператора □ ;

(* ***fixed_ :*** | ***assign_ :***

/* данные атрибуты уточняют, известно (вычислено) ли значение операнда или необходимо найти sc-элемент и установить значение операнда данного оператора */

[* (* ***const_ :*** | ***var_ :*** | ***meta_ :*** *) *] *)

/* данные атрибуты уточняют тип искомого sc-элемента по признаку "константа – переменная – метапеременная" */

1_ : □ ***операнд x1*** □ **,**

/* операндом является scp-переменная, если указан атрибут ***assign_***, значением которой становится найденный sc-элемент. Если указан атрибут ***fixed_***, то операндом может быть и scp-переменная, и scp-константа */

(* ***fixed_ :*** | ***assign_ :***

/* данные атрибуты уточняют, известно (вычислено) ли значение операнда или необходимо найти sc-элемент и установить значение операнда данного оператора */

[* (* ***pos_ :*** | ***neg_ :*** | ***fuz_ :*** *) *]

/* данные атрибуты уточняют тип искомой sc-дуги по признаку "позитивная дуга – негативная дуга – нечеткая дуга" */

[* (* ***const_ :*** | ***var_ :*** | ***meta_ :*** *) *] *)

/* данные атрибуты уточняют тип искомого sc-элемента по признаку "константа – переменная –

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

метапеременная" */

2_ : □ ***операнд x2*** □ ***,*** /* операндом является scp-переменная, если указан атрибут ***assign_***, значением которой становится найденный sc-элемент. Если указан атрибут ***fixed_***, то операндом может быть и scp-переменная и scp-константа */

(* ***fixed_ :*** | ***assign_ :*** /* данные атрибуты уточняют, известно (вычислено) ли значение операнда или необходимо найти sc-элемент и установить его значением операнда данного оператора */

(* [* ***node_ :*** | ***elem_ :*** | ***arc_ :*** *] /* данные атрибуты уточняют тип искомого sc-элемента по признаку "узел – элемент неопределенного типа – дуга" */

[* (* ***pos_ :*** | ***neg_ :*** | ***fuz_ :*** *) *] . *) /* данные атрибуты уточняют тип искомого sc-дуги по признаку "позитивная дуга – негативная дуга – нечеткая дуга" */

[* (* ***const_ :*** | ***var_ :*** | ***meta_ :*** *) *] *) /* данные атрибуты уточняют тип искомого sc-элемента по признаку "константа – переменная – метапеременная" */

3_ : □ ***операнд x3*** □ ***,*** /* операндом является scp-переменная, если указан атрибут ***assign_***, значением которой становится

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

	найденный sc-элемент. Если указан атрибут ***fixed_***, то операндом может быть и scp-переменная, и scp-константа */
then_ : □ ***идентификатор scp-оператора*** □ ;	/* здесь указывается идентификатор того scp-оператора, которому передается управление после реализации данного scp-оператора и нахождения требуемой sc-конструкции */
else_ : □ ***идентификатор scp-оператора*** □ ;	/* здесь указывается идентификатор того scp-оператора, которому передается управление после реализации данного scp-оператора и ненахождения требуемой sc-конструкции */

Результаты выполнения операторов типа ***searchElStr3***:

86. Если первый операнд помечен атрибутом assign_, то будет искаться sc-узел. Тип искомого sc-узла уточняется следующими атрибутами:
    - const_ – ищется константный sc-узел;
    - var_ – ищется переменный sc-узел;
    - meta_ – ищется метапеременный sc-узел.
87. Если второй операнд помечен атрибутом assign_, то будет искаться sc-дуга. Тип искомой sc-дуги уточняется следующими атрибутами:
    - pos_ – ищется позитивная sc-дуга;
    - neg_ – ищется негативная sc-дуга;
    - fuz_ – ищется нечеткая sc-дуга;
    - const_, var_, meta_ – ищется соответственно константная, переменная или метапеременная sc-дуга.
88. Если искомая sc-дуга найдена и указан атрибут f_, то эта sc-дуга удаляется.
89. Если третий элемент помечен атрибутом assign_, то будет искаться sc-элемент, тип которого уточняется следующими атрибутами:
    - node_ – ищется sc-узел;
    - elem_ – ищется sc-элемент неопределенного типа;

Библиотека БГУИР

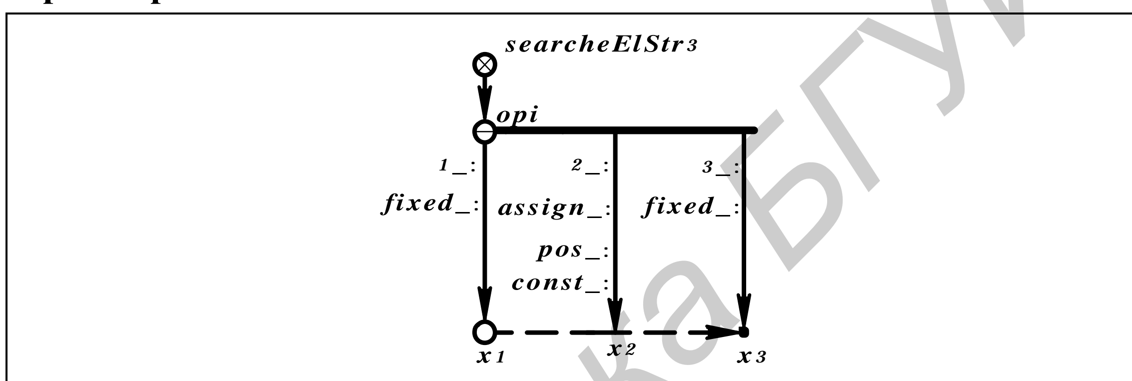
PDF created with pdfFactory Pro trial version www.pdffactory.com

- arc_ – ищется sc-дуга. В этом случае для уточнения типа могут дополнительно использоваться атрибуты pos_, neg_, fuz_ (семантику смотри выше);
- const_, var_, meta_ – ищется соответственно константный, переменный или метапеременный sc-элемент.

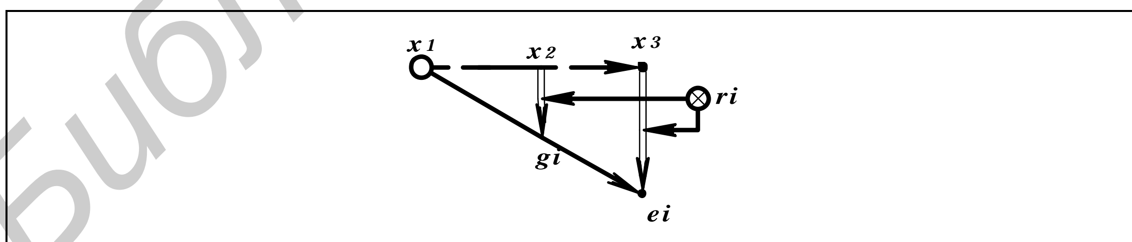
Примечание. Хотя бы один операнд оператора должен быть помечен атрибутом ***fixed_***. Если при поиске будут найдены несколько подходящих sc-конструкций, то выберется одна из них.

Модификации scp-операторов типа ***searchElStr3*** задаются комбинацией атрибутов, которыми помечен операнд. Приведем некоторые модификаций searchElStr3-оператора.

Пример 5.20



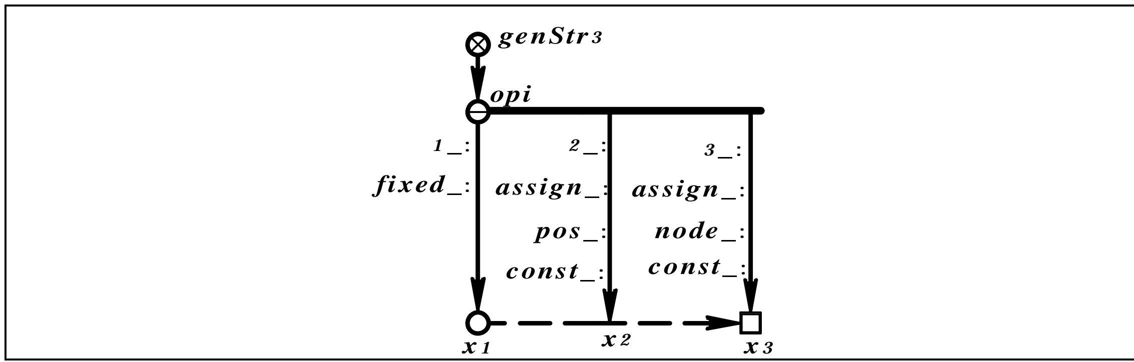
В результате успешного выполнения данного оператора (***opi***) будет найдена константная позитивная sc-дуга (***gi***), соединяющая sc-узел, являющийся значением операнда ***x1***, и sc-элемент (***ei***), являющийся значением операнда ***x3***. Также будет сгенерирована дуга, связывающая операнд ***x2*** с sc-элементом ***gi***, и дуга принадлежности, связывающая указанную выше дугу со знаком соответствующего бинарного отношения (***ri***). Генерация этой дуги означает, что у операнда ***x2*** сформировалось (вычислилось) значение:



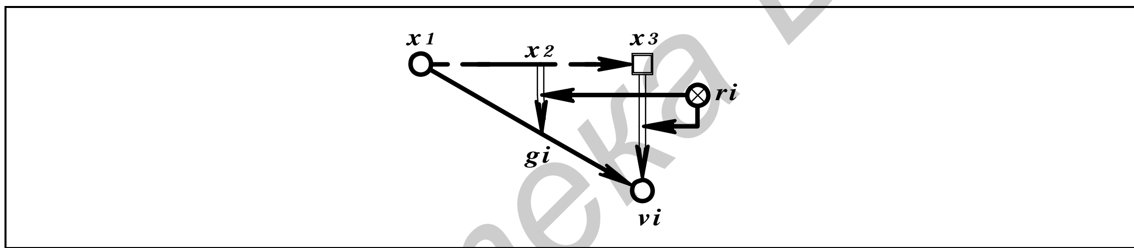
Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

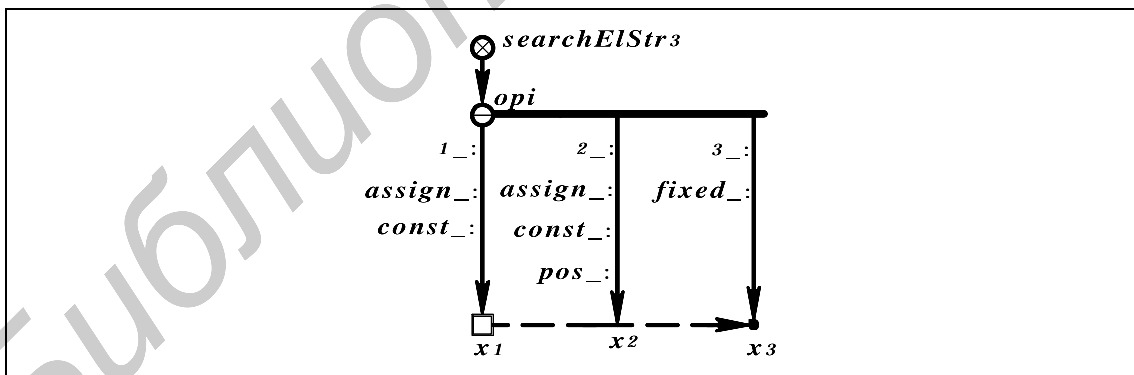
Пример 5.21



В результате успешного выполнения данного оператора (***opi***) будут найдены константный sc-узел (***vi***) и константная позитивная sc-дуга (***gi***), соединяющая его с sc-узлом, являющимся значением операнда ***x1***. Также будут сгенерированы дуга, связывающая операнд ***x2*** с sc-элементом ***gi***, и дуга, связывающая операнд ***x3*** с sc-элементом ***vi***, а кроме того, дуги принадлежности, связывающие указанные выше дуги со знаком соответствующего бинарного отношения (***ri***). Генерация этих дуг означает, что у операндов ***x2*** и ***x3*** сформировалось (вычислилось) значение:



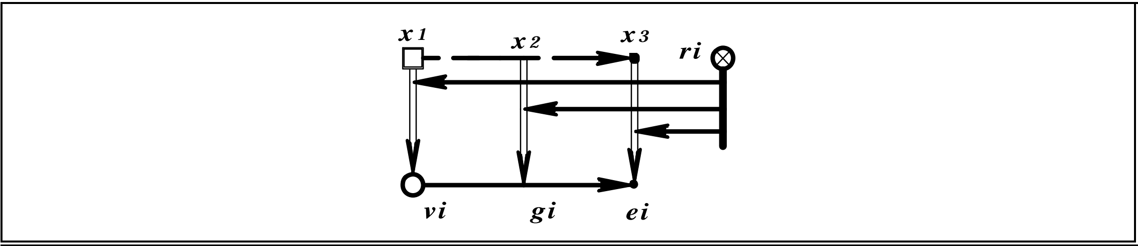
Пример 5.22



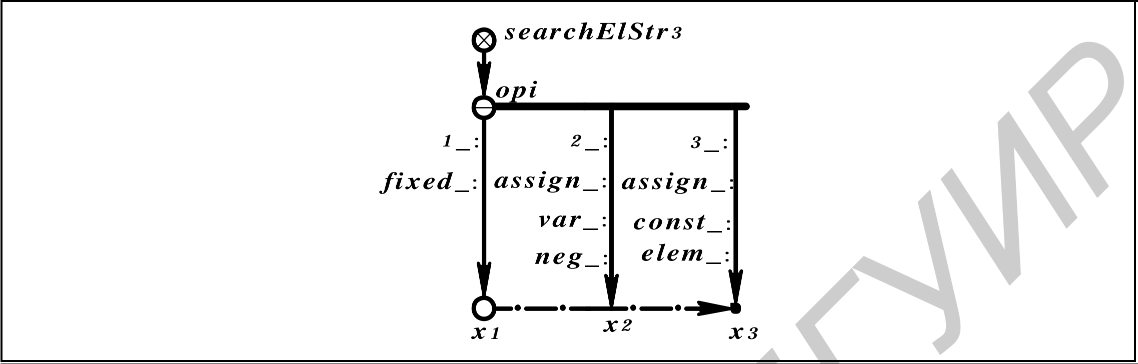
В результате успешного выполнения данного оператора (***opi***) будут найдены константный sc-узел (***vi***) и константная позитивная sc-дуга (***gi***), соединяющая его с sc-элементом (***ei***), являющимся значением операнда ***x3***. Также будут сгенерированы дуга, связывающая операнд ***x2*** с sc-элементом ***gi***, и дуга, связывающая операнд ***x1*** с sc-элементом ***vi***, а также дуги принадлежности, связывающие указанные выше дуги со знаком соответствующего бинарного отношения (***ri***). Генерация этих дуг означает, что у операндов ***x1*** и ***x2*** сформировалось (вычислилось) значение:

Библиотека БГУИР

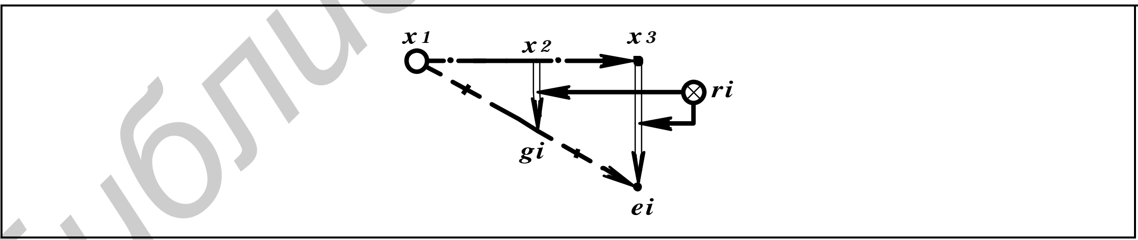
PDF created with pdfFactory Pro trial version www.pdffactory.com



Пример 5.23



В результате успешного выполнения данного оператора (***opi***) будут найдены константный sc-элемент неопределенного типа (***ei***) и переменная негативная sc-дуга (***gi***), соединяющая его с sc-узлом, являющимся значением операнда ***x1***. Также будут сгенерированы дуга, связывающая операнд ***x2*** с sc-элементом ***gi***, и дуга, связывающая операнд ***x3*** с sc-элементом ***ei***, а также дуги принадлежности, связывающие указанные выше дуги со знаком соответствующего бинарного отношения (***ri***). Генерация этих дуг означает, что у операндов ***x2*** и ***x3*** сформировалось (вычислилось) значение:



Операторы ассоциативного поиска произвольных элементов sc-конструкций, удовлетворяющих заданным условиям и состоящих из трех элементов, и формирования из них множества (searchSetStr3-операторы, операторы типа *searchSetStr3*)

Операторы данного типа осуществляют ассоциативный поиск произвольных элементов sc-конструкций, удовлетворяющих заданным условиям и состоящих из трех элементов, и формирования из них множества.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Запись scp-операторов типа ***searchSetStr3*** на языке SCs имеет следующий вид:

□ ***запись searchSetStr3 –оператора*** □

::=

searchSetStr3 ;

□ ***идентификатор searchSetStr3-оператора*** □ ;

(* ***fixed_*** **:** | /* данный атрибут уточняет, известно (вычислено) ли значение операнда */

[* (* ***const_*** **:** | ***var_*** **:** | ***meta_*** **:** *) *] *) /* данные атрибуты уточняют тип искомого sc-узла по признаку "константа – переменная – метапеременная" */

1_ **:** □ ***операнд x1*** □ , /* операндом является программная переменная, или, если указан атрибут ***fixed_***, то операндом может быть и константа программы */

(* ***fixed_*** **:** | /* данный атрибут уточняет, известно (вычислено) ли значение операнда */

[* (* ***pos_*** **:** | ***neg_*** **:** | ***fuz_*** **:** *) *] /* данные атрибуты уточняют тип искомой sc-дуги по признаку "позитивная дуга – негативная дуга – нечеткая дуга" */

[* (* ***const_*** **:** | ***var_*** **:** | ***meta_*** **:** *) *] *) /* данные атрибуты уточняют тип искомого sc-элемента по признаку "константа – переменная – метапеременная" */

2_ **:** □ ***операнд x2*** □ , /* операндом является программная переменная, или, если указан атрибут ***fixed_***, то операндом может быть и константа программы */

(* ***fixed_*** **:** | /* данный атрибут уточняет, известно

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

	(вычислено) ли значение операнда */
(* [* ***node_ :*** | ***elem_ :*** | ***arc_ :*** *]	/* данные атрибуты уточняют тип искомого sc-элемента по признаку "узел – элемент неопределенного типа – дуга" */
[* (* ***pos_ :*** | ***neg_ :*** | ***fuz_ :*** *) *] . *)	/* данные атрибуты уточняют тип искомой sc-дуги по признаку "позитивная дуга – негативная дуга – нечеткая дуга" */
[* (* ***const_ :*** | ***var_ :*** | ***meta_ :*** *) *] *)	/* данные атрибуты уточняют тип искомого sc-элемента по признаку "константа – переменная – метапеременная" */
3_ : □ ***операнд x3*** □ **,**	/* операндом является программная переменная, или, если указан атрибут ***fixed_***, то операндом может быть и константа программы */
(* ***fixed_ :*** | ***assign_ :*** *)	/* данные атрибуты уточняют, известно (вычислено) ли значение операнда или необходимо сгенерировать sc-элемент и установить значение операнда данного оператора */
set1_ : □ ***операнд xs1*** □ **,**	/* операндом является программная переменная, или, если указан атрибут ***fixed_***, то операндом может быть и константа программы. Значением данного операнда становится множество “первых” элементов найденных sc-конструкций */

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

(* ***fixed_* : | *assign_* : *) /* данные атрибуты уточняют, известно (вычислено) ли значение операнда или необходимо сгенерировать sc-элемент и установить его значением операнда данного оператора */

set2_* :** □ ***операнд xs2 □ **,** /* операндом является программная переменная, или, если указан атрибут ***fixed_***, то операндом может быть и константа программы. Значением данного операнда становится множество “вторых” элементов найденных sc-конструкций */

(* ***fixed_* : | *assign_* : *) /* данные атрибуты уточняют, известно (вычислено) ли значение операнда или необходимо сгенерировать sc-элемент и установить значение операнда данного оператора */

set3_* :** □ ***операнд xs3 □ **,** /* операндом является программная переменная, или, если указан атрибут ***fixed_***, то операндом может быть и константа программы. Значением данного операнда становится множество “третьих” элементов найденных sc-конструкций */

then_* :** □ ***идентификатор scp-оператора □ **;** /* здесь указывается идентификатор того scp-оператора, которому передается управление после реализации данного scp-оператора и нахождения требуемой sc-конструкции */

else_* :** □ ***идентификатор /* здесь указывается

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

scp-оператора □ ; идентификатор того scp-оператора, которому передается управление после реализации данного scp-оператора и ненахождения требуемой sc-конструкции */

Результаты выполнения операторов типа ***searchSetStr3***:

90. Если первый операнд не помечен атрибутом fixed_, то будет искаться sc-узел. Тип искомого sc-узла уточняется следующими атрибутами:
   - const_ – ищется константный sc-узел;
   - var_ – ищется переменный sc-узел;
   - meta_ – ищется метапеременный sc-узел.
91. Если второй операнд не помечен атрибутом fixed_, то будет искаться sc-дуга. Тип искомой sc-дуги уточняется следующими атрибутами:
   - pos_ – ищется позитивная sc-дуга;
   - neg_ – ищется негативная sc-дуга;
   - fuz_ – ищется нечеткая sc-дуга;
   - const_, var_, meta_ – ищется соответственно константная, переменная или метапеременная sc-дуга.
92. Если третий элемент не помечен атрибутом fixed_, то будет искаться sc-элемент, тип которого уточняется следующими атрибутами:
   - node_ – ищется sc-узел;
   - elem_ – ищется sc-элемент неопределенного типа;
   - arc_ – ищется sc-дуга. В этом случае для уточнения типа могут дополнительно использоваться атрибуты pos_, neg_, fuz_ (семантику смотри выше);
   - const_, var_, meta_ – ищется соответственно константный, переменный или метапеременный sc-элемент.

Примечание 1. Хотя бы один операнд из первых трёх операндов должен быть помечен атрибутом ***fixed_***.

Примечание 2. Последние три операнда (***xs1***, ***xs2*** и ***xs3***) задают множества соответственно “первых”, “вторых” и “третьих” элементов sc-конструкций. Если указан атрибут ***assign_***, то генерируется новое множество, а если указан атрибут ***fixed_***, то множество дополняется найденными sc-конструкциями.

Примечание 3. Операнды ***xs1***, ***xs2*** и ***xs3*** могут быть scp-переменными и scp-константами. Должен присутствовать хотя бы один из операндов ***xs1***, ***xs2*** и ***xs3***.

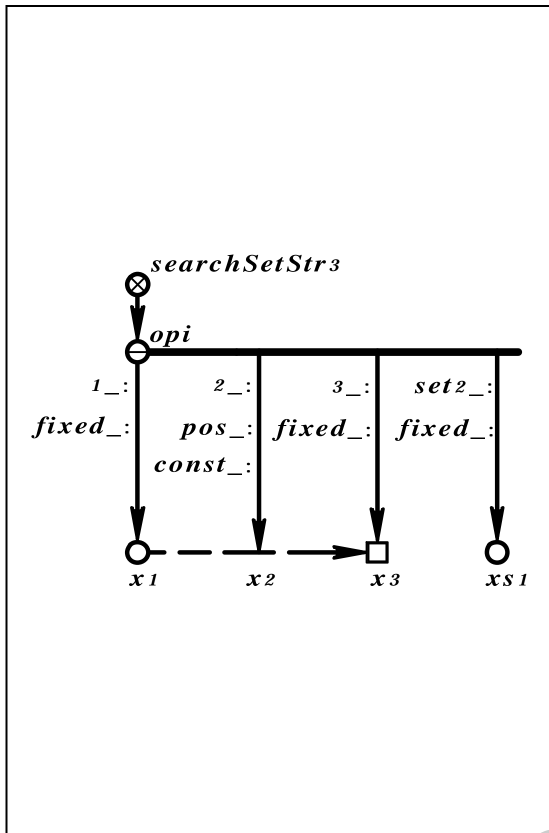
Примечание 4. Операнды ***xs1***, ***xs2*** и ***xs3*** могут совпадать.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

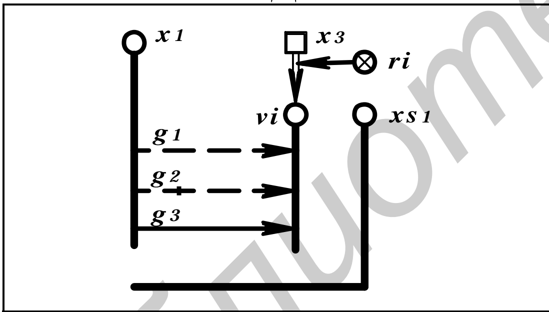
Модификации scp-операторов типа ***searchSetStr3*** задаются комбинацией атрибутов, которыми помечен операнд. Приведем некоторые модификации searchSetStr3-оператора.

Пример 5.24

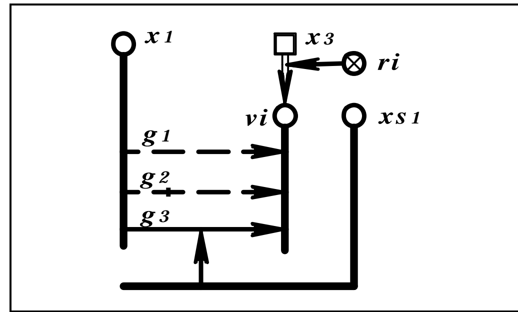


В результате успешного выполнения данного оператора (***opi***) будет найдена константная позитивная sc-дуга (***v1***) и константная позитивная sc-дуга (***gi***), соединяющая его с sc-элементом (***ei***), являющимся значением операнда ***x3***. Также будут сгенерированы дуга, связывающая операнд ***x2*** с sc-элементом ***gi***, и дуга, связывающая операнд ***x1*** с sc-элементом ***vi***, а также дуги принадлежности, связывающие указанные выше дуги со знаком соответствующего бинарного отношения (***ri***). Генерация этих дуг означает, что у операндов ***x1*** и ***x2*** сформировалось (вычислилось) значение:

До



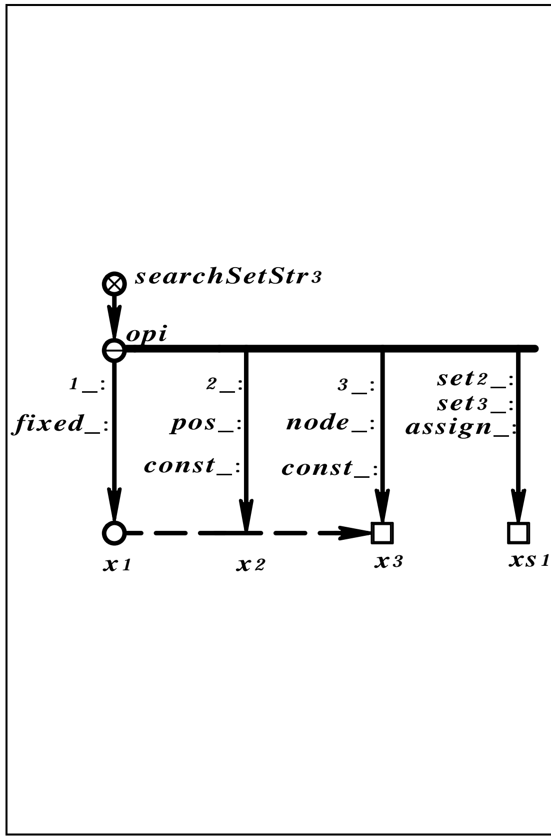
После



Библиотека БГУИР

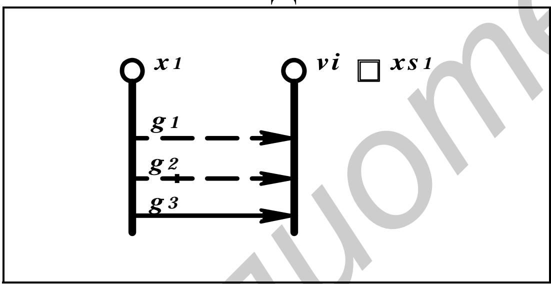
PDF created with pdfFactory Pro trial version www.pdffactory.com

Пример 5.25

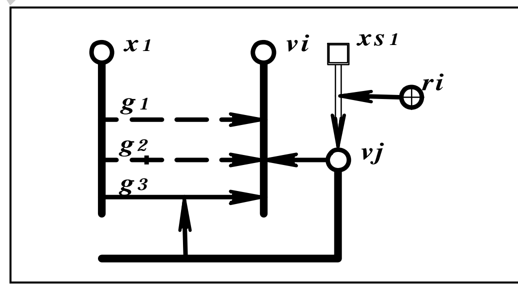


В результате успешного выполнения данного оператора (***opi***) будут найдены константный sc-узел (***vi***) и константная позитивная sc-дуга (***gi***), соединяющая его с sc-элементом (***ei***), являющимся значением операнда ***x3***. Также будут сгенерированы дуга, связывающая операнд ***x2*** с sc-элементом ***gi***, и дуга, связывающая операнд ***x1*** с sc-элементом ***vi***, а также дуги принадлежности, связывающие указанные выше дуги со знаком соответствующего бинарного отношения (***ri***). Генерация этих дуг означает, что у операндов ***x1*** и ***x2*** сформировалось (вычислилось) значение:

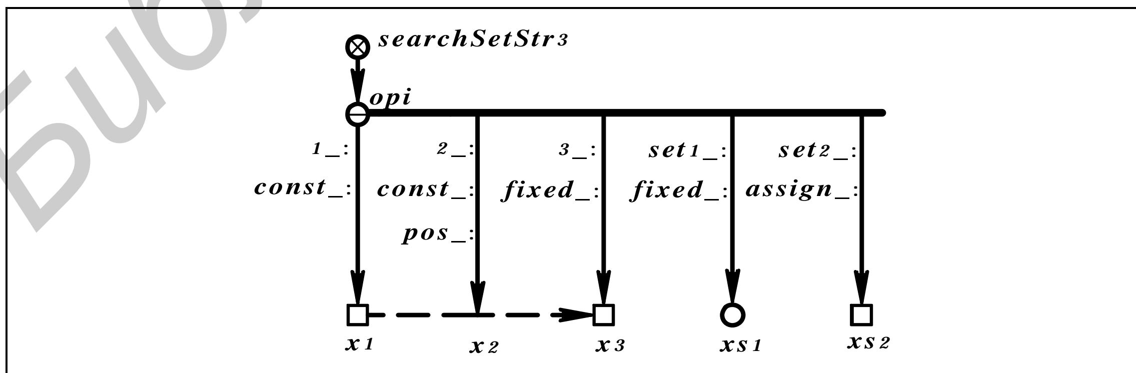
До



После



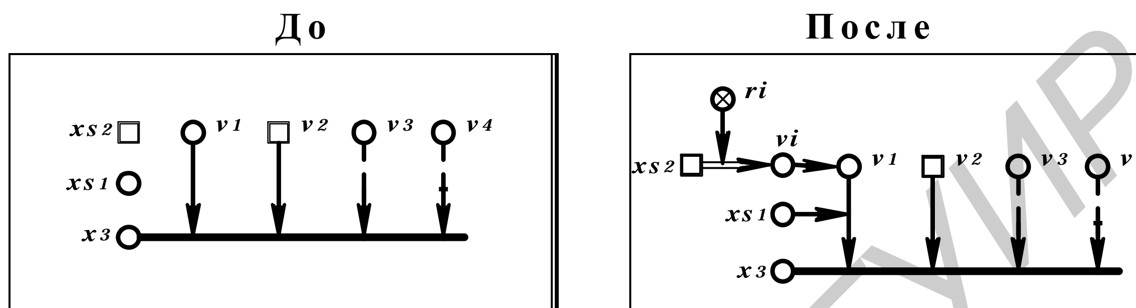
Пример 5.26



Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

В результате успешного выполнения данного оператора (***opi***) будут найдены константный sc-узел (***vi***) и константная позитивная sc-дуга (***gi***), соединяющая его с sc-элементом (***ei***), являющимся значением операнда ***x3***. Также будут сгенерированы дуга, связывающая операнд ***x2*** с sc-элементом ***gi***, и дуга, связывающая операнд ***x1*** с sc-элементом ***vi***, а также дуги принадлежности, связывающие указанные выше дуги со знаком соответствующего бинарного отношения (***ri***). Генерация этих дуг означает, что у операндов ***x1*** и ***x2*** сформировалось (вычислилось) значение:



Операторы ассоциативного поиска произвольных элементов, удовлетворяющих заданным условиям, и формирования из них множества (selectYStr3-операторы, операторы типа *selectYStr3*)

Операторы данного типа осуществляют ассоциативный поиск произвольных элементов sc-конструкций, удовлетворяющих заданным условиям и состоящих из трех элементов, и формирования из них множества, элементами которого остаются только sc-элементы, удовлетворяющие заданным условиям.

Запись scp-операторов типа ***selectYStr3*** на языке SCs имеет следующий вид:

 запись selectYStr3 –оператора ::=

selectYStr3 ;

 идентификатор

selectYStr3-оператора ;

(* ***fixed_*** : |	/* данный атрибут уточняет, известно (вычислено) ли значение операнда */
[* (* ***const_*** : | ***var_*** : | ***meta_*** : *) *] *)	/* данные атрибуты уточняют тип искомого sc-узла по признаку "константа – переменная – метапеременная" */
1_ : ***операнд x1*** ,	/* операндом является программная переменная,

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

	или, если указан атрибут ***fixed_***, то операндом может быть и константа программы */
(* ***fixed_ :*** |	/* данный атрибут уточняет, известно (вычислено) ли значение операнда */
[* (* ***pos_ :*** | ***neg_ :*** | ***fuz_ :*** *) *]	/* данные атрибуты уточняют тип искомой sc-дуги по признаку «позитивная дуга – негативная дуга – нечеткая дуга» */
[* (* ***const_ :*** | ***var_ :*** | ***meta_ :*** *) *] *)	/* данные атрибуты уточняют тип искомого sc-элемента по признаку «константа – переменная – метапеременная» */
2_ : □ ***операнд x2*** □ **,**	/* операндом является программная переменная, или, если указан атрибут ***fixed_***, то операндом может быть и константа программы */
(* ***fixed_*** : |	/* данный атрибут уточняет, известно (вычислено) ли значение операнда */
(* [* ***node_ :*** | ***elem_ :*** | ***arc_ :*** *]	/* данные атрибуты уточняют тип искомого sc-элемента по признаку «узел – элемент неопределенного типа – дуга» */
[* (* ***pos_ :*** | ***neg_ :*** | ***fuz_ :*** *) *] . *)	/* данные атрибуты уточняют тип искомой sc-дуги по признаку «позитивная дуга – негативная дуга – нечеткая дуга» */

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

[* (* ***const_*** : | ***var_*** : | ***meta_*** : *) *] *)	/* данные атрибуты уточняют тип искомого sc-элемента по признаку «константа – переменная – метапеременная» */
3_ : □ ***операнд x3*** □ ,	/* операндом является программная переменная, или, если указан атрибут ***fixed_***, то операндом может быть и константа программы */
[* ***set1_*** : □ ***операнд xs1*** □ , *]	/* операнд, значением которого является множество “первых” элементов sc-конструкции */
[* ***set2_*** : □ ***операнд xs2*** □ , *]	/* операнд, значением которого является множество “вторых” элементов sc-конструкции */
[* ***set3_*** : □ ***операнд xs3*** □ , *]	/* операнд, значением которого является множество “третьих” элементов sc-конструкции */
then_ : □ ***идентификатор scp-оператора*** □ ;	/* здесь указывается идентификатор того scp-оператора, которому передается управление после реализации данного scp-оператора и нахождения требуемой sc-конструкции */
else_ : □ ***идентификатор scp-оператора*** □ ;	/* здесь указывается идентификатор того scp-оператора, которому передается управление после реализации данного scp-оператора и ненахождения требуемой sc-конструкции */

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Результаты выполнения операторов типа ***selectYStr3***:

93. Если первый операнд не помечен атрибутом fixed_, то будет искаться sc-узел. Тип искомого sc-узла уточняется следующими атрибутами:
   - const_ – ищется константный sc-узел;
   - var_ – ищется переменный sc-узел;
   - meta_ – ищется метапеременный sc-узел.
94. Если второй операнд не помечен атрибутом fixed_, то будет искаться sc-дуга. Тип искомой sc-дуги уточняется следующими атрибутами:
   - pos_ – ищется позитивная sc-дуга;
   - neg_ – ищется негативная sc-дуга;
   - fuz_ – ищется нечеткая sc-дуга;
   - const_, var_, meta_ – ищется соответственно константная, переменная или метапеременная sc-дуга.
95. Если третий элемент не помечен атрибутом fixed_, то будет искаться sc-элемент, тип которого уточняется следующими атрибутами:
   - node_ – ищется sc-узел;
   - elem_ – ищется sc-элемент неопределенного типа;
   - arc_ – ищется sc-дуга. В этом случае для уточнения типа могут быть дополнительно использоваться атрибуты pos_, neg_, fuz_ (семантику смотри выше);
   - const_, var_, meta_ – ищется соответственно константный, переменный или метапеременный sc-элемент.

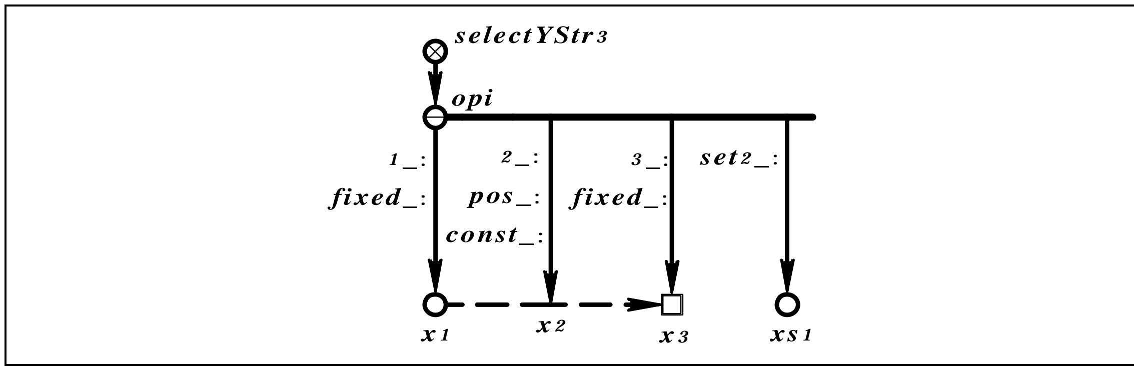
Примечание. Хотя бы один операнд из первых трёх операндов должен быть помечен атрибутом ***fixed_***. Последние три операнда (***xs1***, ***xs2*** и ***xs3***) задают множества соответственно “первых”, “вторых” и “третьих” элементов sc-конструкций. Значение этих операндов должно быть вычислено. В этих множествах остаются только те sc-конструкции, которые удовлетворяют заданному условию. Условие задается атрибутами на операндах ***x1***, ***x2*** и ***x3*** (см. выше). Операнды ***xs1***, ***xs2*** и ***xs3*** могут быть scp-переменными и scp-константами. Должен присутствовать хотя бы один из операндов ***xs1***, ***xs2*** и ***xs3***. Операнды ***xs1***, ***xs2*** и ***xs3*** могут совпадать.

Модификации scp-операторов типа ***selectYStr3*** задаются комбинацией атрибутов, которыми помечен операнд. Приведем некоторые модификации selectYStr3-оператора.

Библиотека БГУИР

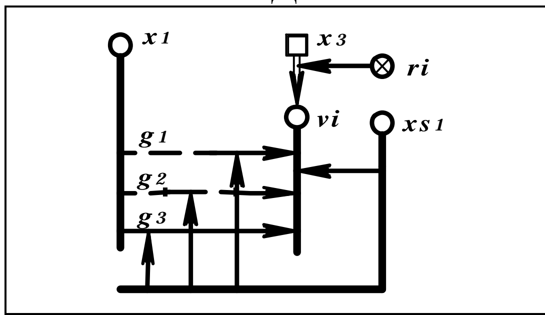
PDF created with pdfFactory Pro trial version www.pdffactory.com

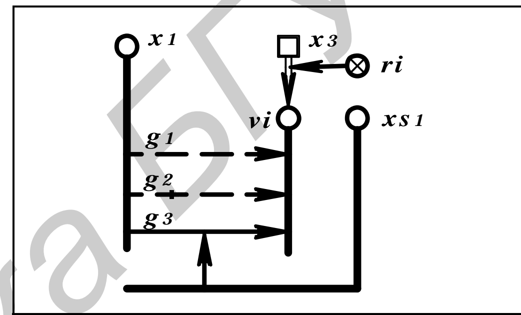
Пример 5.27



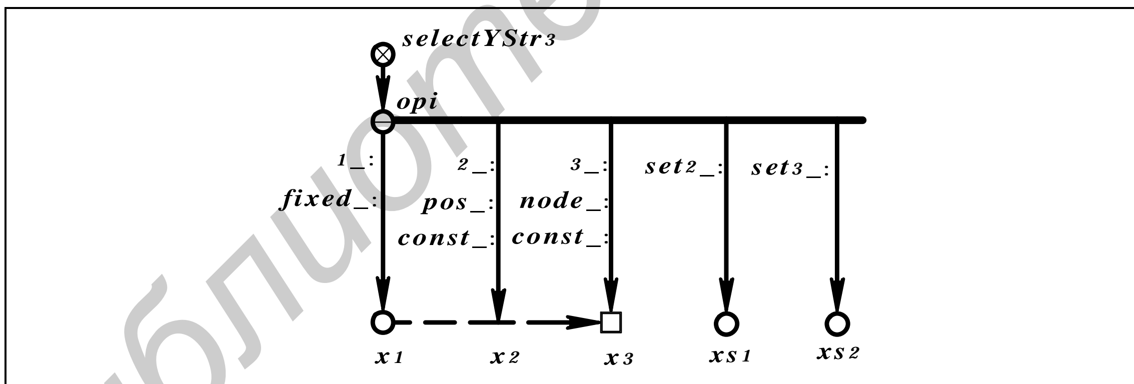
В результате успешного выполнения данного оператора (***opi***) во множестве вторых элементов ***xs1*** будет найдена константная позитивная sc-дуга (***v3***), значение операнда ***x1*** и значение операнда ***x3***– sc-узел (***vi***). Все элементы, кроме найденной sc-дуги (***v3***), будут удалены из множества ***xs1***.

До **После**





Пример 5.28

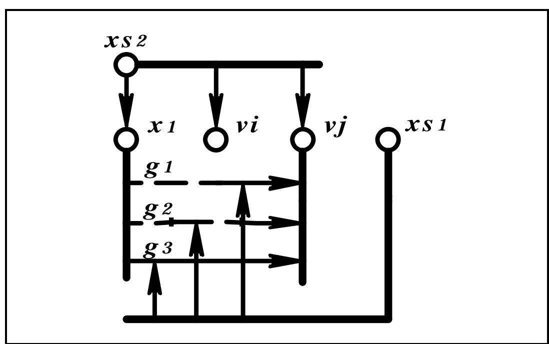


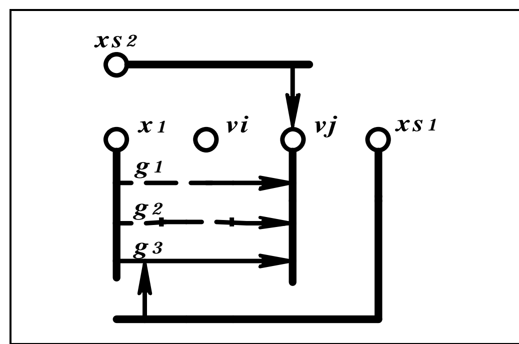
В результате успешного выполнения данного оператора (***opi***) будут найдены константный sc-узел (***vj***) и константная позитивная sc-дуга (***g3***), соединяющая его со значением операнда ***x1***, причем найденный sc-узел (***vj***) принадлежит множеству третьих элементов ***xs1***, а найденная sc-дуга (***g3***) – множеству вторых элементов ***xs2***. Все остальные элементы удаляются из этих множеств.

До **После**

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com





Операторы ассоциативного поиска произвольных элементов, не удовлетворяющих заданным условиям, и формирования из них множества (selectNStr3-операторы, операторы типа *selectNStr3*)

Операторы данного типа осуществляют ассоциативный поиск произвольных элементов sc-конструкций, удовлетворяющих заданным условиям и состоящих из трех элементов, и формирования из них множества, элементами которого остаются только sc-элементы, не удовлетворяющие заданным условиям.

Запись scp-операторов типа ***selectNStr3*** на языке SCs имеет следующий вид:

 запись selectNStr3 –оператора ::=

selectNStr3 ;

 идентификатор selectNStr3-оператора ;

(* ***fixed_*** : |	/* данный атрибут уточняет, известно (вычислено) ли значение операнда */
[* (* ***const_*** : | ***var_*** : | ***meta_*** : *) *] *)	/* данные атрибуты уточняют тип искомого sc-узла по признаку «константа – переменная – метапеременная» */
1_ : ***операнд x1*** ,	/* операндом является программная переменная, или, если указан атрибут ***fixed_***, то операндом может быть и константа программы */
(* ***fixed_*** : |	/* данный атрибут уточняет, известно (вычислено) ли значение операнда */

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

[* (* ***pos_ :*** | ***neg_ :*** | ***fuz_ :*** *) *]	/* данные атрибуты уточняют тип искомой sc-дуги по признаку «позитивная дуга – негативная дуга – нечеткая дуга» */
[* (* ***const_ :*** | ***var_ :*** | ***meta_ :*** *) *] *)	/* данные атрибуты уточняют тип искомого sc-элемента по признаку «константа – переменная – метапеременная» */
2_ : □ ***операнд x2*** □ **,**	/* операндом является программная переменная, или, если указан атрибут ***fixed_***, то операндом может быть и константа программы */
(* ***fixed_ :*** |	/* данный атрибут уточняет, известно (вычислено) ли значение операнда */
(* [* ***node_ :*** | ***elem_ :*** | ***arc_ :*** *]	/* данные атрибуты уточняют тип искомого sc-элемента по признаку «узел – элемент неопределенного типа – дуга» */
[* (* ***pos_ :*** | ***neg_ :*** | ***fuz_ :*** *) *] . *)	/* данные атрибуты уточняют тип искомой sc-дуги по признаку «позитивная дуга – негативная дуга – нечеткая дуга» */
[* (* ***const_ :*** | ***var_ :*** | ***meta_ :*** *) *] *)	/* данные атрибуты уточняют тип искомого sc-элемента по признаку «константа – переменная – метапеременная» */
3_ : □ ***операнд x3*** □ **,**	/* операндом является программная переменная, или, если указан атрибут ***fixed_***, то операндом может быть и константа программы */
[* ***set1_ :*** □ ***операнд xs1*** □ **,** *]	/* операнд, значением которого является

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

	множество “первых” элементов sc-конструкции */
[* ***set2_ :*** □ ***операнд xs2*** □ **,** *]	/* операнд, значением которого является множество “вторых” элементов sc-конструкции */
[* ***set3_ :*** □ ***операнд xs3*** □ **,** *]	/* операнд, значением которого является множество “третьих” элементов sc-конструкции */
then_ : □ ***идентификатор scp-оператора*** □ **;**	/* здесь указывается идентификатор того scp-оператора, которому передается управление после реализации данного scp-оператора и нахождения требуемой sc-конструкции */
else_ : □ ***идентификатор scp-оператора*** □ **;**	/* здесь указывается идентификатор того scp-оператора, которому передается управление после реализации данного scp-оператора и ненахождения требуемой sc-конструкции */

Результаты выполнения операторов типа ***selectNStr3*** полностью аналогичны результатам выполнения операторов типа ***selectYStr3***, за исключением того что во множествах, которые являются значениями операндов ***xs1 – xs3***, остаются только те sc-конструкции, которые не удовлетворяют заданному условию.

5.2.5. Операторы проверки условий

Семейство scp-операторов проверки условий (такие операторы будем также называть if-операторами) разбивается на следующие типы scp-операторов:

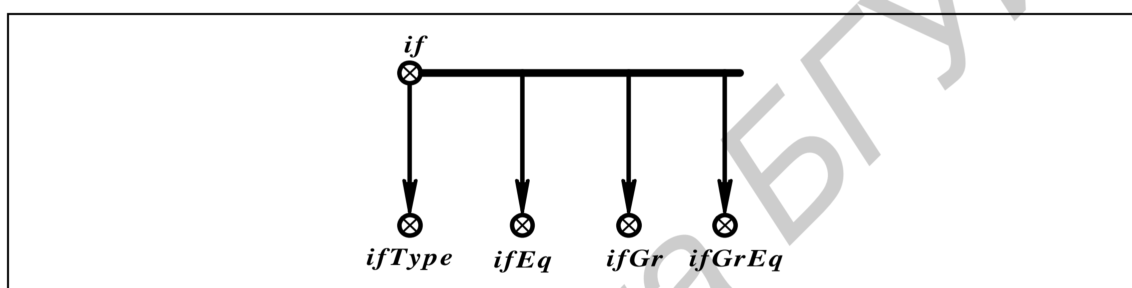
- операторы проверки типа sc-элемента (ifType-операторы);
- операторы проверки равенства значений двух операндов (ifEq-операторы);

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

- операторы проверки строгого неравенства значений двух операндов (ifGr-операторы);
- операторы проверки неравенства значений двух операндов (ifGrEq-операторы).

Множество ifType-операторов в языке SCP будем обозначать ключевым узлом, которому поставим в соответствие идентификатор ***“ifType”***. Множество ifEq-операторов в языке SCP будем обозначать ключевым узлом, которому поставим в соответствие идентификатор ***“ifEq”***. Множество ifGr-операторов в языке SCP будем обозначать ключевым узлом, которому поставим в соответствие идентификатор ***“ifGr”***. Множество ifGrEq-операторов в языке SCP будем обозначать ключевым узлом, которому поставим в соответствие идентификатор ***“ifGrEq”***. Семейство типов if-операторов в языке SCP обозначается ключевым узлом с идентификатором ***“if”***. Соотношение между указанными ключевыми узлами задается следующей sc-конструкцией:



Перейдем к более подробному рассмотрению указанных выше типов scp-операторов.

Операторы проверки типа sc-элемента (ifType-операторы, операторы типа *ifType*)

Операторы данного типа осуществляют проверку типа sc-элемента. Тип проверяется по признаку «константа – переменная – метапеременная», «узел – элемент неопределенного типа – дуга», «позитивная дуга – негативная дуга – нечеткая дуга», а также проверяется, вычислено ли значение переменной.

Запись scp-операторов типа ***ifType*** на языке SCs имеет следующий вид:

*** запись ifType –оператора*** ::=

ifType ;

*** идентификатор***

ifType-оператора ;

(* [* ***node_*** : | ***elem_*** : | ***arc_*** : *] /* данные атрибуты уточняют тип sc-элемента по признаку «узел – элемент неопределенного типа – дуга» */

[* (* ***pos_*** : | ***neg_*** : | ***fuz_*** : *) /* данные атрибуты

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

*] . *)	уточняют тип sc-дуги по признаку «позитивная дуга – негативная дуга – нечеткая дуга» */
[* (* ***const_ :*** | ***var_ :*** | ***meta_ :*** *) *] *)	/* данные атрибуты уточняют тип sc-элемента по признаку «константа – переменная – метапеременная» */
□ ***операнд x1*** □ **,**	/* операндом является scp-переменная или scp-константа */
then_ : □ ***идентификатор scp-оператора*** □ **;**	/* здесь указывается идентификатор того scp-оператора, которому передается управление после реализации данного scp-оператора и если значение операнда удовлетворяет условию, заданному атрибутами */
else_ : □ ***идентификатор scp-оператора*** □ **;**	/* здесь указывается идентификатор того scp-оператора, которому передается управление после реализации данного scp-оператора и если значение операнда не удовлетворяет условию, заданному атрибутами */

Результаты выполнения операторов типа ***ifType***:

96. Если значение операнда не вычислено, то управление передается scp-оператору, помеченному атрибутом else_.
97. Проверяется значение операнда, тип значения уточняется следующими атрибутами:
    - node_, elem_, arc_ – соответственно sc-узел, sc-элемент неопределенного типа или sc-дуга;
    - pos_, neg_, fuz_ – соответственно позитивная, негативная или нечеткая sc-дуга;
    - const_, var_, meta_ – соответственно константный, переменный или метапеременный sc-элемент.
98. Если тип значения операнда соответствует заданному типу, то управление после реализации данного scp-оператора передается

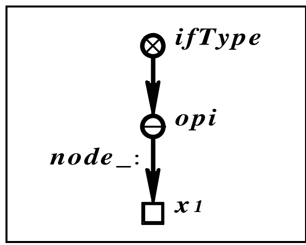
Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

scp-оператору, помеченному атрибутом then_, иначе управление передается scp-оператору, помеченному атрибутом else_.

Модификации scp-операторов типа ***ifType*** задаются комбинацией атрибутов, которыми помечен операнд. Приведем пример модификации ifType-оператора.

Пример 5.29



Если значением операнда ***x1*** является sc-узел, то проверяемое условие считается истинным, иначе условие ложно.

Операторы проверки равенства значений двух sc-элементов (ifEq-операторы, операторы типа *ifEq*)

Операторы данного типа осуществляют проверку равенства значений двух sc-элементов.

Запись scp-операторов типа ***ifEq*** на языке SCs имеет следующий вид:

 запись ifEq –оператора ::=

ifEq ;

 идентификатор ifEq-оператора ;

1_ : ***операнд x1*** , /* операндом является scp-переменная или scp-константа */

2_ : ***операнд x2*** , /* операндом является scp-переменная или scp-константа */

then_: ***идентификатор scp-оператора*** ; /* здесь указывается идентификатор того scp-оператора, которому передается управление после реализации данного scp-оператораи, если значения операндов равны */

else_: ***идентификатор scp-оператора*** ; /* здесь указывается идентификатор того scp-оператора, которому передается управление после реализации данного

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

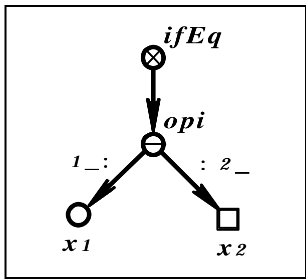
scp-оператора, если значения операндов не равны или не вычислены */

Результаты выполнения операторов типа ***ifEq***:

99. Если значения операндов не вычислены, то управление передается scp-оператору, помеченному атрибутом else_.
100. Если значение первого операнда равно значению второго операнда, то управление передается scp-оператору, помеченному атрибутом then_, иначе управление передается scp-оператору, помеченному атрибутом else_.

Существует одна модификация scp-операторов типа ***ifEq***. Приведем пример этой модификации:

Пример 5.30



Если значение операнда ***x2*** равно значению операнда ***x1*** (в данном случае это сам ***x1***), то условие считается истинным, иначе условие ложно.

Операторы проверки строгого неравенства значений двух sc-элементов (ifGr-операторы, операторы типа *ifGr*)

Операторы данного типа осуществляют проверку значений двух sc-элементов по признаку “больше”.

Запись scp-операторов типа ***ifGr*** на языке SCs имеет следующий вид:

 запись ifGr–оператора **::=**

ifGr ;

 идентификатор ifGr-оператора ;

1_ : ***операнд x1*** , /* операндом является scp-переменная или scp-константа */

2_ : ***операнд x2*** , /* операндом является scp-переменная или scp-константа */

then_ : ***идентификатор scp-оператора*** ; /* здесь указывается идентификатор того scp-оператора, которому

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

передается управление после реализации данного scp-оператора, если значение первого операнда больше значения второго операнда */

else_ **:** □ ***идентификатор scp-оператора*** □ **;** /* здесь указывается идентификатор того scp-оператора, которому передается управление после реализации данного scp-оператора, если значение первого операнда не больше значения второго операнда или значения операндов не вычислены */

Результаты выполнения операторов типа ***ifGr***:

101. Если значения операндов не вычислены, то управление передается scp-оператору, помеченному атрибутом else_.
102. Если значение первого операнда больше значения второго операнда, то управление передается scp-оператору, помеченному атрибутом then_, иначе управление передается scp-оператору, помеченному атрибутом else_.

Существует одна модификация scp-операторов типа ***ifGr*** (аналогично ifEq-оператору).

Операторы проверки неравенства значений двух операндов (ifGrEq-операторы, операторы типа *ifGrEq*)

Операторы данного типа осуществляют проверку значений двух sc-элементов по признаку “больше или равно”.

Запись scp-операторов типа ***ifGrEq*** на языке SCs имеет следующий вид:

□ ***запись ifGrEq–оператора*** □**::=**

ifGrEq **;**

□ ***идентификатор ifGrEq-оператора*** □ **;**

1_ **:** □ ***операнд x1*** □ **,** /* операндом является scp-переменная или scp-константа */

2_ **:** □ ***операнд x2*** □ **,** /* операндом является

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

scp-переменная или scp-константа */

***then_* : ⟨ *идентификатор scp-оператора* ⟩ ;** /* здесь указывается идентификатор того scp-оператора, которому передается управление после реализации данного scp-оператора, если значение первого операнда больше или равно значению второго операнда */

***else_* : ⟨ *идентификатор scp-оператора* ⟩ ;** /* здесь указывается идентификатор того scp-оператора, которому передается управление после реализации данного scp-оператора, если значение первого операнда не больше и не равно значению второго операнда или значения операндов не вычислены */

Результаты выполнения операторов типа ***ifGrEq***:

103. Если значения операндов не вычислены, то управление передается scp-оператору, помеченному атрибутом else_.
104. Если значение первого операнда больше или равно значению второго операнда, то управление передается scp-оператору, помеченному атрибутом then_, иначе управление передается scp-оператору, помеченному атрибутом else_.

Существует одна модификация scp-операторов типа ***ifGrEq*** (аналогично ifEq-оператору).

5.2.6. Операторы изменения свойств элемента конструкций языка SC

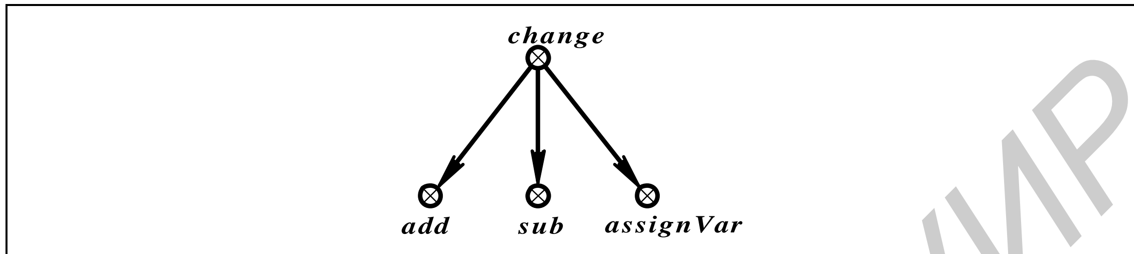
Семейство scp-операторов изменения свойств sc-элемента (такие операторы будем также называть change-операторами) разбивается на следующие типы scp-операторов:

- операторы сложения содержимых sc-элементов (add-операторы);
- операторы вычитания содержимых sc-элементов (sub-операторы);
- операторы пересылки значения программной переменной (assign-операторы).

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Множество add-операторов в языке SCP будем обозначать ключевым узлом, которому поставим в соответствие идентификатор ***“add”***. Множество sub-операторов в языке SCP будем обозначать ключевым узлом, которому поставим в соответствие идентификатор ***“sub”***. Множество assign-операторов в языке SCP будем обозначать ключевым узлом, которому поставим в соответствие идентификатор ***“assign”***. Семейство типов change-операторов в языке SCP обозначается ключевым узлом с идентификатором ***“change”***. Соотношение между указанными ключевыми узлами задается следующей sc-конструкцией:



Перейдем к более подробному рассмотрению указанных выше типов scp-операторов.

Операторы сложения содержимых sc-элементов (add-операторы, операторы типа *add*)

Операторы данного типа осуществляют сложение содержимых двух sc-узлов.

Запись scp-операторов типа ***add*** на языке SCs имеет следующий вид:

□ ***запись add–оператора*** □ ::=

add ;

□ ***идентификатор add-оператора*** □ ;

1_ : □ ***операнд x1*** □ ,	/* операндом является scp-переменная или scp-константа */
2_ : □ ***операнд x2*** □ ,	/* операндом является scp-переменная или scp-константа */
3_ : □ ***операнд x3*** □ ,	/* операндом является scp-переменная или scp-константа */
goto_ : □ ***идентификатор scp-оператора*** □ ;	/* здесь указывается идентификатор того scp-оператора, которому передается управление после реализации данного scp-оператора */

Библиотека БГУИР

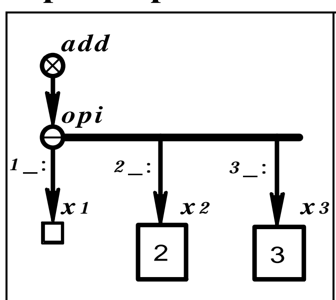
PDF created with pdfFactory Pro trial version www.pdffactory.com

Результаты выполнения операторов типа ***add***:

105. Если значение первого операнда не вычислено (т.е. первый операнд является scp-переменной), то генерируется sc-узел, который становится значением первого операнда. В содержимое сгенерированного sc-узла записывается сумма содержимых значений двух других операндов.
106. Если значение первого операнда вычислено, то содержимое формируется аналогичным образом (см. выше).

Существует одна модификация scp-операторов типа ***add***. Приведем пример этой модификации:

Пример 5.31



В результате выполнения оператора ***opi*** сгенерируется sc-узел с содержимым, равным числу 5, и этот узел станет значением операнда ***x1***.

Операторы вычитания содержимых sc-элементов (sub-операторы, операторы типа *sub*)

Операторы данного типа осуществляют вычитание числового содержимого одного sc-узла из другого.

Запись scp-операторов типа ***sub*** на языке SCs имеет следующий вид:

□ ***запись sub–оператора*** □ ::=	
sub ;	
□ ***идентификатор sub-оператора*** □ ;	
1_ : □ ***операнд x1*** □ ,	/* операндом является scp-переменная или scp-константа */
2_ : □ ***операнд x2*** □ ,	/* операндом является scp-переменная или scp-константа */
3_ : □ ***операнд x3*** □ ,	/* операндом является scp-переменная или scp-константа */
goto_ : □ ***идентификатор scp-оператора*** □ ;	/* здесь указывается идентификатор того scp-оператора, которому передается управление

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

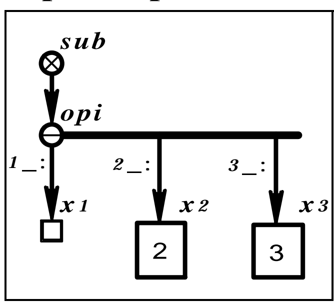
после реализации данного scp-оператора */

Результаты выполнения операторов типа ***sub***:

107. Если значение первого операнда не вычислено (т.е. первый операнд является scp-переменной), то генерируется sc-узел, который становится значением первого операнда. В содержимое сгенерированного sc-узла записывается разность содержимых значений второго операнда и третьего.
108. Если значение первого операнда вычислено, то содержимое формируется аналогичным образом (см. выше).

Существует одна модификация scp-операторов типа ***sub***. Приведем пример модификации.

Пример 5.32



В результате выполнения оператора ***opi*** сгенерируется sc-узел с содержимым, равным числу -1, и этот узел станет значением операнда ***x1***.

Операторы пересылки значения программной переменной (assignVar-операторы, операторы типа *assignVar*)

Операторы данного типа осуществляют присваивание значения одной scp-переменной или scp-константы другой scp-переменой.

Запись scp-операторов типа ***assignVar*** на языке SCs имеет следующий вид:

□ ***запись assignVar–оператора*** □ ::=
assignVar ;
□ ***идентификатор assignVar-оператора*** □ ;
1_ : □ ***операнд x1*** □ , /* операндом является scp-переменная */
2_ : □ ***операнд x2*** □ , /* операндом является scp-переменная или scp-константа */
goto_ : □ ***идентификатор scp-оператора*** □ ; /* здесь указывается идентификатор того scp-оператора, которому передается управление после реализации данного

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

scp-оператора */

Результаты выполнения операторов типа ***assignVar***:

109. Если второй операнд является scp-переменной, то значением первого операнда становится sc-элемент, являющийся значением второго операнда.
110. Если второй операнд является scp-константой, то значением первого операнда становится sc-элемент, обозначающий второй операнд.

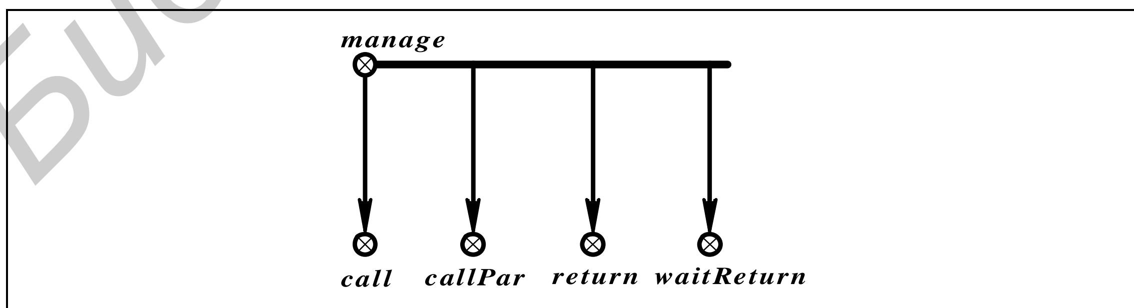
Существует одна модификация scp-операторов типа ***assignVar***.

5.2.7. Операторы управления scp-процессами

Семейство scp-операторов управления scp-процессами (такие операторы будем также называть manage-операторами) разбивается на следующие типы scp-операторов:

- операторы порождения scp-процесса (call-операторы);
- операторы порождения параллельных scp-процессов (callPar-операторы);
- операторы завершения scp-процесса (return-операторы);
- операторы ожидания завершения scp-процесса (waitReturn-операторы).

Множество call-операторов в языке SCP будем обозначать ключевым узлом, которому поставим в соответствие идентификатор ***"call"***. Множество callPar-операторов в языке SCP будем обозначать ключевым узлом, которому поставим в соответствие идентификатор ***"callPar"***. Множество return-операторов в языке SCP будем обозначать ключевым узлом, которому поставим в соответствие идентификатор ***"return"***. Множество waitReturn-операторов в языке SCP будем обозначать ключевым узлом, которому поставим в соответствие идентификатор ***"waitReturn"***. Семейство типов manage-операторов в языке SCP обозначается ключевым узлом с идентификатором ***"manage"***. Соотношение между указанными ключевыми узлами задается следующей sc-конструкцией:



Перейдем к более подробному рассмотрению указанных выше типов scp-операторов.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Операторы порождения scp-процесса (call-операторы, операторы типа *call*)

Операторы данного типа осуществляют порождение scp-процесса для выполнения указанной scp-программы.

Запись scp-операторов типа ***call*** на языке SCs имеет следующий вид:

 запись call–оператора **::=**

call **;**

 идентификатор call-оператора **;**

1_ : ***операнд x1*** **,**	/* значением операнда является sc-узел вызываемой программы. Операнд может быть scp-переменной или scp-константой */
2_ : ***операнд x2*** **,**	/* значением операнда является множество параметров, передаваемых вызываемой программе. Операнд может быть scp-переменной или scp-константой */
3_ : ***операнд x3*** **,**	/* значением операнда после успешной реализации данного scp-операнда становится sc-узел, обозначающий scp-процесс, выполняющий указанную scp-программу. Операнд должен быть scp-переменной */
goto_ : ***идентификатор scp-оператора*** **;**	/* здесь указывается идентификатор того scp-оператора, которому передается управление после реализации данного scp-оператора */

Результаты выполнения операторов типа_***call***:

111. Создается scp-процесс, который будет выполнять scp-программу, обозначенную sc-узлом, являющимся значением первого операнда.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

112. Второй операнд является множеством параметров, передаваемых в scp-программу, а также множеством возвращаемых значений из этой scp-программы (см. п. 5.2.1).
113. SC-узел, обозначающий созданный scp-процесс, становится значением третьего операнда.

П р и м е ч а н и е. Для получения значений из вызываемой программы необходимо после call-оператора поставить waitReturn-оператор.

Существует одна модификация scp-операторов типа ***call***.

Операторы порождения параллельных scp-процессов (callPar-операторы, операторы типа *callPar*)

Операторы данного типа осуществляют порождение параллельных scp-процессов.

Запись scp-операторов типа ***callPar*** на языке SCs имеет следующий вид:

□ ***запись callPar–оператора*** □ **::=**

callPar **;**

□ ***идентификатор callPar-оператора*** □ **;**

1_ **:** □ ***операнд x1*** □ **,** /* значением операнда является sc-узел вызываемой программы. Операнд может быть scp-переменной или scp-константой */

2_ **:** □ ***операнд x2*** □ **,** /* значением операнда является множество, по которому происходит распараллеливание. Операнд может быть scp-переменной или scp-константой */

3_ **:** □ ***операнд x3*** □ **,** /* значением операнда является множество параметров, передаваемых вызываемой программе. Операнд может быть scp-переменной или scp-константой */

goto_ **:** □ ***идентификатор scp-оператора*** □ **;** /* здесь указывается идентификатор того

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

scp-оператора, которому передается управление после реализации данного scp-оператора */

Результаты выполнения операторов типа *callPar*:

114. Создается множество scp-процессов, которые будут выполнять scp-программу, обозначенную sc-узлом, являющимся значением первого операнда. Количество создаваемых scp-процессов зависит от мощности множества, которое является значением второго операнда.
115. Третий операнд является множеством параметров, передаваемых в scp-программу, а также множеством возвращаемых значений из этой scp-программы. В этом множестве должен отсутствовать элемент с атрибутом 1_. Вместо этого элемента в вызываемую программу будут передаваться элементы из второго множества, которое является значением второго операнда. В каждую создаваемую scp-программу будут передаваться различные элементы.

Существует одна модификация scp-операторов типа *callPar*.

Операторы завершения scp-программы (return-операторы, операторы типа *return*)

Операторы данного типа осуществляют завершение scp-программы.

Запись scp-операторов типа *return* на языке SCs имеет следующий вид:

 запись *return*–***оператора*** **::=**

return **;**

 идентификатор

return-***оператора*** **;**

Результат выполнения операторов типа *return*: завершается выполнение scp-программы. Вызвавшей scp-программе передаются возвращаемые значения.

Существует одна модификация scp-операторов типа *return*.

Операторы ожидания завершения scp-процесса (waitReturn-операторы, операторы типа *waitReturn*)

Операторы данного типа осуществляют ожидание завершения scp-процесса.

Запись scp-операторов типа *waitReturn* на языке SCs имеет следующий вид:

 запись *waitReturn*–***оператора*** **::=**

waitReturn **;**

 идентификатор

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

*waitReturn-**оператора*** □ ;

[* ***process_ :*** □ ***операнд*** □ **,** *]	/* значением операнда является sc-узел, обозначающий scp-процесс, завершение которого необходимо ожидать. Операндом должна быть scp-переменная */
goto_ : □ ***идентификатор scp-оператора*** □ **;**	/* здесь указывается идентификатор того scp-оператора, которому передается управление после реализации данного scp-оператора */

Результаты выполнения операторов типа ***waitReturn***:

116. Если есть операнд с атрибутом process_, то происходит ожидание завершения scp-процесса, обозначенного sc-узлом, являющимся значением данного операнда.
117. Если операнда с атрибутом process_ нет, то происходит ожидание завершения всех дочерних scp-процессов данного scp-процесса.

Существуют две модификации scp-операторов типа ***waitReturn***.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Литература

1. ***Гордеев А..2001кн-СистеПО***
   Гордеев А., Молчанов А. Системное программное обеспечение. – СПб.: Питер, 2001. – 736 с.
2. ***Дейтел Г.1987кн-ВведеВОС***
   Дейтел Г. Введение в операционные системы: В 2 т. Пер. с англ.– М.: Мир, 1987. – 428 с.
3. ***Кузьмицкий В.М.1999ст-ПринцПГАП***
   Кузьмицкий В.М. Принципы построения графодинамической ассоциативной памяти // Интеллектуальные системы: Сб. науч. тр. Вып.2. / НАН Беларуси. Ин-т техн.кибернетики; Науч.ред. А.М.Крот.– Мн, 1999. – С.125-133.
4. ***Лорин Г..1984кн-ОпераС***
   Лорин Г., Дейтел Х.М. Операционные системы.: Пер. с англ.– М.: Финансы и статистика, 1984. – 243 с.
5. ***Олифер В.Г..2001кн-СетевОС***
   Олифер В.Г., Олифер Н.А. Сетевые операционные системы. – СПб.: Питер, 2001. – 189 с.
6. ***ПредсИОЗвГАМ-2001кн***
   Представление и обработка знаний в графодинамических ассоциативных машинах / В.В.Голенков, О.Е.Елисеева, В.П.Ивашенко и др.; Под ред. В.В.Голенкова. – Мн.: БГУИР, 2001. – 237 с.
7. ***ПрогрВАМ-2001кн***
   Программирование в ассоциативных машинах/ В.В.Голенков, Г.С.Осипов, Н.А.Гулякина и др. – Мн.:БГУИР, 2001. – 175 с.
8. ***Сердюков Р.Е.2000ст-РазраМУПП***
   Сердюков Р.Е. Разработка методов управления параллельными процессами при распределенной переработке базы знаний. // Сб. науч. тр. IV междунар. летней школы-семинара по искусственному интеллекту для студентов и аспирантов. Мн.: БГУИР, 2000. – С.157-163.
9. ***Столлингс В.2002кн-ОпераС***
   Столлингс В. Операционные системы. 4-е изд. / Пер. с англ. – М.: Издательский дом “Вильямс”, 2002. – 672 с.

Таненбаум Э.2002кн-СовреОС
Таненбаум Э. Современные операционные системы. – СПб.: Питер, 2002. – 1040 с.

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com

Св. план 2005, поз. 35

Учебное издание

Голенков Владимир Васильевич,
Сердюков Роман Евгеньевич

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ В ГРАФОДИНАМИЧЕСКИХ АССОЦИАТИВНЫХ МАШИНА

Учебное пособие по курсу
Операционные системы традиционных и интеллектуальных компьютеров
для студентов специальности I-40 03 01 «Искусственный интеллект» всех форм обучения

Редактор Т.А. Лейко
Корректор Н.В. Гриневич

Подписано в печать	Формат 60х84 1/8.	Бумага офсетная.
Гарнитура «Таймс».	Печать ризографическая.	Усл. печ. л.
Уч.-изд.л.	Тираж 100 экз.	Заказ

Издатель и полиграфическое исполнение: Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»
Лицензия на осуществление издательской деятельности №02330/0056964 от 01.04.2004.
Лицензия на осуществление полиграфической деятельности №02330/0131518 от 30.04.2004.
220013, Минск, П. Бровки, 6

Библиотека БГУИР

PDF created with pdfFactory Pro trial version www.pdffactory.com