Федеральное агентство по образованию

Государственное образовательное учреждение
высшего профессионального образования
Московский государственный университет леса

Государственный экзамен по направлению 552800 (230100) «Информатика и вычислительная техника»

Методические указания
Для студентов специальности
552800 (230100)

Издательство Московского государственного университета леса
Москва — 2006

УДК 004.451(076.3)
Г72

Г72 **Государственный** экзамен по направлению 552800 (230100) «Информатика и вычислительная техника»: Методические указания для студентов специальности 552800 (230100). / Составитель А. В. Чернышов. — М.: МГУЛ, 2006. — 94 с.: ил.

Настоящие методические указания содержат материалы, необходимые студентам IV курса потока ВТ для подготовки к государственному экзамену.

Разработано в соответствии с Государственным образовательным стандартом ВПО 2000 г. для подготовки специалистов по направлению 654600 на основе примерной программы дисциплины «Операционные системы» для специальности 552800 (230100) «Информатика и вычислительная техника».

Одобрено и рекомендовано к изданию в качестве методических указаний редакционно-издательским советом университета

Рецензент —

Кафедра вычислительной техники

Составитель — Александр Викторович Чернышов, доцент

© Чернышов А. В., 2006
© Московский государственный университет леса, 2006

Предисловие

Настоящие методические указания предназначены для подготовки к выпускному государственному экзамену студентов IV курса, обучающихся по направлению 552800 (230100) «Информатика и вычислительная техника». В них собраны материалы, касающиеся общих вопросов подготовки к экзамену и его проведения, предметов и вопросов, вынесенных на экзамен.

Методические указания содержат также краткий конспект ответов на вопросы экзамена. Ограниченный объём издания не позволил включить в него развёрнутые ответы.

Для подготовки к экзамену рекомендуется прежде всего пользоваться одной из книг, приведённых в списке литературы. А конспект использовать только для систематизации полученных знаний и определения основных моментов, которые желательно осветить в ответе на каждый вопрос.

Составитель выражает глубокую благодарность студентам потока ВТ IV курса 2005 г., чьи конспекты, выполненные в процессе подготовки к государственному экзамену, легли в основу настоящих методических указаний.

Положение о проведении государственного экзамена по направлению «Информатика и вычислительная техника»

Государственный экзамен по специальности (далее Экзамен) является итоговым испытанием, на котором студенты-выпускники должны подтвердить качество своих профессиональных знаний, полученных в процессе обучения.

Экзамен является междисциплинарным и включает в себя проверку знаний по группе изучавшихся специальных дисциплин.

Перечень вынесенных на Экзамен дисциплин утверждается не позднее, чем за полгода до проведения Экзамена и доводится до сведения студентов.

Экзамен сдаётся в устной форме Государственной экзаменационной комиссии на закрытом заседании (посторонние и зрители не допускаются).

Порядок подготовки к экзамену

Кафедра не позднее, чем за полгода до проведения экзамена доводит до сведения студентов перечень специальных дисциплин, вынесенных на Экзамен.

Перечень вопросов по каждой дисциплине, оформляется отдельным списком и доводится до сведения студентов не позднее, чем за полгода до проведения экзамена.

Перед проведением Экзамена по каждой дисциплине проводятся консультации студентов с преподавателем, ведущим данную дисциплину.

Кроме того, проводится консультация студентов с секретарём ГЭК по общим вопросам проведения Экзамена.

Для проведения Экзамена составляются экзаменационные билеты. Каждый билет содержит два вопроса из разных дисциплин, произвольно выбранных из перечней вопросов по дисциплинам, вынесенным на Экзамен.

Содержание экзаменационных билетов до сведения студентов не доводится.

На каждое заседание ГЭК составляется список студентов, допущенных к Экзамену.

Даты заседания ГЭК и списки допущенных утверждаются на кафедре и доводятся до сведения студентов.

Порядок проведения экзамена

Для проведения Экзамена выделяется аудитория, удовлетворяющая следующим требованиям:

– наличие мест для размещения Государственной экзаменационной комиссии;

– наличие мест для подготовки студентов к ответу из расчёта один полный стол (парта) на студента;

– наличие доски для рисования мелом.

Заседание ГЭК начинается с входом в аудиторию первой партии студентов.

Студенты берут билеты, номера которых регистрируются в протоколе заседания, и приступают к подготовке на указанных им местах в аудитории.

На подготовку к ответу в среднем отводится 30 минут.

В течение первых 5 минут подготовки студент имеет право обратиться к комиссии с просьбой о замене билета. Билет может быть заменён только один раз. За замену билета оценка не понижается.

Студент готовится к ответу, составляя краткий конспект ответа и делая другие необходимые пометки на листах бумаги. Эти листы после ответа студента сдаются ГЭК.

Ответ студента на поставленные в билете вопросы осуществляется устно перед ГЭК в виде краткого доклада в соответствии с составленным им конспектом. При этом, учитывая междисциплинарный состав ГЭК, доклад студента должен быть направлен на изложение основных положений и идей, касающихся поставленного вопроса. Несущественные детали и подробности необходимо опускать.

В процессе доклада для иллюстрации своих высказываний студент может пользоваться доской и мелом.

Продолжительность доклада по каждому вопросу не должна превышать 5 минут.

По окончании доклада члены ГЭК имеют право задать студенту дополнительные вопросы, касающиеся вопросов экзаменационного билета.

После ответов на вопросы ГЭК студент сдаёт билет и листы своих записей и покидает аудиторию. Возвращённый студентом билет в течение данного заседания ГЭК больше к раздаче не допускается.

На место ответившего студента в аудиторию допускается следующий студент, который берёт билет и начинает готовиться к ответу.

В это время начинает отвечать следующий студент, из числа уже готовившихся к ответу.

После ответа последнего студента из списка допущенных к заседанию ГЭК проводит закрытое заседание, на котором устанавливаются оценки всем студентам.

Оценки доводятся до сведения студентов в тот же день сразу после заседания ГЭК.

* * *

На государственный экзамен по направлению «Информатика и вычислительная техника» вынести вопросы по курсу «Операционные системы».

Выписка из заседания кафедры Вычислительной техники
№ XX от XX ааааааа 2003 г.

Вопросы к государственному экзамену по направлению «Информатика и вычислительная техника»

1. Понятие операционной системы, её назначение. Современные ОС.

2. Основные виды классификаций ОС.

3. Понятие мобильной операционной системы. ОС UNIX.

4. Понятие открытого программного обеспечения. Его преимущества. Программное обеспечение GNU.

5. Пакетные операционные системы.

6. Операционные системы разделения времени и многопользовательские ОС.

7. Операционные системы реального времени.

8. Иерархический принцип построения операционной системы. Простая и расширенная машины.

9. Виртуальные машины.

10. Цели и задачи применения мультипрограммирования.

11. Понятие ядра операционной системы.

12. Понятия процесса и потока.

13. Планирование процессов как функция ядра операционной системы.

14. Понятие ресурса. Оперативно перераспределяемые и оперативно неперераспределяемые ресурсы.

15. Распределение ресурсов и управление ресурсами как функция операционной системы.

16. Понятие взаимоисключения нескольких процессов и критические участки.

17. Алгоритмы взаимоисключения Деккера и Петерсона.

18. Семафоры и мьютексы.

19. Реализация взаимоисключения на семафорах.

20. Мониторы ресурсов и реализация взаимоисключения на мониторах.

21. Реализация взаимоисключения на аппаратном уровне.

22. Тупики и методы борьбы с ними.

23. Методы предотвращения тупиков.

24. Методы обхода тупиков. Алгоритм банкира.

25. Методы обнаружения тупиков.

26. Методы восстановления после тупиков.

27. Методы управления оперативной памятью.

28. Стратегии поиска подходящего блока оперативной памяти.

29. Понятие виртуального ресурса.

30. Виртуальная память. Принцип организации и основной алгоритм функционирования.

31. Страничная организация виртуальной памяти.

32. Сегментная организация виртуальной памяти.

33. Странично-сегментная организация виртуальной памяти.

34. Проблема предотвращения «пробуксовки» системы.

35. Проблема эффективности при планировании процессов в системе.

36. Стратегии управления планированием процессов в системе.

37. Трёхуровневое планирование выполнения задач в системе.

38. Кэширование. Принцип работы кэш-памяти.

39. Управление вводом-выводом как функция операционной системы.

40. Назначение каналов ввода-вывода и организация управления ими в операционной системе.

41. Управление печатью на принтере как функция операционной системы.

42. Назначение файловых систем.

43. Поддержка файловой системы как функция операционной системы.

44. Варианты организации доступа к файлам в операционной системе. Преимущества и недостатки.

45. Понятие драйвера. Аппаратные и программные драйвера.

46. Иерархия драйверов в операционной системе.

47. Проблема эффективности при доступе к вращающимся накопителям информации (например, жёстким дискам).

48. Стратегии оптимизации среднего времени доступа к жёсткому диску.

49. Условия эффективного и неэффективного применения стратегий оптимизации среднего времени доступа к жёсткому диску.

50. Эффективность функционирования операционной системы.

51. Цели и методы сбора информации об эффективности функционирования операционной системы и ЭВМ.

52. Оптимизация работы вычислительной системы.

53. Программы с оверлейной структурой. Цель применения. Принципы построения и функционирования. Преимущества и недостатки.

54. Раскручивающие загрузчики. Назначение. Принцип многоступенчатой загрузки ОС.

55. Проблема безопасности в операционных системах. Основные вопросы защиты.

56. Программирование для многопроцессорных структур.

57. Классификация многопроцессорных структур.

58. Многопроцессорные операционные системы.

59. Сетевые операционные системы.

60. Распределённые операционные системы.

Конспект ответов на вопросы

1. Понятие ОС, её назначение. Современные ОС

Под операционной системой (ОС) понимается организованная совокупность программ (как обычных, так и микропрограмм), которая действует как интерфейс между аппаратурой ЭВМ и пользователем. Задача ОС заключается в том, чтобы:

– облегчить проектирование, программирование, отладку и сопровождение программ;

– распределить ресурсы ЭВМ с целью эффективного использования всех её компонентов (центрального процессора, устройств ввода/вывода и т. п.).

Операционная система является неотъемлемой частью любого современного компьютера.

Операционная система реализует следующие функции:

– определяет интерфейс пользователя;

– обеспечивает разделение аппаратных средств между пользователями;

– планирует доступ пользователей к общим ресурсам;

– обеспечивает эффективное выполнение операций ввода и вывода;

– осуществляет восстановление информации и вычислительного процесса в случае ошибок.

Операционная система управляет следующими основными ресурсами:

– процессорами;

– памятью;

– устройствами ввода/вывода;

– данными.

Операционная система взаимодействует с:

– операторами ЭВМ;

– прикладными программистами;

– системными программистами;

– административным персоналом;

– программами;

– аппаратными средствами;

– пользователями.

Операторы ЭВМ осуществляют непосредственное управление вычислительным процессом.

Прикладные программисты занимаются разработкой, отладкой и сопровождением целевых программ для конечных пользователей.

Системные программисты занимаются сопровождением ОС, осуществляют её настройку применительно к требованиям конкретной машины и при необходимости доработку для обслуживания новых типов устройств.

Администраторы систем устанавливают порядок работы на ЭВМ и взаимодействуют с ОС, чтобы обеспечить соблюдение принятого порядка.

Программы обращаются к ОС при помощи специальных команд (вызов монитора, супервизора и т.п.), не нарушающих её целостности и работоспособности.

Пользователи решают на ЭВМ свои задачи (производственные, научные и т. п.) в соответствии с выделенными им полномочиями. Операционной системе как правило присваивается статус самого полномочного пользователя.

Современных ОС насчитывается более сотни. Наиболее известные из них — это универсальные ОС, так или иначе тяготеющие к персональным компьютерам: семейство Unix (Linux, FreeBSD, QNX, Solaris), семейство Windows (XP, NT, 2000, Millenium), OS/2, — а также ОС для карманных компьютеров: Palm OS, Windows CE. Ситуация на рынке современных ОС меняется каждый год.

2. Основные виды классификаций ОС

ОС можно классифицировать по различным признакам.

1) По количеству процессоров в системе:
– однопроцессорные;
– многопроцессорные.

2) По способу организации вычислений:
– системы с пакетной обработкой;
– системы обработки транзакций;
– системы оперативного доступа (разделения времени);
– системы реального времени («жёсткого» и «мягкого»).

3) По назначению:
– универсальные;
– специализированные (в том числе встроенные).

4) По количеству одновременно выполняемых задач:
– однозадачные;
– многозадачные.

5) По количеству одновременно обслуживаемых пользователей:

– однопользовательские;
– многопользовательские.

6) По архитектурному принципу (концепции организации): монолитное или микроядро, наличие/отсутствие многозадачности с переключением, использование/отсутствие механизмов виртуальной памяти и т. п.

7) По типу интерфейса пользователя:
– работа только в пакетном режиме;
– интерфейс типа командной строки;
– интерфейс типа интерактивного меню;
– интерфейс типа графической рабочей среды.

Возможны и другие типы классификации.

3. Понятие мобильной ОС. ОС Unix

Если код ОС может быть сравнительно легко перенесён с процессора одного типа на процессор другого типа и с аппаратной платформы одного типа на аппаратную платформу другого типа, то такую ОС называют переносимой (мобильной). Мобильность — это понятие степени. Вопрос не в том, может ли быть система перенесена, а в том, насколько легко можно это сделать.

Для того, чтобы обеспечить свойство мобильности ОС, разработчики должны следовать следующим правилам.

1. Большая часть кода быть написана на языке, трансляторы которого имеются на всех машинах, куда предполагается переносить систему. Такими языками являются стандартизованные языки высокого уровня. Большинство переносимых ОС написано на языке С, который имеет много особенностей, полезных для разработки кодов операционной системы, и компиляторы которого широко доступны. Язык Ассемблера используется только для тех непереносимых частей системы, которые должны непосредственно взаимодействовать с аппаратурой, или для частей, которые требуют максимальной скорости.

2. Объем машинно-зависимых частей кода, которые непосредственно взаимодействуют с аппаратными средствами, должен быть по-возможности минимизирован. Так, например, следует всячески избегать прямого манипулирования регистрами и другими аппаратными средствами процессора.

3. Аппаратно-зависимый код должен быть надёжно изолирован в нескольких модулях, а не быть распределён по всей системе. Изоляции подлежат все части ОС которые отражают специфику как процессора, так и аппаратной платформы в целом. Низкоуровневые компоненты ОС, имеющие доступ к процессорно-зависимым структурам данных и регистрам, должны быть оформлены в виде компактных модулей, которые могут быть заменены аналогичными модулями для других процессоров.

В понятие мобильность ОС входит также и мобильность её программного обеспечения (прикладного и системного). На данный момент для обеспечения мобильности существующих и вновь разрабатываемых ОС разработано семейство стандартов POSIX. ОС, следующая рекомендациям POSIX, обладает стандартизованными интерфейсами, гарантирующими пользователям удобство её применения.

Unix — полноценная, изначально многопользовательская, многозадачная и многотерминальная операционная система. На сегодняшний день семейство Unix включает большое число ОС, самые известные из которых Linux, FreeBSD, Solaris, AIX, HP UX. Каждая ОС в семействе соответствует стандарту POSIX, чем обеспечивается мобильность самих ОС и возможность простой переносимости программ между этими ОС.

4. Понятие открытого программного обеспечения. Его преимущества. Программное обеспечение GNU

«Открытость» в мире ПО понимается как предоставление пользователям и сторонним разработчикам какой-либо информации о функционировании, структуре, интерфейсах (и т. п.) какого-либо программного продукта. Предполагается, что предоставленная информация должна позволить потребителям использовать программный продукт с большей степенью эффективности.

В настоящее время можно чётко выделить следующие типы «открытости».

1) Открытые интерфейсы — предоставляется информация по используемым в продукте программным запросам и протоколам, позволяющая сторонним разработчикам создавать свои продукты, опираясь на взаимодействие с данным продуктом. Иногда разработчик открывает (описывает) интерфейсы собственной разработки. Но значительно чаще разработчик продукта сам использует интерфейсы, построенные на базе опубликованных стандартов. Это позволяет, в

частности, использовать программный продукт в комплексе с другими программами практически без доработок.

Необходимо понимать, что сам программный продукт при этом может быть коммерческим и предоставляться пользователю по лицензии, ограничивающей его распространение и применение.

1a) Открытые алгоритмы — потребителям предоставляется информация об алгоритмах, используемых в программном продукте. В частности, информация об алгоритмах формирования/чтения массивов данных, файлов и т. п. На основании этой информации потребители могут подбирать сторонние программные продукты работы с данными, формируемыми данным программным продуктом, либо писать свои программы.

2) Открытые исходные коды — потребителям предоставляются исходные тексты программного продукта. Потребители могут изучать эти тексты в различных целях, самостоятельно транслировать и получать программный продукт в исполняемой форме.

Другие возможности пользователей зависят от типа лицензии:

– Open Source — пользователям разрешается вносить в исходный текст свои исправления и дополнения с целью устранения замеченных ошибок и введения дополнительных возможностей. Приветствуется распространение исправленных (улучшенных) версий программного продукта с обязательным условием предоставления исходных текстов. Недостатком системы является правовая неопределённость — программист, внёсший изменения в текст, может объявить эти изменения своей собственностью и закрыть для других потребителей;

– Public Domain — потребители имеют право бесплатно копировать, использовать и распространять программный продукт. Исходные тексты разрешено изучать, но запрещено модифицировать в рамках данного продукта. При необходимости создания новой программы на базе данных исходных текстов новая программа должна получить другое название. Под новым названием возможно её дальнейшее распространение.

При обнаружении серьёзных ошибок в исходном программном продукте изменения в его текст может вносить только разработчик.

3) Открытая лицензия — программный продукт распространяется по лицензии, гарантирующей потребителю права:

– получения программного продукта бесплатно или по цене копирования (не предполагающей извлечения коммерческой выгоды);

– получения исходных текстов программного продукта на аналогичных условиях;

– использования программного продукта по своему усмотрению, в своих целях (в том числе для извлечения коммерческой выгоды) без каких-либо ограничений;

– исправления программного продукта для устранения ошибок и введения дополнительных возможностей;

– создания собственных программных продуктов с применением исходных текстов из других программных продуктов;

– распространения исходного и/или исправленного программного продукта бесплатно или по цене копирования без каких-либо ограничений, но с обязательным условием соблюдения всех гарантий исходной лицензии;

– получения коммерческой прибыли за поддержку/сопровождение программного продукта у других потребителей.

Существует несколько таких лицензий, самой известной из которых является «Публичная лицензия GNU».

Эта лицензия даётся на ПО, предоставляемое мировому сообществу Фондом свободного программного обеспечения (FSF). В настоящее время фонд объединяет большое количество программ, аналогичных программам коммерческих разработчиков и часто превосходящих их по возможностям и качеству работы (иногда, впрочем, уступающих им).

Лицензия даёт потребителям программ все права, необходимые для получения, эффективного использования, совершенствования и распространения программных продуктов. Единственное условие лицензии — потребители, использующие программы, должны соблюдать все требования этой лицензии, то есть сохранять за всеми остальными потребителями те же права, которые получили они сами.

Преимуществами открытого ПО являются:

– возможность создания систем обработки данных как комплексов программных продуктов разных разработчиков, в том числе работающих на разных платформах;

– обеспечение переносимости программ и данных между различными ОС и платформами;

– первоочередная реализация в программных продуктах наиболее востребованных потребителями функций;

– широкомасштабное тестирование программных продуктов и, как следствие, их высокая надёжность;

– независимость от поведения первоначального разработчика программного продукта.

5. Пакетные ОС

В первых ЭВМ ОС отсутствовали. В результате для того, чтобы выполнить программу, программисту необходимо было вручную подготовить ЭВМ к работе — загрузить в определённые области памяти программу и данные, проинициализировать необходимые устройства ЭВМ, затем выдать команду на старт программы.

По завершении работы программы необходимо было вручную привести ЭВМ в исходное состояние — «почистить».

Эти ручные операции отнимали слишком много очень дорогого машинного времени. В связи с этим были предприняты попытки автоматизировать процессы подготовки ЭВМ к работе и её «чистки».

Решением явились пакетные операционные системы. Основная их идея заключалась в том, что программист не вводил программу в ЭВМ сам, а готовил и отдавал свою программу на перфокартах оператору. Оператор, собрав несколько таких заданий от программистов, формировал пакет — снабжал колоду перфокарт специальными управляющими картами, которые обеспечивали в необходимые моменты времени вызов специальной небольшой управляющей программы — монитора. Монитор обеспечивал загрузку в память очередной программы и её данных, передавал ей управление, а после окончания работы программы автоматически «чистил» память ЭВМ и загружал следующую программу из пакета.

Такой порядок работы позволил значительно сократить простои ЭВМ и поднять эффективность её использования.

В дальнейшем для управления загрузкой и выполнением программ в пакете были разработаны специальные языки управления (JCL), а колоды перфокарт были заменены на магнитные ленты. Сами ленты с пакетами заданий для больших ЭВМ готовились на малых ЭВМ, получивших название сателлитов.

Пакетные ОС и сегодня используются в крупных вычислительных центрах, выполняющих преимущественно вычислительные (неинтерактивные) задания.

6. ОС разделения времени и многопользовательские ОС

В однозадачном режиме работы процессор периодически вынужден простаивать в ожидании окончания операций ввода и вывода, поскольку устройства ввода-вывода работают значительно медленнее процессора.

Желание программистов использовать время простоя процессора привело к появлению многозадачных ОС.

Предпосылками к появлению таких систем явились:

– увеличение объёма ОП, что позволило загружать в память одновременно несколько задач;

– появление системы прерываний (в частности прерываний по окончанию операций ввода-вывода и прерываний таймера);

– появление каналов ввода-вывода, осуществляющих прямой доступ к ОП, минуя процессор.

Если теперь какая-либо задача блокировалась до окончания операции ввода-вывода, то процессор мог переключиться на выполнение другой задачи, и процессорное время использовалось более эффективно.

Многозадачный режим работы может быть реализован с применением разных стратегий планирования, что влияет на выбор для выполнения на процессоре очередной задачи, из стоящих в очереди. Все стратегии планирования, кроме того, делятся на прерываемые и непрерываемые.

Для пакетных ОС удобнее применять непрерываемые стратегии.

Для ОС, в которых важно время отклика задачи, в частности, для ОС, в которых предусмотрен интерактивный режим работы пользователей, применяются стратегии с прерываниями, получившие обобщённое название планирования с разделением времени.

Основная идея стратегии планирования с разделением времени заключается в том, что каждой работающей в системе задаче выделяется свой квант времени, по истечение которого она обязана передать управление следующей задаче в очереди. Таким образом каждая работающая в системе задача периодически получает возможность выполняться на процессоре, и у пользователей складывается впечатление, что их задачи постоянно готовы воспринимать их команды. Для реализации такой стратегии в системе необходимо наличие системного таймера и прерывания по таймеру.

В зависимости от стоящих перед системой целей, очередная задача выбирается из очереди готовых задач либо произвольным образом,

либо в порядке общей очереди, либо в соответствии с присвоенным ей приоритетом.

При правильной организации планирования ОС разделения времени обеспечивает всем пользователям, работающим в системе, разумные времена ответов на их запросы.

Из-за необходимости постоянного переключения между задачами режим разделения времени менее эффективно использует ресурсы ЭВМ для решения задач, чем режим пакетной обработки. Но разработчики ОС идут на это сознательно, поскольку в данном случае преследуется цель повышения эффективности работы не самого компьютера, а человека, взаимодействующего с компьютером.

7. ОС реального времени

Эти ОС предназначены для различных систем управления и делятся на два типа:

– жёсткого реального времени;

– мягкого реального времени.

Системы жёсткого РВ предназначены для управления процессами, в которых максимальные времена реакций на все поступающие внешние события жёстко определены и ни в коем случае не могут быть превышены. Последствия превышения заданных времён могут привести к катастрофическим последствиям.

Примерами таких систем являются системы управления ядерными электростанциями, химическими производствами, различные автопилоты и т. п.

В системах мягкого реального времени также заданы максимальные времена реакций на события, но превышение этих времён, хотя и ухудшит работу системы в целом, но не приведёт к катастрофическим последствиям.

Главное свойство систем РВ — предсказуемость времени реакции. Все реакции системы просчитываются заранее на этапе проектирования и должны быть гарантированы по времени.

В силу указанных свойств системы РВ обычно работают с большой недогрузкой. Их главная задача — обеспечить своевременную реакцию на все возможные события, поступающие в непредсказуемые моменты времени, в пределе — одновременно. Из-за этого системы РВ очень дороги в реализации, но использовать вместо них системы других типов невозможно, поскольку негарантированность времени реакции может привести к катастрофическим последствиям, ущерб от которых может на несколько порядков превышать стоимость системы РВ.

8. Иерархический принцип построения ОС. Простая и расширенная машины

При проектировании ОС удобно выделять несколько уровней иерархии. В основе иерархии лежит аппаратура компьютера. На следующем уровне иерархии находится ядро ОС. Над ядром в иерархии находятся различные процессы ОС, которые обеспечивают поддержку процессов пользователя (например, процессы управления внешними устройствами). На вершине иерархии располагаются сами процессы пользователей.

Иногда несколько уровней иерархии выделяют и внутри самого ядра ОС. Например, могут быть выделены уровни диспетчера процессов, менеджера памяти, супервизора ввода-вывода и т. п.

Выделяют два типа иерархических схем:

– строгие — из данного уровня иерархии возможно обращение только к рядом лежащим уровням. При необходимости обратиться к более глубоко лежащим уровням иерархии используются специальные вызовы, ретранслируемые с уровня на уровень;

– прозрачные — из данного уровня иерархии возможно обращение к любому уровню иерархии.

Иерархическое построение ОС позволяет строго описать интерфейсы каждого уровня иерархии, что позволяет в свою очередь создавать и отлаживать эти уровни независимо друг от друга. Их могут даже разрабатывать параллельно разные программисты, что позволяет сократить время разработки.

Полезно различать понятия реальной и расширенной машины.

Реальная машина — набор аппаратных средств самой ЭВМ, предоставляющий, в частности набор команд процессора, регистров ввода-вывода периферийных устройств и т. п.

Расширенная машина — набор стандартных подпрограмм, предоставляемых операционной системой прикладным программам в качестве стандартных средств по выполнению различных системных функций (распределение памяти, организация ввода-вывода и др.). При этом средства расширенной машины значительно проще в использовании и менее подвержены ошибкам, поскольку в них учтено множество нюансов выполнения соответствующих операций.

9. Виртуальные машины

Термин «виртуальный» в буквальном переводе означает «кажущийся». Говоря о виртуальных машинах, обычно имеют в виду такой способ организации вычислений, когда каждому из множества пользователей, работающих на одной ЭВМ, кажется, что он работает на этой машине один, и все ресурсы машины полностью находятся в его распоряжении.

В частности, каждый пользователь имеет возможность запустить собственную ОС и работать с ней, не мешая другим пользователям.

Такая организация вычислений возможна с применением специального системного ПО, получившего название Менеджер виртуальных машин (МВМ). В типичной системе виртуальных машин МВМ становится самым низким уровнем иерархии ПО, выполняющимся непосредственно на аппаратуре ЭВМ (рис. 9.1). Все остальные программы, в том числе все ОС, загружаются под его управлением и занимают более высокие уровни иерархии. Для обеспечения возможности поочерёдной работы всех загруженных ОС МВМ реализует режим разделения времени.

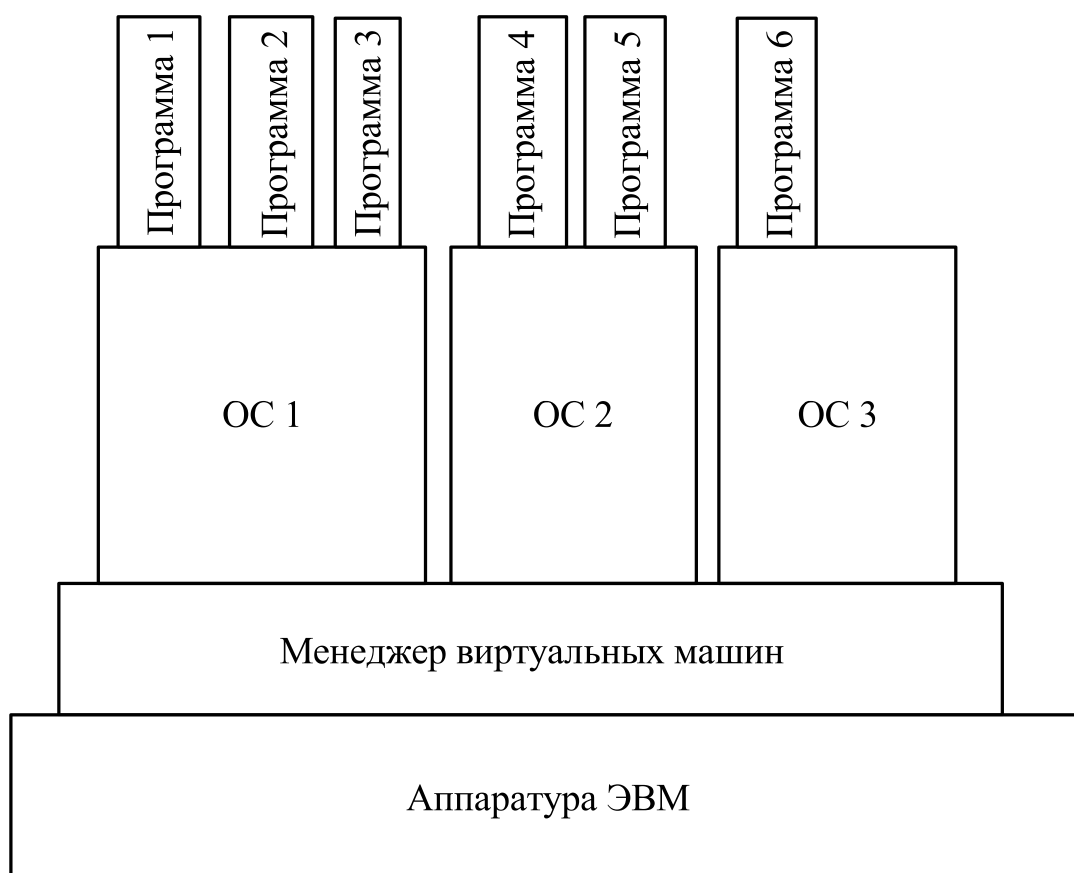

Рис. 9.1. Принцип организации виртуальных машин

Поскольку загруженные ОС «считают», что они выполняются непосредственно на аппаратуре ЭВМ, необходима аппаратная поддержка перехвата привилегированных команд, выполняемых ядрами этих ОС. Перехваченные команды обрабатываются МВМ в режиме эмуляции их выполнения, после чего управление возвращается ОС.

МВМ должен также разделять все ресурсы ЭВМ между загруженными ОС, эмулируя при необходимости недостающие.

10. Цели и задачи мультипрограммирования

Мультипрограммирование или многозадачность — это способ организации вычислительного процесса, при котором на одном процессоре попеременно выполняются сразу несколько программ. Эти программы совместно используют не только процессор, но и другие ресурсы компьютера: оперативную и внешнюю память, устройства ввода-вывода, и т. п.

Мультипрограммирование призвано повысить эффективность использования вычислительной системы. Идея мультипрограммного режима работы заключается в том, что пока одна программа ожидает завершения очередной операции ввода-вывода или наступления какого-либо события, другая программа может быть поставлена на решение. Это позволяет более полно использовать имеющиеся ресурсы и уменьшить общее время, необходимое для решения некоторого множества задач.

Наиболее характерными целями мультипрограммирования являются:

– увеличение пропускной способности — количества задач, выполняемых вычислительной системой в единицу времени;

– удобство работы пользователей, заключающееся, в частности, в том что они имеют возможность интерактивно работать одновременно с несколькими приложениями на одной машине;

– улучшение реактивности системы — способности системы выдержать заранее заданные интервалы времени между запуском программы и получением результата.

Важно понимать, что описанные цели могут быть достигнуты лишь «в среднем». Отдельно взятая задача при мультипрограммном режиме работы никогда не сможет выполниться быстрее, чем в однозадачном режиме. А во многих случаях время выполнения отдельной задачи может даже увеличиться. Но в общем смесь задач выполняется с большей эффективностью, чем при однозадачном способе организации вычислений.

При организации мультипрограммного режима работы системы необходимо уделять внимание разделению ресурсов системы между параллельно работающими процессами и особое внимание вопросам защиты программ друг от друга и ОС от программ пользователей. Если такую защиту организовать невозможно, то система не может считаться надёжной.

11. Понятие ядра ОС

Ядро является главной частью любой ОС, определяющей все основные свойства ОС, касающиеся управления процессами и взаимодействия с пользователями. Функции, выполняемые модулями ядра, являются наиболее часто используемыми функциями операционной системы, поэтому скорость их выполнения определяет производительность всей системы в целом. Для обеспечения высокой скорости работы ОС все модули ядра или большая их часть постоянно находятся в оперативной памяти, то есть являются резидентными.

В состав ядра входят функции, решающие внутрисистемные задачи организации вычислительного процесса, такие как переключение контекстов, загрузка/выгрузка страниц, обработка прерываний. Эти функции недоступны для приложений. В ядре также могут содержаться функции для поддержки приложений, создающие так называемую прикладную программную среду. Приложения могут обращаться к ядру с запросами (системными вызовами) для выполнения тех или иных действий, например для открытия и чтения файла, вывода графической информации на дисплей, получения системного времени и т. д. Функции ядра, которые могут вызываться приложениями, образуют интерфейс прикладного программирования API.

Ядро является самым критичным компонентом в компьютерной системе — крах ядра равносилен краху всей системы. Поэтому разработчики операционной системы уделяют особое внимание надёжности кодов ядра.

Обычно ядро оформляется в виде программного модуля некоторого специального формата, отличающегося от формата пользовательских приложений.

Основные функции ядра:
– управление процессами;
– организация взаимодействий между процессами;
– синхронизация процессов;
– планирование (диспетчирование) работы процессов;
– поддержка операций ввода-вывода;
– поддержка распределения и перераспределения памяти;
– поддержка функций по ведению статистики работы машины.

12. Понятия процесса и потока

Процесс — одно из фундаментальных понятий в любой ОС. К сожалению, однозначного определения этого термина не существует до сих пор. В большинстве случаев достаточно считать, что процесс — это единица работы системы, которая описывается в системе в виде специальной структуры, часто называемой дескриптором процесса, и которой распределяются системные ресурсы.

Программный код только тогда начнёт 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.

В современных ОС в рамках одного процесса может выполняться несколько потоков.

Поток — диспетчеризуемая единица работы, включающая контекст процессора (куда входит содержимое программного счётчика и указателя вершины стека), а также свою собственную область стека (для организации вызова подпрограмм и хранения локальных данных). Команды потока выполняются последовательно; поток может быть прерван при переключении процессора на обработку другого потока.

Важно понимать, что все потоки в рамках одного процесса используют общие ресурсы системы, выделенные данному процессу. Можно даже сказать, что процессы конкурируют за все ресурсы системы, кроме процессорного времени, в то время как потоки конкурируют только за процессорное время.

В многозадачной (многопроцессной) системе процесс (или поток, если процесс состоит из нескольких потоков) может находиться в одном из трёх основных состояний (рис. 12.1):

АКТИВЕН — процесс (поток) обладает всеми необходимыми ресурсами и непосредственно выполняется процессором;

БЛОКИРОВАН — процесс не может выполняться по своим внутренним причинам, он ждёт осуществления некоторого события, например, завершения операции ввода-вывода, получения сообщения от другого процесса, освобождения какого-либо необходимого ему ресурса;

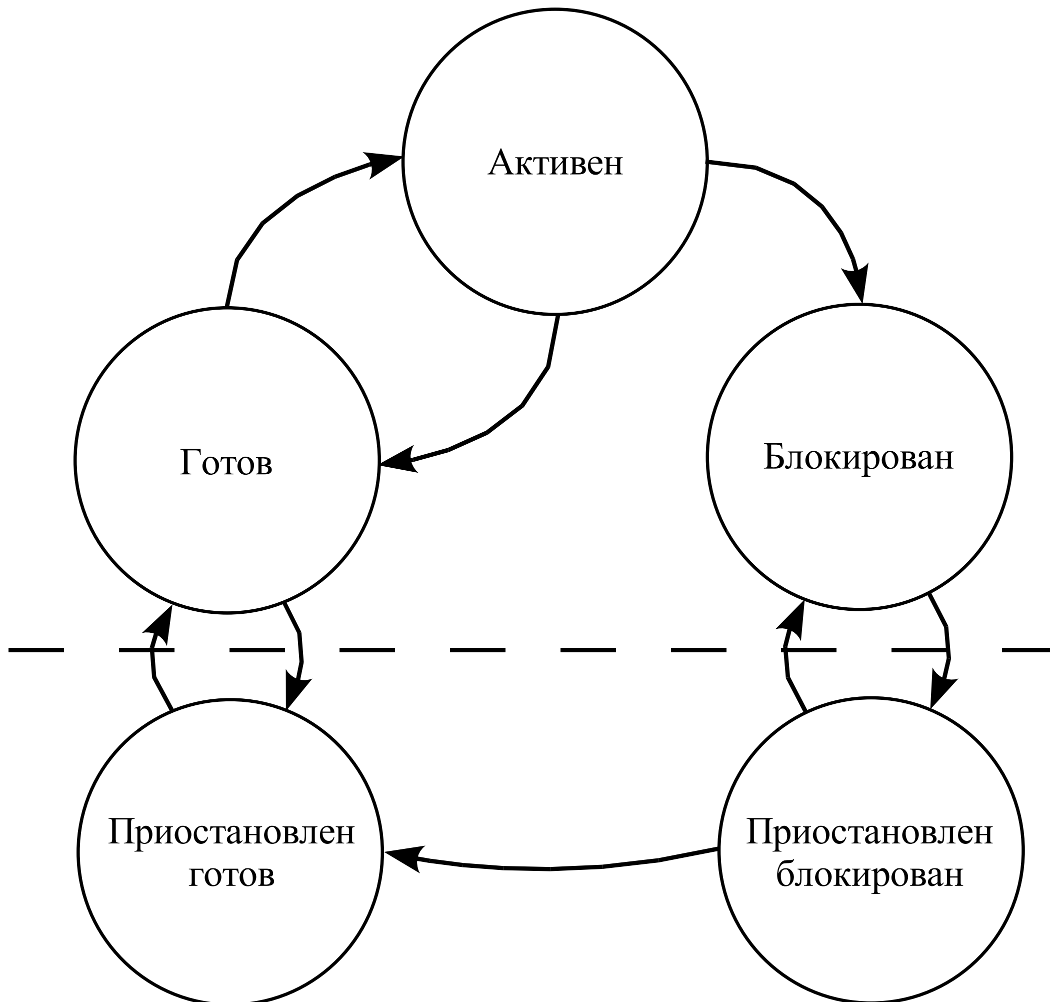

Рис. 12.1. Граф состояния процессов

ГОТОВ — процесс (поток) обладает всеми необходимыми ресурсами для продолжения работы и ожидает освобождения процессора, который занят выполнением другого процесса (потока).

В ходе жизненного цикла каждый процесс (поток) переходит из одного состояния в другое в соответствии с алгоритмом планирования, реализуемым в данной операционной системе.

В активном состоянии в однопроцессорной системе в каждый момент времени может находиться только один процесс (поток).

В состояниях блокировки и готовности в системе любого типа может находиться сразу несколько процессов (потоков). При этом в состоянии готовности ожидающие процессы (потоки) образуют очередь. В состоянии блокировки процессы (потоки) обычно не упорядочиваются, так как каждый из них ожидает наступления своего события.

В некоторых системах для временного уменьшения нагрузки предусмотрена возможность приостановки отдельных процессов.

В этом случае появляется два дополнительных состояния, в которых могут оказываться процессы:

ПРИОСТАНОВЛЕН БЛОКИРОВАН — приостанавливается процесс, находившийся в состоянии блокировки;

ПРИОСТАНОВЛЕН ГОТОВ — приостанавливается процесс, находившийся в состоянии готовности.

Приостановить можно только процесс целиком со всеми его потоками, так как эта процедура обычно связана с попыткой временно изъять у процесса часть распределённых ему ресурсов системы, чтобы передать их другим работающим в системе процессам.

13. Планирование процессов как функция ядра операционной системы

Планирование процессов на уровне ядра ОС 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.

Существует множество различных алгоритмов планирования процессов, по разному решающих вышеперечисленные задачи, преследующих различные цели и обеспечивающих различное качество мультипрограммирования. Обычно всё множество этих алгоритмов можно классифицировать по нескольким признакам.

1. Алгоритмы с применением прерываний процессов и без них. (Иногда применяются термины «вытесняющая многозадачность» и «невытесняющая многозадачность».)

При использовании прерываний всё время процессора делится на кванты. Каждому процессу в очереди предоставляется свой квант процессорного времени. По истечение каждого кванта времени генерируется прерывание, при обработке которого ОС переводит текущий активный процесс в очередь готовых, а на его место выбирает очередной процесс из очереди готовых к выполнению.

Процесс может и сам освободить процессор, если в течение своего кванта времени он будет заблокирован ожиданием какого-либо события, например, окончания операции ввода-вывода.

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 пришёл — первый обслужен» (FIFO) или по какому-либо другому принципу.

Такой порядок планирования применяется в системах разделения времени, ориентированных на интерактивное взаимодействие с пользователями.

В случае непрерываемого планирования каждый процесс, получивший управление, работает на процессоре до тех пор, пока сам его не освободит. Процессор освобождается если:

– процесс завершился;

– процесс перешёл в состояние блокировки;

– процесс самостоятельно передал управление другим процессам.

Такие алгоритмы планирования наиболее эффективны в пакетных системах обработки.

2. Алгоритмы с использованием приоритетов и без них.

При приоритетном планировании в момент выбора на исполнение нового процесса учитываются приоритеты, присвоенные процессам в очереди. Приоритеты могут быть статические (не изменяющиеся во времени) и динамические (накапливаемые в течение времени, пока процесс ожидает в очереди, либо изменяемые системой в зависимости от поведения процесса), относительные (активный процесс не может быть прерван в течение своего кванта времени при появлении в очереди более приоритетного процесса) и абсолютными (в противном случае).

При бесприоритетном планировании очередной процесс выбирается из очереди в соответствии со стратегией, учитывающей равенство процессов. Это могут быть: случайная выборка, выборка по принципу FIFO и т. п.

Во многих ОС планировщики построены как комбинация описанных выше принципов планирования.

14. Понятие ресурса. Оперативно перераспределяемые и оперативно неперераспределяемые ресурсы

К числу основных ресурсов современных ЭВМ могут быть отнесены:

– процессоры;

– оперативная память;

– таймеры;

– внешняя память (в том числе диски, ленты и накопители других типов);

– файлы и другие наборы данных;

– программы;

– принтеры и другие устройства документального вывода;

– сетевые контроллеры;

– другие устройства.

Все ресурсы, кроме процессоров, распределяются между процессами. Процессорное время распределяется между потоками. За распределение ресурсов отвечает ОС.

Ресурсы, которые можно временно изъять у процесса (потока) без больших накладных затрат для передачи другим процессам, называются оперативно перераспределяемыми. Такими ресурсами могут быть процессоры, оперативная помять.

Ресурсы, которые нельзя или очень сложно изъять у процесса до завершения его работы (или до тех пор, пока он сам их не освободит), называются оперативно неперераспределяемыми. К таким ресурсам относятся, например, магнитные ленты и принтеры.

Некоторые виды ресурсов (например, диски) можно разделять между несколькими процессами (каждый процесс читает или записывает свой файл на одном диске).

Программы (процессы) сами могут выступать как ресурсы системы. Например, нескольким пользователям может потребоваться программа текстового редактора. Хорошо организованная ОС предоставит всем пользователям единственную копию редактора в памяти, обеспечив каждому пользователю свой блок редактируемых данных.

15. Распределение ресурсов и управление ресурсами как функция ОС

Операционная система не только предоставляет пользователям и программистам удобный интерфейс к аппаратным средствам компьютера, но и является механизмом, распределяющим ресурсы компьютера. К числу основных ресурсов современных вычислительных систем могут быть отнесены процессоры, основная память, таймеры, наборы данных, диски, накопители на магнитных лентах, принтеры, сетевые устройства и некоторые другие. Ресурсы распределяются между процессами (кроме процессоров, которые распределяются между потоками).

Управление ресурсами вычислительной системы с целью наиболее эффективного их использования является назначением операционной системы. Например, мультипрограммная операционная система организует одновременное выполнение сразу нескольких процессов на одном компьютере, поочерёдно переключая процессор с одного процесса на другой, исключая простои процессора, вызываемые обращениями процессов к вводу-выводу. ОС также отслеживает и разрешает конфликты, возникающие при обращении нескольких процессов к одному и тому же устройству ввода-вывода или к одним и тем же данным.

Критерий эффективности, в соответствии с которым ОС организует управление ресурсами компьютера, может быть различным. Например, в одних системах важен такой критерий, как пропускная способность вычислительной системы, в других — время её реакции. Соответственно выбранному критерию эффективности операционные системы по-разному организуют вычислительный процесс.

Управление ресурсами включает решение следующих общих, не зависящих от ресурса задач:

– планирование ресурса, то есть определение, какому процессу, когда и в каком количестве (если ресурс может выделяться частями) следует выделить ресурс;

– удовлетворение запросов на ресурсы;

– отслеживание состояния и учёт использования ресурса, то есть поддержание оперативной информации о том, занят или свободен ресурс и какая доля ресурса уже распределена;

– разрешение конфликтов между процессами.

Задача организации эффективного совместного использования ресурсов несколькими процессами является весьма сложной, и сложность эта порождается в основном случайным характером возникновения запросов на потребление ресурсов. В мультипрограммной

системе образуются очереди заявок от одновременно выполняемых программ к разделяемым ресурсам компьютера: процессору, страницам памяти, к принтерам, к дискам. Операционная система организует обслуживание этих очередей по разным алгоритмам: в порядке поступления, на основе приоритетов, кругового обслуживания и т. д.

16. Понятие взаимоисключения нескольких процессов и критические участки

При работе вычислительной системы в режиме мультипрограммирования (многозадачности) возможны ситуации, когда:

– несколько процессов обмениваются данными посредством обращения к общим областям памяти (сотрудничают). При этом необходимо, чтобы в каждый момент времени модифицировать разделяемую область памяти мог только один из сотрудничающих процессов;

– несколько независимо работающих процессов время от времени должны получать в монопольное владение какой-либо ресурс системы.

Любая из названных ситуаций приводит к необходимости взаимоисключения, то есть процесс, получивший право модифицировать разделяемую область памяти или использовать ресурс системы, исключает эту возможность для других процессов на время, необходимое ему для выполнения своих задач.

Взаимоисключение обеспечивается включением в код программы специальных критических участков (или критических секций). Каждый критический участок отвечает за свой разделяемый ресурс. Процесс может войти в свой критический участок только в том случае, если ни один другой процесс не вошёл в такой же свой критический участок. В противном случае процесс должен ждать выхода другого процесса из своего критического участка.

Основные требования, выдвигаемые к разработке кодов критических участков:

– в любой момент времени только один процесс может находиться в своей критической секции. Возможно, что в какой-то момент времени ни один процесс не будет находиться в критической секции;

– ни один процесс не должен бесконечно находиться в своей критической секции;

– ни один процесс не должен бесконечно долго ожидать разрешения на вход в свою критическую секцию;

– процесс, находящийся вне своей критической секции не должен мешать другим процессам входить в свои критические секции;

– если два или более процесса хотят войти в свои критические секции, то решение вопроса о том, кто из них первым войдёт в критическую секцию, не должно быть бесконечно долгим;

– если какой-либо процесс, участвующий во взаимоисключении, завершается, это не должно влиять на возможность других процессов входить в свои критические секции.

Режим взаимоисключения может быть реализован различными средствами.

Прежде всего, это могут быть программные и программно-аппаратные методы.

В первом случае режим взаимоисключения реализуется полностью программным образом без поддержки специфических аппаратных компонентов ЭВМ. В этом смысле такие реализации являются лучше переносимыми между различными архитектурами.

Во втором случае для реализации взаимоисключения используются специальные неделимые команды конкретных ЭВМ, позволяющие единым актом проверить значение ячейки памяти и установить в ней новое значение.

Среди программных методов выделяют методы:

– основанные на программном коде, встраиваемом в сами прикладные процессы (алгоритмы Деккера и Петерсона);

– основанные на семафорах;

– основанные на мониторах.

17. Алгоритмы взаимоисключения Деккера и Петерсона

Оба алгоритма относятся к алгоритмам взаимоисключения, построенным по чисто программному принципу, причём ответственность за взаимоисключение в обоих случаях возложена на сами процессы. При этом каждый процесс должен координировать свои действия с другими процессами, пытающимися войти в критические секции.

Оба алгоритма рассчитаны на использование в случае взаимоисключения двух процессов. Для более, чем двух процессов они не применяются.

Основные условия, которым удовлетворяют оба алгоритма:

– задача взаимоисключения решается чисто программным способом;

– не делается никаких предположений об относительных скоростях выполнения процессов;

– процессы, находящиеся вне своих критических участков, не могут мешать другим процессам входить в их собственные критические участки;

– не должно быть бесконечного откладывания момента входа процессов в их критические участки.

Алгоритм Деккера был предложен в середине 1960 годов.

```
Листинг алгоритма Деккера

boolean flag0, flag1;
int turn;

void P0()
{
  while(true){
    flag0=true;
    while(flag1){
      if(turn==1){
        flag0=false;
        while(turn==1)
          /* Ничего не делать */;
        flag0=true;
      }
    }

/* Критический участок */;

    turn=1;
    flag0=false;

/* Остальной код */;

  }
}


void P1()
{
  while(true){
    flag1=true;
```

```
    while(flag0){
      if(turn==0){
        flag1=false;
        while(turn==0)
         /* Ничего не делать */;
        flag1=true;
      }
    }

/* Критический участок */;

    turn=0;
    flag1=false;

/* Остальной код */;

  }
}

void main()
{
  flag0=false;
  flag1=false;
  turn=1;
  parbegin(P0,P1);
}
```

Алгоритм Деккера решает задачу взаимных исключений, но достаточно сложным путём, корректность которого не так легко доказать. Петерсон предложил в 1981 г. простое и элегантное решение той же проблемы.

```
Листинг аглоритма Петерсона

boolean flag[2];
int turn;

void P0()
{
  while(true){
    flag[0]=true;
```

```
    turn=1;
    while(flag[1] && turn==1)
      /* Ничего не делать */;

/* Критический участок */;

    flag[0]=false;

/* Остальной код */;

  }
}

void P1()
{
  while(true){
    flag[1]=true;
    turn=0;
    while(flag[0] && turn==0)
      /* Ничего не делать */;

/* Критический участок */;

    flag[1]=false;

/* Остальной код */;

  }
}

void main()
{
  flag[0]=false;
  flag[1]=false;
  parbegin(P0,P1);
}
```

18. Семафоры и мьютексы

Семафор — специальная защищённая переменная в ОС. Под защищённостью понимается невозможность для прикладной программы работать с этой переменной непосредственно. Для доступа к семафору программа должна использовать специальные системные вызовы ядра (примитивы):

– установить начальное значение семафора S;

– P(S) — если S>0, то S:=S−1, иначе ожидать на S;

– V(S) — если один или более процессов ожидают на S, то позволить одному из них продолжить работу, иначе S:=S+1.

Обратите внимание, что вызов примитива P(S) может заблокировать процесс до выполнения примитива V(S) другим процессом.

Все вызовы ядра для управления семафорами атомарны, то есть не могут быть прерваны другими процессами.

Семафоры, управляемые описанными примитивами, называются считающими, так как в принципе могут принимать любые неотрицательные значения. В частности, такие семафоры могут использоваться для управления несколькими однотипными ресурсами в системе.

Семафоры, которые могут принимать только два значения: 0 и 1, — называются мьютексами. Такие семафоры чаще всего применяются для организации процедуры взаимоисключения.

В современных ОС реализуется несколько разных механизмов управления семафорами, модифицирующих базовые алгоритмы, описанные выше.

19. Реализация взаимоисключения на семафорах

Рассмотрим только самый простой случай взаимоисключения, когда нескольким процессам необходимо время от времени входить в критическую секцию. Подчеркнём, что в отличие от алгоритмов Деккера и Петерсона с использованием семафоров достаточно легко реализовать взаимоисключение произвольного количества процессов. Приведённый ниже листинг описывает функции входа в критическую секцию (enter) и выхода из критической секции (leave), которые могут быть применены в любом из процессов, конкурирующих за вход в критическую секцию.

```
void enter()
{
  P(S);
}

void leave(){
  V(S);
}

main()
{
  /* установить семафору S значение 1 */
  parbegin(список процессов);
}
```

20. Мониторы ресурсов и реализация взаимоисключения на мониторах

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.

Соблюдение условия выполнения только одного процесса в определённый момент времени позволяет монитору обеспечить взаимоисключения. Данные монитора доступны в этот момент только одному процессу, следовательно, защитить совместно используемые структуры данных можно, просто поместив их в монитор. Если данные в мониторе представляют некий ресурс, то монитор обеспечивает взаимоисключение при обращении к ресурсу.

На основании данного определения монитора легко реализовать функции входа в критическую секцию (enter) и выхода из критической секции (leave) для произвольного числа конкурирующих процессов.

```
monitor
{
  boolean flag=false;  /* флаг занятости ресурса */
  queue S;  /* очередь ожидания */

  enter()
  {
    if(flag==true)
        wait(S);  /* ждать */
    flag=true;
  }

  leave()
  {
    flag=false;
    signal(S);  /* разбудить ожидающие процессы */
  }
}
```

Как можно видеть, для эффективной реализации монитора необходимы системные примитивы wait() и signal(), позволяющие организовать ожидание процессом наступления события «ресурс свободен» и возобновление процесса при наступлении этого события. Причём, примитив wait() выводит процесс из монитора, и для продолжения работы процесс должен снова пытаться войти в монитор (в функцию enter()). Этим обеспечивается возможность взаимоисключения для любого количества процессов.

21. Реализация взаимоисключения на аппаратном уровне

При реализации взаимоисключения на аппаратном уровне используются специальные команды ЭВМ, позволяющие единым актом (без прерываний) выполнить проверку значения заданной переменной (ячейки памяти) и её установку в заданное значение. Этого оказывается достаточно, чтобы гарантировать простой механизм взаимоисключения для произвольного количества процессов.

В качестве примера укажем на инструкцию процессора Intel 8086 XCHG. Инструкция позволяет обменять значения двух своих операндов. Механизм взаимоисключения с применением этой инструкции может базироваться на следующей последовательности команд.

```
      mov  ax,1
wait:
      xchg ax,flag
      test ax
      jnz  wait

; критическая секция

      mov flag,0
```

Взаимоисключение реализуется с использованием переменной-флага flag. В начальный момент времени переменная flag равна 0, то есть критическая секция свободна.

Для входа в критическую секцию процесс выполняет предварительную запись признака занятости критической секции (значение 1) в регистр ax, после чего обменивает значение регистра ax и переменной flag. Поскольку обмен выполняется одной непрерываемой командой, ни один другой процесс не может повлиять на результат обмена.

Теперь достаточно проверить значение регистра ax. Если оно равно 0, то критическая секция была свободна, и наш процесс уже её занял, записав 1 в переменную flag. Процесс может войти в критическую секцию.

Если же значение регистра ax равно 1, то критическая секция уже занята, и наш процесс должен ждать её освобождения. При этом наша операция обмена не испортила значение переменной flag.

Преимуществом такого подхода является простота реализации. Кроме того, при использовании нескольких переменных возможно простое управление несколькими критическими секциями.

Но у этого подхода имеются и недостатки:

– непереносимость;

– ожидание разрешения входа в критическую секцию выполняется в активном цикле, то есть перерасходуется ресурс процессорного времени;

– возможна ситуация бесконечного откладывания, так как не существует механизма упорядочивания процессов, ожидающих входа в критическую секцию;

– возможна ситуация голодания, когда вошедший в критическую секцию менее приоритетный процесс будет ожидать, пока выполняется более приоритетный процесс, находящийся в цикле активного ожидания права войти в критическую секцию.

22. Тупики и методы борьбы с ними

Выполняющийся процесс оказывается в тупике, если он ожидает наступления события, которое никогда не произойдёт.

Причины возникновения тупиков многообразны. Решения проблемы тупиков в общем случае не существует.

Классическим (но далеко не единственным) примером тупика является следующий. Предположим, что процессам А и Б, работающим в системе, для выполнения своей работы необходимы ресурсы Р1 и Р2, причём каждый из этих ресурсов не может разделяться между процессами и, кроме того, является оперативно неперераспределяемым.

Пусть процесс А запросил и получил в своё распоряжение ресурс Р1. После этого закончился его квант времени, и управление получил процесс Б.

Процесс Б запросил и получил в своё распоряжение ресурс Р2. После этого процесс Б запрашивает ресурс Р1, но не может его получить, так как ресурс уже занят процессом А, и должен ожидать освобождения ресурса Р1.

Получив управление, процесс А запрашивает ресурс Р2, но не может его получить, так как ресурс уже занят процессом Б.

Оба процесса будут ожидать освобождения необходимых им ресурсов, но это событие никогда не наступит.

Для того, чтобы в системе возник тупик, необходимо выполнение следующих условий.

1) Условие взаимоисключения. Одновременно использовать ресурс может только один процесс.

2) Условие ожидания ресурсов. Процессы удерживают ресурсы, уже выделенные им, и могут запрашивать другие ресурсы.

3) Условие неперераспределяемости. Ресурс, выделенный ранее, не может быть принудительно отобран у процесса. Он может быть освобождён только процессом, который его удерживает.

4) Условие кругового ожидания. Существует кольцевая цепь процессов, в которой каждый процесс ждёт доступа к ресурсу, удерживаемому другим процессом цепи.

Тупик возникает в системе только при одновременном выполнении всех четырёх условий.

Соответственно, методы борьбы с тупиками направлены на устранение хотя бы одного из перечисленных условий.

Рассматриваются следующие методы:

– игнорирование проблемы в целом;
– предотвращение тупиков;
– обход тупиков;
– обнаружение тупиков;
– восстановление после тупиков.

Метод игнорирования проблемы возникновения тупиков основан на наблюдаемом свойстве многих систем, заключающемся в том, что возникновение тупиков — не такое уж частое событие. Игнорирование проблемы возникновения тупиков позволяет создать более производительную, более удобную и комфортную для программистов и пользователей систему.

Однако не во всех системах эту проблему можно игнорировать. Например, возникновение тупиков недопустимо в системах реального времени.

23. Методы предотвращения тупиков

Предотвращение тупиков предполагает исключение самой возможности возникновения тупиков в системе. Предотвращение тупиков возможно только за счёт нарушения условий возникновения тупиков.

Нарушение условия взаимоисключения

В общем случае избежать взаимоисключений невозможно. Доступ к некоторым ресурсам должен быть исключительным. Тем не менее некоторые устройства удаётся сделать разделяемыми. В качестве примера рассмотрим принтер. Известно, что пытаться осуществлять вывод на принтер могут несколько процессов. Для предоставления им такой возможности в современных системах используют механизм спулинга — данные для принтера от каждого процесса помещаются в буферные файлы, а собственно вывод на принтер этих файлов выполняет специальный системный процесс.

К сожалению, не для всех устройств и не для всех данных можно организовать спулинг. Неприятным побочным следствием такой модели может быть потенциальная тупиковая ситуация из-за конкуренции за дисковое пространство для буфера спулинга. Тем не менее в той или иной форме эта идея применяется часто.

Нарушение условия ожидания дополнительных ресурсов

В этом случае система требует, чтобы каждый процесс запрашивал все требуемые ему ресурсы сразу, перед началом выполнения.

Если какой-либо ресурс не может быть предоставлен, процесс просто не может начать выполняться, и должен ждать освобождения всех необходимых ему ресурсов, не удерживая при этом за собой никаких ресурсов.

Этот метод требует от программиста (пользователя) точного знания о том, какие ресурсы потребуются процессу во время выполнения. А эта информация не всегда доступна до начала выполнения процесса.

Кроме того, применение этого метода ведёт к непроизводительному расходованию ресурсов, так как не все ресурсы, выделенные процессу в начале работы, будут нужны ему в течение всего времени работы.

Нарушение принципа отсутствия перераспределения

Если бы можно было отбирать ресурсы у удерживающих их процессов до завершения этих процессов, то удалось бы добиться невыполнения третьего условия возникновения тупиков. Перечислим минусы данного подхода.

Во-первых, отбирать у процессов можно только те ресурсы, состояние которых легко сохранить, а позже восстановить, например состояние процессора.

Во-вторых, если процесс в течение некоторого времени использует определённые ресурсы, а затем освобождает эти ресурсы, он может потерять результаты работы, проделанной до настоящего момента.

Наконец, следствием данной схемы может быть дискриминация отдельных процессов, у которых постоянно отбирают ресурсы.

Весь вопрос в цене подобного решения, которая может быть слишком высокой, если необходимость отбирать ресурсы возникает часто.

Нарушение условия кругового ожидания

Можно предложить упорядочить все ресурсы в системе и потребовать, чтобы каждый процесс запрашивал ресурсы в строго определённом порядке.

На практике эта схема трудно реализуема и ведёт к неэффективности использования ресурсов и существенному усложнению программирования.

Реально такая схема может быть применена лишь к ограниченному классу ресурсов.

24. Методы обхода тупиков. Алгоритм банкира

Данный метод налагает на процессы в системе меньшие ограничения, чем метод предотвращения тупиков, позволяя системе работать с большей эффективностью.

Возникновение тупиков в системе потенциально возможно, но используются специальные алгоритмы распределения ресурсов, которые позволяют обойти ситуации, грозящие возникновением тупиков.

Среди такого рода алгоритмов наиболее известен алгоритм банкира, предложенный Дейкстрой, который базируется на так называемых безопасных или надёжных состояниях. Безопасное состояние — это такое состояние, для которого имеется по крайней мере одна последовательность событий, которая не приведёт к тупику. Модель

алгоритма основана на действиях банкира, который, имея в наличии капитал, выдаёт кредиты.

Суть алгоритма состоит в следующем.

Предположим, что у системы в наличии n свободных устройств, например магнитных лент.

ОС принимает запрос от пользовательского процесса, если его максимальная потребность не превышает n.

Пользователь гарантирует, что если ОС в состоянии удовлетворить его запрос, то все устройства будут возвращены системе в течение конечного времени.

Текущее состояние системы называется надёжным, если ОС может обеспечить всем процессам их завершение (выделить необходимые ресурсы) в течение конечного времени.

В соответствии с алгоритмом банкира выделение устройств возможно, только если состояние системы остаётся надёжным.

Рассмотрим надёжные и ненадёжные состояния на примере. Предположим, что в системе имеется 10 ленточных устройств и работает 3 процесса. Для завершения работы Процессу 1 потребуется 5 устройств, Процессу 2 — 9 устройств, Процессу 3 — 3 устройства. В какой-то момент времени распределение устройств процессам выглядит так, как показано в таблице.

Процессы	Выделено ресурсов	Максимальная потребность
Процесс 1	2	5
Процесс 2	5	9
Процесс 3	1	3
Свободных ресурсов	2	–

Состояние системы, представленное в таблице, является надёжным, поскольку существует такая последовательность выделения ресурсов процессам, что с течением времени все потребности процессов в ресурсах будут удовлетворены, и все процессы смогут завершиться в конечное время. Для этого сперва необходимо удовлетворить потребности Процесса 3, затем (после завершения Процесса 3), Процесса 1, и лишь затем Процесса 2. При таком порядке удовлетворения запросов на ресурсы система будет проходить только через надёжные состояния.

Если же распределять оставшиеся ресурсы в другом порядке, то система перейдёт в ненадёжное состояние.

Термин ненадёжное состояние не предполагает, что обязательно возникнут тупики. Он лишь говорит о том, что в случае неблагоприятной последовательности событий система может зайти в тупик.

Данный алгоритм обладает тем достоинством, что при его использовании нет необходимости в перераспределении ресурсов и откате процессов назад. Однако использование этого метода требует выполнения ряда условий.

1) Число процессов и число ресурсов должно быть фиксировано.

2) Число работающих процессов не должно не увеличиваться. И не должно появляться новых процессов.

3) Алгоритм требует, чтобы клиенты гарантированно возвращали ресурсы.

4) Должны быть заранее указаны максимальные требования процессов к ресурсам. Чаще всего данная информация отсутствует.

Кроме того, во многих случаях гарантия завершения любого процесса в течение конечного времени является недостаточной. Требуется более точная и предсказуемая оценка времени завершения.

25. Методы обнаружения тупиков

Обнаружение тупика — это установление факта, что возникла тупиковая ситуация, и определение процессов и ресурсов, вовлечённых в эту тупиковую ситуацию. Алгоритмы обнаружения тупиков, как правило, применяются в системах, где выполняются первые три необходимых условия возникновения тупиковой ситуации. Эти алгоритмы затем определяют, не создался ли режим кругового ожидания.

Применение алгоритмов обнаружения тупиков сопряжено с определёнными дополнительными затратами машинного времени. Таким образом, здесь мы снова сталкиваемся с необходимостью прибегать к компромиссным решениям, столь распространённым в операционных системах. Необходимо решить, будут ли накладные расходы, связанные с реализацией алгоритмов обнаружения тупиковых ситуаций, в достаточной степени оправданы потенциальными выгодами от локализации и устранения тупиков.

При рассмотрении задачи обнаружения тупиков применяется весьма распространённая нотация, согласно которой распределение ресурсов и запросы изображаются в виде направленного графа. Квадраты обозначают процессы, а большие кружки — классы идентичных ресурсов. Малые кружки, находящиеся внутри больших, обозначают количество идентичных ресурсов каждого класса.

Стрелка от квадрата к большому кружку показывает запрос процессом соответствующего ресурса. Стрелка от маленького кружка к квадрату показывает, что единица ресурса выделена процессу (процесс удерживает за собой эту единицу ресурса).

Графы запросов и распределения ресурсов динамично меняются по мере того, как процессы запрашивают ресурсы, получают их в своё распоряжение, а через какое-то время возвращают операционной системе.

Задача механизма обнаружения тупиков — определить, не возникла ли в системе тупиковая ситуация. Одним из способов обнаружения тупиков является приведение, или редукция графа, — это позволяет определять процессы, которые могут завершиться, и процессы, которые будут оставаться в тупиковой ситуации.

Если запросы ресурсов для некоторого процесса могут быть удовлетворены, то мы говорим, что граф можно редуцировать на этот процесс. Такая редукция эквивалентна изображению графа в том виде, который он будет иметь в случае, если данный процесс завершит свою работу и возвратит ресурсы системе. Редукция графа на конкретный процесс изображается исключением стрелок, идущих к этому процессу от ресурсов (т. е. ресурсов, выделенных данному процессу) и стрелок к ресурсам от этого процесса (т. е. текущих запросов данного процесса на выделение ресурсов). Если граф можно редуцировать на все процессы, значит, тупиковой ситуации нет, а если этого сделать нельзя, то все нередуцируемые процессы образуют набор процессов, вовлечённых в тупиковую ситуацию.

Здесь важно отметить, что порядок, в котором осуществляется редукция графа, не имеет значения: окончательный результат в любом случае будет тем же самым.

На рис. 25.1 показан граф, который может быть полностью редуцирован (рис. 25.2), что говорит о том, что в данном случае тупиковой ситуации в системе нет.

На рис. 25.3 показан типичный нередуцируемый граф.

26. Методы восстановления после тупиков

Обнаружив тупик, можно вывести из него систему, нарушив одно из условий существования тупика. При этом, возможно, несколько процессов частично или полностью потеряют результаты проделанной работы.

Сложность восстановления обусловлена рядом факторов.

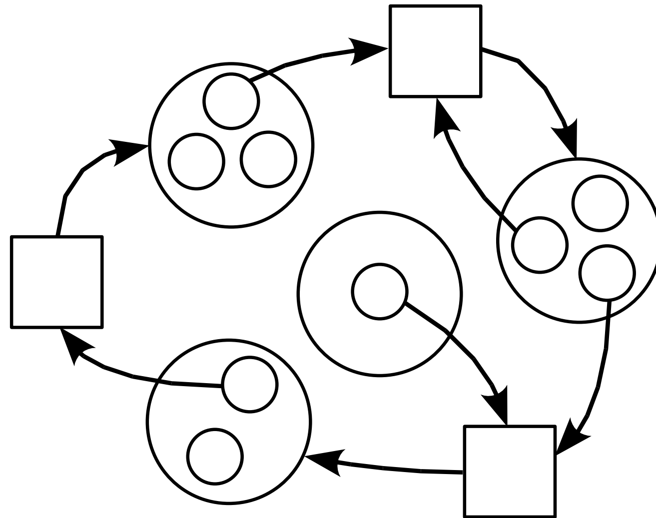

Рис. 25.1. Исходный граф

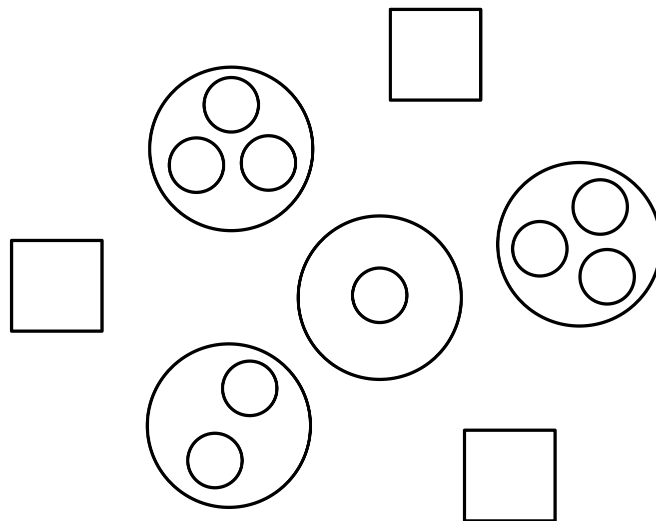

Рис. 25.2. Редуцированный граф

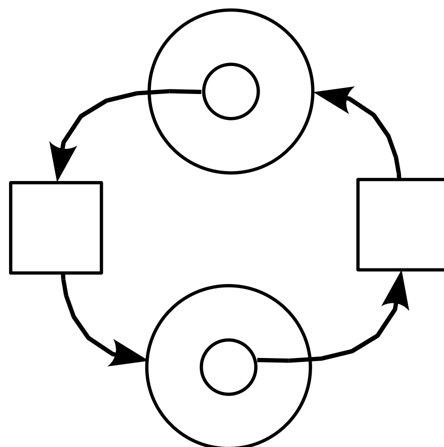

Рис. 25.3. Нередуцируемый граф

В большинстве систем нет достаточно эффективных средств, чтобы приостановить процесс, вывести его из системы и возобновить впоследствии с того места, где он был остановлен.

Если даже такие средства есть, то их использование требует затрат и внимания оператора.

Восстановление после тупика может потребовать значительных усилий.

Самый простой и наиболее распространённый способ устранить тупик — завершить выполнение одного или более процессов, чтобы впоследствии использовать его ресурсы. Тогда в случае удачи остальные процессы смогут выполняться. Если это не помогает, можно ликвидировать ещё несколько процессов. После каждой ликвидации должен запускаться алгоритм обнаружения тупика.

По возможности лучше ликвидировать тот процесс, который может быть без ущерба возвращён к началу (такие процессы называются идемпотентными). Примером такого процесса может служить компиляция. С другой стороны, процесс, который изменяет содержимое базы данных, не всегда может быть корректно запущен повторно.

В некоторых случаях можно временно забрать ресурс у текущего владельца и передать его другому процессу. Возможность забрать ресурс у процесса, дать его другому процессу и затем без ущерба вернуть назад сильно зависит от природы ресурса. Подобное восстановление часто затруднительно, если не невозможно.

В ряде систем реализованы средства отката и перезапуска или рестарта с контрольной точки (сохранение состояния системы в какой-то момент времени). Если проектировщики системы знают, что тупик вероятен, они могут периодически организовывать для процессов контрольные точки. Иногда это приходится делать разработчикам прикладных программ.

Когда тупик обнаружен, видно, какие ресурсы вовлечены в цикл кругового ожидания. Чтобы осуществить восстановление, процесс, который владеет таким ресурсом, должен быть отброшен к моменту времени, предшествующему его запросу на этот ресурс.

Средства рестарта с контрольной точки, реализованные во многих системах, обеспечивают приостановку/возобновление вычислений с потерей результатов только после последней контрольной точки (от момента, когда запоминалось состояние системы). Однако в конструкции многих систем такой достаточно эффективный механизм контрольных точек с рестартом не предусмотрен. Поэтому разработчикам прикладных программ обычно приходится тратить усилия

на включение контрольных точек с возможностью рестарта в свою программу, так что если речь не идёт о программах, выполнение которых требует много часов машинного времени, подобный механизм применяется редко.

27. Методы управления оперативной памятью

Все исторически возникшие и существующие на сегодня методы управления оперативной памятью можно свести к следующему списку.

1) Распределение всей ОП ЭВМ одному процессу. Используется только в однозадачных ЭВМ.

2) Разделение ОП ЭВМ на фиксированные разделы. При этом в момент запуска ОС память ЭВМ разделяется на несколько разделов заданного (не обязательно одинакового) размера. В каждый раздел может быть загружен только один процесс подходящего размера. По завершении процесса раздел освобождается, и в него можно загрузить следующий процесс.

В зависимости от способа подготовки программ (наличия механизма перемещения), могут быть сформированы очереди задач для каждого раздела или одна общая очередь для всех разделов.

3) Использование разделов переменного размера. В этом случае в момент старта ОС память ЭВМ представляется единым разделом. ОС загружает процессы, выделяя для них разделы точно по требуемому размеру. В результате ОП используется более эффективно, чем в случае её деления на фиксированные разделы. Но по мере работы системы (когда некоторые процессы завершают работу и освобождают память) возникает фрагментация памяти.

Фрагментация порождает несколько проблем. В частности, проблему поиска свободного участка памяти для загрузки нового процесса, проблему объединения соседних освободившихся участков памяти, проблему дефрагментации памяти и др.

4) Использование малых разделов фиксированного размера (страниц). При этом память делится на страницы небольшого и равного размера. Для каждого процесса выделяется непрерывная последовательность страниц, достаточная по суммарному объёму для размещения процесса.

Достоинство метода — возможность контролировать занятость страниц памяти с помощью простейшего механизма — битовой маски.

5) Использование метода двойников. При вся ОП системы делится пополам, образуя двух двойников. Каждый из них снова делится пополам и так далее, пока не образуются разделы, подходящие для размещения процессов. При необходимости загрузить процесс, требующий больше памяти, необходимо объединить несколько двойников в один блок большего размера. Объединяться могут только двойники.

Описанные выше методы предполагают управление физической ОП без привлечения дополнительных аппаратных средств.

Распространённые в последнее время методы управления виртуальной памятью не отменяют необходимости управлять на уровне ОС физической ОП.

28. Стратегии поиска подходящего блока оперативной памяти

Стратегии обычно используются при управлении памятью методами с переменными размерами разделов. При этом в процессе работы по мере завершения процессов в памяти образуются свободные фрагменты — дыры. Новые процессы можно загрузить только в эти дыры, причём для загрузки процесса необходимо, чтобы размер дыры был не меньше размера процесса.

Существуют следующие стратегии поиска подходящей дыры:

– первый подходящий — просматриваются все дыры от начала памяти и выбирается первая дыра, размер которой не меньше размера загружаемого процесса;

– следующий подходящий — аналог предыдущего метода, но каждый новый поиск начинается не с начала памяти, а от точки окончания предыдущего поиска;

– наиболее подходящий — из всех подходящих дыр выбирается наиболее близкая по размеру (наименьшая) к загружаемому процессу;

– наименее подходящий — из всех подходящих дыр выбирается самая большая.

Каждая из стратегий имеет лишь эмпирическое (умозрительное, теоретически недоказуемое) обоснование своей эффективности. Практика показывает, что любая из стратегий может превосходить другие по эффективности лишь в случае определённой последовательности загружаемых процессов.

С точки зрения минимальных затрат системных ресурсов наиболее эффективной является стратегия «первый подходящий».

29. Понятие виртуального ресурса

Термин «виртуальный» в буквальном переводе означает «кажущийся». Обычно под виртуальным ресурсом понимают ресурс, который представляется потребителям свойствами, отличными от тех, которые он имеет на самом деле.

Примеры:

– виртуальный диск — специально выделенная область ОП, обращение к которой выполняется прикладными процессами с помощью специального драйвера не прямым обращением, а по запросам, содержащим номера головки, дорожки, сектора, которых у ОП, естественно, нет и быть не может;

– виртуальный терминал — программа, эмулирующая работу аппаратуры терминала, в частности, отрабатывающая управляющие ESC-последовательности и демонстрирующая результат, например, в графическом окне;

– виртуальная память — механизм реализации недостающего объёма ОП на внешних носителях информации, чаще всего на жёстких дисках.

Виртуализированы могут быть многие устройства ЭВМ.

30. Виртуальная память. Принцип организации и основной алгоритм функционирования

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 распространёнными реализациями виртуальной памяти является страничное, сегментное и странично-сегментное распределение памяти, а также свопинг.

Принцип организации.

1) Все обращения к памяти в рамках процесса представляют собой логические (виртуальные) адреса, которые динамически транслируются в физические адреса во время исполнения. Это означает, что процесс может быть выгружен на диск и вновь загружен в основную память, так что во время работы он может находиться в разных местах основной памяти.

2) Процесс может быть разбит на ряд частей (страниц или сегментов), которые не обязательно должны располагаться в основной памяти единым непрерывным блоком. Это обеспечивается за счёт динамической трансляции адресов и использования таблицы страниц или сегментов.

Если в системе выполняются указанные принципы, то наличие всех страниц или сегментов процесса в основной памяти одновременно не является обязательным.

При загрузке каждого нового процесса операционная система размещает в памяти только один или нескольких блоков, включая блок, содержащий начало программы. Часть процесса, располагающаяся в некоторый момент времени в основной памяти, называется резидентным множеством процесса. Во время выполнения процесса всё происходит так, как если бы все ссылки были только на резидентное множество процесса. При помощи таблицы сегментов или страниц процессор всегда может определить, располагается ли блок, к которому требуется обращение, в основной памяти. Если процессор сталкивается с виртуальным адресом, который не находится в основной памяти, он генерирует прерывание, свидетельствующее об ошибке доступа к памяти. Операционная система переводит прерванный процесс в заблокированное состояние и получает управление. Чтобы продолжить выполнение прерванного процесса, операционной системе необходимо загрузить в основную память блок, содержащий вызвавший проблемы виртуальный адрес. Для этого операционная система использует запрос на чтение с диска (во время выполнения которого может продолжаться выполнение других процессов). После того, как необходимый блок загружен в основную память, выполняется прерывание ввода-вывода, передающее управление операционной системе, которая, в свою очередь, переводит заблокированный процесс в состояние готовности.

Как можно видеть, для реализации виртуальной памяти необходима аппаратная поддержка процесса трансляции виртуальных адресов в физические.

Обычно виртуальный адрес представляет собой пару значений (s, d), где s — номер сегмента, а d — смещение внутри сегмента.

В системе (процессоре) обязательно имеется регистр, содержащий адрес таблицы сегментов (блоков) текущего процесса.

Процесс трансляции в общем виде выглядит следующим образом (рис. 30.1). При обращении к памяти из таблицы сегментов, на которую указывает специальный регистр процессора, выбирается строка, соответствующая номеру сегмента s, записанному в виртуальном адресе. Из строки выбирается физический адрес s' загрузки данного сегмента в ОП. Далее вычисляется исполнительный физический адрес ia по формуле

$$ia = s' + d.$$

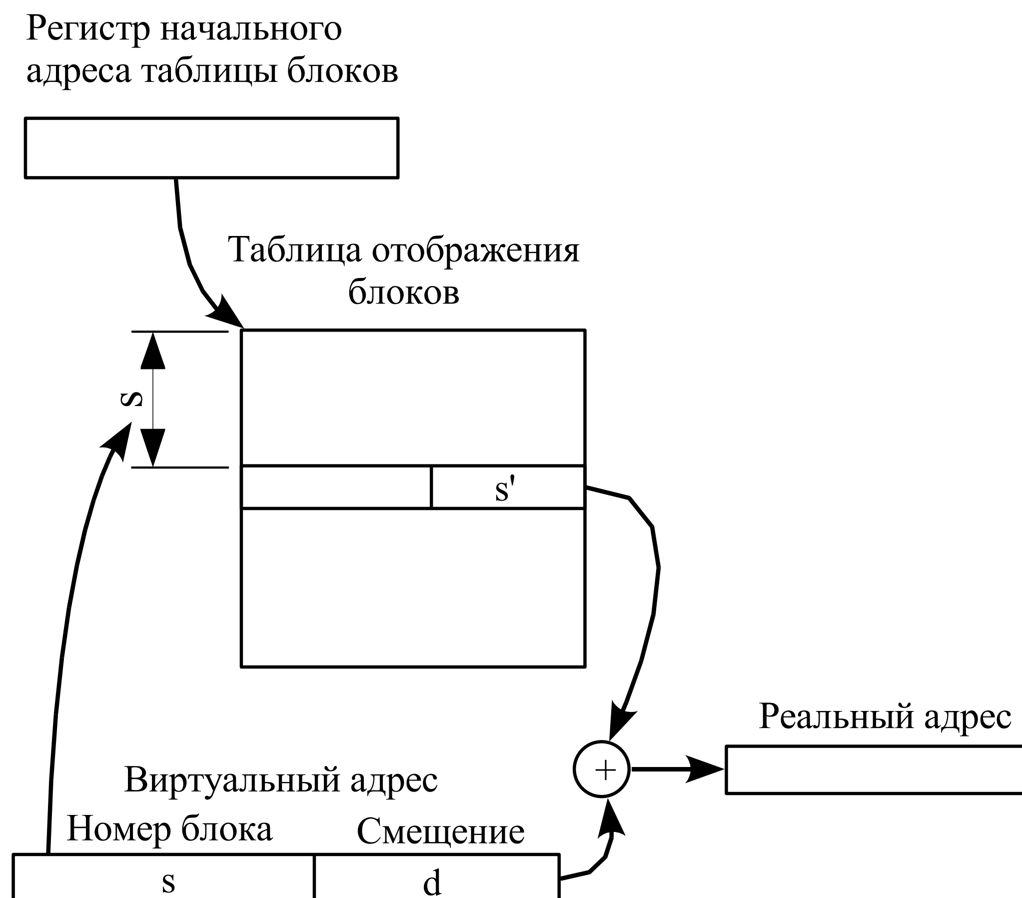

Рис. 30.1. Схема процесса трансляции виртуального адреса centerlinesв реальный

Как можно видеть, каждое обращение к памяти требует на самом деле минимум двух обращений к ОП (не считая обращений к внешней памяти, если нужный сегмент в ОП отсутствует). Если бы запросы к памяти были распределены по процессу равномерно, то механизм виртуальной памяти был бы крайне неэффективным. Однако благодаря свойству локальности и наличию механизмов кэширования в современных ЭВМ, реальная производительность механизма виртуальной памяти достигает 95 % и более от номинальной производительности ЭВМ.

31. Страничная организация виртуальной 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 создаёт 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 подсчёта числа обращений за определённый 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 её в оперативную память.

После загрузки требуемой страницы выполняется преобразование виртуального адреса в физический.

Виртуальный адрес при страничном распределении может быть представлен в виде пары (s, d), где s — номер виртуальной страницы процесса (нумерация страниц начинается с 0), а d — смещение в пределах виртуальной страницы. Учитывая, что размер страницы равен 2^k, смещение d может быть получено простым отделением k младших разрядов в двоичной записи виртуального адреса. Оставшиеся старшие разряды представляют собой двоичную запись номера страницы s.

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

1. На основании начального адреса таблицы страниц (содержимое регистра адреса таблицы страниц), номера виртуальной страницы s (старшие разряды виртуального адреса) и длины записи в таблице страниц (системная константа) определяется адрес нужной записи в таблице.

2. Из этой записи извлекается номер физической страницы s'.

3. К номеру физической страницы присоединяется (конкатенируется) смещение d (младшие разряды виртуального адреса).

Использование в пункте (3) 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.

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 упрощённом 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 счёт того, что программа может загружаться в несмежные области, а также того, что при загрузке виртуальных страниц никогда не образуется остатков.

К числу недостатков страничной организации виртуальной памяти можно отнести невозможность манипулирования частями программы, соответствующими её логической структуре, что может приводить к непроизводительным перезагрузкам страниц, а также исключает возможность организации эффективных механизмов защиты.

32. Сегментная организация виртуальной памяти

В данном случае каждая программа делится на осмысленные, логически законченные части — сегменты, которые становятся единицами манипулирования при реализации механизма виртуальной памяти. Отдельный сегмент может представлять собой подпрограмму, массив данных и т. п. Это позволяет дифференцировать способы доступа к разным частям программы (сегментам), а это свойство часто бывает очень полезным. 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.

Виртуальное адресное пространство процесса делится на сегменты программистом при разработке программы, либо компилятором при её трансляции.

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 создаёт 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.

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, разрешён ли доступ требуемого типа к данному сегменту.

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

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

33. Странично-сегментная организация виртуальной памяти

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 равного размера, кратные степени двойки.

Загрузка процесса выполняется операционной системой постранично, при этом часть страниц размещается в оперативной памяти, а часть на диске. Для каждого сегмента создаётся своя таблица страниц, структура которой полностью совпадает со структурой таблицы страниц, используемой при страничном распределении. Для каждого процесса создаётся 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.

Виртуальный адрес в такой системе представляет собой тройку (s, p, d), где s — номер сегмента, p — номер страницы в сегменте, d — смещение в странице.

Преобразование виртуального адреса в физический выполняется следующим образом. По номеру s из таблицы сегментов выбирается запись, соответствующая запрашиваемому сегменту. По имеющейся в записи информации выполняется проверка прав доступа. Далее из записи выбирается адрес таблицы страниц данного сегмента и по номеру p из таблицы страниц выбирается запись, соответствующая заданной странице. Из выбранной записи извлекается физический адрес начала страницы p', к которому добавляется (конкатенируется) смещение d в странице.

Как можно видеть, одно обращение к памяти требует минимум трёх реальных обращений, что крайне неэффективно. Механизм приобретает эффективность лишь с применением кэширования и ассоциативной памяти для хранения информации из таблиц страниц и сегментов.

34. Проблема предотвращения «пробуксовки» системы

Идея эффективной работы механизма виртуальной памяти основана на том факте, что любой программе для своей работы в течение небольшого интервала времени требуется лишь небольшая часть её программного кода. Иными словами, после загрузки в память очередной страницы (сегмента) некоторое время обращения к памяти будут осуществляться в пределах данной страницы без необходимости загрузки дополнительных страниц.

Чем больше страниц (сегментов) данной программы одновременно находится в оперативной памяти, тем меньше прерываний по отсутствию страниц будет генерироваться, тем быстрее будет выполняться программа. К сожалению, в некоторых «предельных» случаях это правило нуждается в коррекции.

Каждое прерывание по отсутствию страницы заставляет программу войти в состояние блокировки и ждать, пока системой будет загружена необходимая страница. Загрузка страницы может быть выполнена, как правило, лишь на место какой-то другой страницы,

уже имеющейся в памяти. Поскольку заранее невозможно предсказать какая из страниц понадобится в следующий момент, выбор страницы, которая будет вытолкнута из оперативной памяти, представляет серьёзную проблему.

Теоретически любая программа может выполняться в системе лишь на одной странице, перезагружая её по мере необходимости. Однако при этом время, расходуемое на перезагрузку страницы, будет на несколько порядков больше времени, в течение которого выполняются команды программы. В результате производительность системы резко упадёт. Такая ситуация называется «пробуксовкой системы».

Замечено однако, что на протяжении своего жизненного цикла в системе (времени выполнения) в течение определённых интервалов времени каждая программа обращается лишь к ограниченному набору своих адресов памяти. Это свойство получило название локальности.

Если обеспечить загрузку в оперативную память всех страниц, необходимых программе для работы на данном этапе, то в течение некоторого интервала времени программа будет выполняться, не генерируя прерываний по отсутствию страниц, то есть так, как будто весь её код находится в оперативной памяти.

Такое множество страниц, обеспечивающее программе работу без прерываний по отсутствию страниц в течение некоторого времени, получило название «рабочего множества».

В течение времени работы программы её рабочее множество постепенно меняется. Оно может становиться больше или меньше. Задача операционной системы отследить изменение рабочего множества и, по-возможности, обеспечить его для каждой загруженной программы.

35. Проблема эффективности при планировании процессов в системе

Дисциплина планирования должна:

– относиться ко всем процессам одинаково (ни один процесс не должен пострадать из-за бесконечного откладывания);

– обеспечить максимальную пропускную способность системы;

– обеспечить для максимального числа пользователей приемлемые времена ответа;

– быть предсказуемой. Время выполнения задания не должно зависеть от нагрузки на систему;

– сбалансировать использование ресурсов (предпочтение отдаётся тем процессам, которые используют недогруженные ресурсы);

– учитывать приоритеты.

Планирование может быть эффективным лишь в смысле достижения конкретных целей. Среди этих целей могут рассматриваться:

– максимальное количество завершившихся процессов в единицу времени;

– максимальное количество процессов, обслуживаемых системой;

– минимум простаивающих ресурсов системы;

– высокая надёжность работы системы;

– низкие накладные расходы при эксплуатации системы;

– получение максимальной прибыли от эксплуатации системы;

и многие другие.

Многие из этих целей противоречат друг другу, что делает планирование весьма сложной проблемой.

Для достижения указанных целей механизм планирования должен учитывать:

– лимитируется ли процесс вводом-выводом;

– лимитируется ли процесс центральным процессором;

– является ли процесс пакетным или диалоговым;

– насколько часто при выполнении процесса возникают прерывания по отсутствию в оперативной памяти нужных страниц;

– сколько времени уже использовал данный процесс;

– сколько ещё времени требуется данному процессу для завершения.

36. Стратегии управления планированием процессов в системе

В теории рассматриваются по крайней мере следующие стратегии:

– планирование по сроку завершения — планирование задач выполняется таким образом, чтобы каждая задача могла завершиться к указанному времени;

– планирование FIFO — первая поступившая в систему задача обслуживается первой. Как правило, это стратегия без переключения;

– циклическое планирования — «классическое» планирование с переключением, при котором каждой задаче циклически предоставляется квант процессорного времени;

– приоритетное планирование — более приоритетные задачи обслуживаются перед менее приоритетными;

– кратчайшее задание первым — приоритет отдаётся задачам с минимальным оценочным временем выполнения;

– задание с наименьшим остаточным временем первым — аналог предыдущего, но используется оценка остаточного времени выполнения;

– по наибольшему относительному времени реакции — учитывает не только оценочное время завершения задачи, но и время, проведённое задачей в очереди ожидания на выполнение;

– многоуровневые очереди с обратными связями — учитывает характер выполнения задачи: ориентированные преимущественно на ввод-вывод получают процессор в приоритетном порядке, но малые кванты времени, ориентированные преимущественно на вычисления — менее приоритетны, но получают увеличенные кванты времени.

В хорошо спроектированной системе для достижения хорошей производительности и удовлетворения требований пользователей приходится идти на компромиссы и применять комбинированные стратегии.

37. Трёхуровневое планирование выполнения задач в системе

Планирование на верхнем уровне. Иногда называется планированием заданий. Средства этого уровня определяют, каким заданиям будет разрешено активно конкурировать за захват ресурсов системы. Этот вид планирования иногда называют также планированием допуска, поскольку на этом уровне определяется, какие задания будут допущены в систему. Вошедшие в систему задания становятся процессами или группами процессов.

Планирование на промежуточном уровне. Средства этого уровня определяют, каким процессам будет разрешено состязаться за захват центрального процессора. Планировщик промежуточного уровня оперативно реагирует на текущие колебания системной нагрузки, кратковременно приостанавливая и вновь активизируя (или возбуждая) процессы, что обеспечивает равномерную работу системы и помогает достижению определённых глобальных целевых скоростных характеристик. Таким образом, планировщик промежуточного уровня выполняет как бы функции буфера между средствами допуска заданий в систему и средствами предоставления ЦП для выполнения этих заданий.

Планирование на нижнем уровне. Средства этого уровня определяют, какому из готовых к выполнению процессов будет предоставляться освободившийся ЦП, и фактически выделяют ЦП данному процессу (т. е. осуществляют диспетчерские функции). Планирование на нижнем уровне производится так называемым диспетчером, который работает с большой частотой, много раз в секунду, и поэтому всегда должен располагаться в основной памяти.

38. Кэширование. Принцип работы кэш-памяти

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 расчёте на один бит. Пользователю хотелось бы иметь и недорогую и быструю память. Кэш-память представляет некоторое компромиссное решение этой проблемы.

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 счёт 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 объём. 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; всё это делается автоматически системными средствами.

Например, для уменьшения среднего времени доступа процессора к данным, хранящимся в оперативной памяти, между процессором и оперативной памятью помещается быстрое ЗУ, называемое просто кэш-памятью. В качестве такового может быть использована, например, ассоциативная память. Содержимое кэш-памяти представляет собой совокупность записей обо всех загруженных в неё 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.

Использование кэш-памяти позволяет значительно ускорить выполнение многих операций в системе за счёт свойства локальности, наблюдаемого в выполняемых программах.

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

1. Просматривается содержимое кэш-памяти с целью определения, не находятся ли нужные данные в кэш-памяти; кэш-память не является адресуемой, поэтому поиск нужных данных осуществляется по содержимому — значению поля «адрес в оперативной памяти», взятому из запроса. Причём, поиск должен выполняться быстро, поэтому эта память обычно является ассоциативной.

2. Если данные обнаруживаются в кэш-памяти, то они считываются из неё, и результат передаётся в процессор.

3. Если нужных данных нет, то они вместе со своим адресом копируются из оперативной памяти в кэш-память, и результат выполнения запроса передаётся 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 кэш-попадания, то есть нахождения нужных данных в кэш-памяти.

Всё предыдущее описание справедливо 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, а роль кэш-памяти выполняет буфер в оперативной памяти.

39. Управление вводом-выводом как функция операционной системы

Выполняющимся процессам необходим доступ к различным устройствам ввода-вывода, например, дискам, лентам, принтерам, терминалам. Если бы каждый процесс выполнял управление всеми внешними устройствами самостоятельно, то несогласованность со стороны разных процессов мгновенно привела бы к краху системы.

Кроме того, управление периферийными устройствами требует знания особенностей функционирования каждого устройства. Программы управления объёмны и зачастую сложны. Включение программного кода управления устройствами в каждую программу сильно затрудняет программирование и увеличивает объём программ.

Поэтому одной из важнейших задач современной ОС является предоставление программам простого интерфейса доступа к различным устройствам ввода-вывода и сокрытие от них внутренних механизмов работы этих устройств.

ОС берёт на себя функции собственно управления внешними устройствами и обеспечения совместного использования этих устройств множеством одновременно выполняющихся процессов. При этом для некоторых устройств ОС может реализовать режим разделяемого использования несколькими процессами. Для тех же устройств, для которых режим разделения недопустим, ОС реализует функции закрепления устройств в монопольное использование запросившим их процессам.

В большинстве современных ОС закрепился принцип представления внешних устройств разных типов в виде обобщённой модели специальных файлов, доступ к которым процессов осуществляется стандартными командами чтения файла и записи в файл.

Все реальные операции обслуживания устройств скрыты в программах управления, называемых драйверами.

40. Назначение каналов ввода-вывода и организация управления ими в операционной системе

Каналом ввода-вывода называют специальное устройство, часто имеющее специализированный процессор и работающее по особой программе, предназначенное для обслуживания внешних устройств и выполнения операций ввода-вывода.

При необходимости выполнить операцию ввода-вывода центральный процессор должен подготовить в ОП область данных и программу для канала, после чего запустить канал в работу. Канал будет осуществлять ввод-вывод самостоятельно, получая доступ к ОП

напрямую (методом прямого доступа к памяти, ПДП). Центральный процессор в это время может выполнять другую программу.

По завершении операции ввода-вывода канал выдаст центральному процессору сигнал (прерывание) и сообщит о результатах выполнения операции (нормальное окончание, либо окончание по какой-либо ошибке).

Истинное значение каналов состоит в том, что они позволяют значительно увеличить параллелизм работы аппаратуры компьютера и освобождают процессор от подавляющей части нагрузки, связанной с управлением вводом-выводом.

Для высокоскоростного обмена данными между внешними устройствами и основной памятью используется селекторный канал. Селекторные каналы имеют только по одному подканалу и могут обслуживать в каждый момент времени только одно устройство.

Мультиплексные каналы имеют много подканалов; они могут работать сразу с многими потоками данных в режиме чередования. Байт-мультиплексный канал обеспечивает режим чередования байтов при одновременном обслуживании ряда таких медленных внешних устройств, как терминалы, перфокарточные устройства ввода-вывода, принтеры, а также низкоскоростные линии передачи данных. Блок-мультиплексный канал при обменах в режиме чередования блоков может обслуживать несколько таких высокоскоростных устройств, как лазерные принтеры и дисковые накопители.

41. Управление печатью на принтере как функция операционной системы

Хотя принтер не может печатать информацию, поступающую одновременно от нескольких процессов (точнее, может, но в результате на печати информация от разных процессов будет перемешана, что не позволит её использовать), может оказаться желательным разрешить процессам совершать вывод на принтер параллельно. Для этого операционная система вместо передачи информации напрямую на принтер накапливает выводимые данные в буферах на диске, организованных в виде отдельного спулинг-файла для каждого процесса. После завершения некоторого процесса соответствующий ему спулинг-файл ставится в очередь для реальной печати. Механизм, обеспечивающий подобные действия, и получил название «спулинг».

Другим решением проблемы может быть представление принтера как неразделяемого устройства и требование от процессов получать принтер как ресурс в монопольное владение для осуществления

операций вывода. Это решение может быть оправдано лишь на системах, имеющих недостаточные объёмы дисковой памяти для организации спулинга, так как будет вынуждать процессы, осуществляющие печать, долго ожидать освобождения принтера (скорость печати на несколько порядков ниже скорости вывода информации процессом).

42. Назначение файловых систем

Файл — это поименованный набор связанной информации, записанной во вторичную память.

С точки зрения пользователя, файл — единица внешней памяти, то есть данные или программы, записанные на диск, должны быть в составе какого-нибудь файла или представляться отдельным файлом.

Файловая система — это часть операционной системы, назначение которой состоит в том, чтобы обеспечить пользователю удобный интерфейс при работе с данными, хранящимися во вторичной памяти, и обеспечить совместное использование файлов несколькими пользователями и процессами.

Основные функции файловой системы:

– связывание имени файла с выделенным ему пространством внешней памяти;

– распределение внешней памяти между файлами;

– обеспечение надёжности и отказоустойчивости;

– обеспечение защиты от несанкционированного доступа;

– обеспечение совместного доступа к файлам;

– обеспечение высокой производительности.

43. Поддержка файловой системы как функция операционной системы

Операционная система предоставляет пользователям и программам простую модель доступа к данным на внешних носителях, представляя данные в виде файлов и обеспечивая возможность работы с ними с помощью простых запросов типа «открыть файл», «прочитать из файла», «записать в файл» и т. п. Файлы имеют с точки зрения пользователя линейную (последовательную) организацию.

Файлы могут храниться на внешних устройствах разного типа, например, магнитных дисках, компакт-дисках, флэш-картах и т. п. Независимо от типа устройства, посредством механизма файловой системы операционная система обеспечивает единообразный доступ к информации на всех типах устройств.

Физически данные хранятся на внешних носителях, имеющих сложную организацию. Например, на дисках информация хранится на пластинах, разделённых на секторы и дорожки. Чтобы получить доступ к порции информации необходимо указать номера поверхности (или считывающей головки), дорожки (цилиндра) и сектора. Возможно считывание или запись только блока информации (сектора) полностью.

Операционная система берёт на себя преобразование запросов пользователей (программ) на доступ к файлам в последовательности команд на доступ к конкретным физическим единицам пространства диска, производит буферизацию ввода-вывода, блокирование и разблокирование данных при необходимости.

Операционная система поддерживает структуры данных (файловую систему), обеспечивающие размещение на устройстве файлов, выделение им необходимого пространства, поддержание их целостности и непротиворечивости, защиту от несанкционированного доступа, а также решение других задач.

44. Варианты организации доступа к файлам в операционной системе. Преимущества и недостатки

Доступ к файлам в современных ОС может быть организован одним из двух способов:

– с предварительным установлением связи с файлом;

– без предварительного установления связи с файлом.

В первом случае перед началом работы с файлом его необходимо открыть. При открытии ОС производит поиск файла по его имени, создаёт в ОП структуру, описывающую файл, и возвращает указатель на эту структуру, называемый дескриптором файла. Операции чтения и записи выполняются со ссылками на этот дескриптор, причём ОС сама отслеживает положение указателя чтения (записи) в файле, а также обнаруживает конец файла, то есть прикладной программе, как правило, нет необходимости явно указывать место в файле, где должны располагаться данные. По окончании работы с файлом его необходимо закрыть. При этом осуществляется сброс на носитель файловых буферов и из ОП удаляется структура, описывающая файл.

Во втором случае нет необходимости открывать и закрывать файл. Программа выполняет только обращения к файлу на чтение, либо на запись. Но при каждом обращении необходимо указывать

полную информацию — имя файла, смещение данных от начала файла, тип операции.

Способ с предварительным установлением связи обеспечивает более эффективный режим работы, в том числе за счёт возможности промежуточной буферизации данных и использования метода качающихся буферов. Но его применение нежелательно для сетевых файловых систем. Информация об открытом файле должна храниться на стороне сервера, и при его отказе (зависании, падении) после восстановления работоспособности вероятнее всего будет потеряна, что не позволит продолжать работу прикладным программам клиентов, находящимся на других компьютерах.

Поэтому в сетевых файловых системах применяется способ доступа к файлам без предварительного установления связи с файлом.

На самом деле ОС на стороне клиента скрывает от прикладных программ этот механизм. Обычно прикладные программы осуществляют работу с любыми файлами по способу с предварительным установлением связи. А ОС поддерживает необходимые таблицы и структуры данных, самостоятельно выполняя обращения к файлам, находящимся на сетевых файловых системах, способом без предварительного установления связи с ними.

45. Понятие драйвера. Аппаратные и программные драйвера

Программа, управляющая конкретной моделью внешнего устройства и учитывающая все его особенности обычно называется драйвером этого устройства. Весь зависимый от устройства код помещается в драйвер устройства. Каждый драйвер управляет устройствами одного типа или, может быть, одного класса.

В современных операционных системах выделяют аппаратные и программные драйвера.

Аппаратные драйвера отвечают за собственно взаимодействие с аппаратурой внешних устройств. Но могут выполнять только запросы низкого уровня. Например, драйвер контроллера IDE жёсткого диска воспринимает команды записи на диск и чтения с диска, выраженные в терминах головок, дорожек, секторов.

Программные драйвера предназначены для преобразования высокоуровневых запросов в низкоуровневые. Например, программный драйвер файловой системы воспринимает команды чтения-записи в терминах файлов и транслирует их в команды чтения-записи в терминах головок, дорожек, секторов.

Интерфейсы между всеми драйверами в рамках одной ОС стандартизованы, что позволяет организовывать взаимодействие между драйверами, обеспечивая программистам и пользователям удобство применения ОС.

Драйвера как правило оформляются как часть ядра ОС и работают в привилегированном режиме, что как раз и обеспечивает им возможность непосредственного взаимодействия с аппаратурой внешних устройств.

В зависимости от типа обслуживаемого устройства драйвер может быть блок-ориентированным или байт-ориентированным.

Если обмен с устройством на физическом уровне можно выполнить только блоком данных, то такое устройство является блок-ориентированным и, соответственно, должно обслуживаться блок-ориентированным драйвером. Типичный пример такого устройства — накопитель на магнитной ленте или жёсткий диск.

Многие другие устройства используют побайтовый ввод-вывод на физическом уровне. Такие устройства должны обслуживаться байт-ориентированными драйверами. Примеры таких устройств — принтеры, терминалы, модемы и многие другие.

46. Иерархия драйверов в операционной системе

Многослойное построение программного обеспечения, характерное для операционных систем вообще, оказывается особенно естественным и полезным при построении подсистемы ввода-вывода. При большом разнообразии устройств ввода-вывода, обладающих существенно различными характеристиками, иерархическая структура программного обеспечения позволяет соблюсти баланс между двумя требованиями: с одной стороны, необходимо учесть все особенности каждого устройства, а с другой стороны, обеспечить единое логическое представление и унифицированный интерфейс для устройств всех типов. При этом нижние слои подсистемы ввода-вывода должны включать индивидуальные драйверы, написанные для конкретных физических устройств, а верхние слои должны обобщать процедуры управления этими устройствами, предоставляя общий интерфейс если не для всех устройств, то по крайней мере для групп устройств, обладающих некоторыми общими характеристиками.

Примеры иерархических структур драйверов рассмотрим на конкретных примерах.

В подсистеме дискового ввода-вывода нижний уровень образован аппаратными драйверами устройств (дисков), имеющих различные

аппаратные интерфейсы: MFM, IDE, SCSI, SATA. Следующий уровень иерархии образуют программные драйверы конкретных файловых систем, которые могут быть использованы для хранения данных на дисках, подключённых по этим интерфейсам: FAT, ext2, ext3, NTFS, HPFS и др. Самый верхний уровень иерархии представлен программным драйвером виртуальной (обобщённой) файловой системы, предоставляющей универсальный доступ к файлам в файловой системе любого типа на устройствах с любым интерфейсом.

В подсистеме обмена по сети нижний уровень образован аппаратными драйверами сетевых контроллеров: ne2000, 3com, CNet, Realtek и т. п. Программные драйверы промежуточного уровня могут предоставлять различные уровни сетевых протоколов: UDP, TCP/IP, IPX/SPX, Tokenring и т. п. Программные драйверы верхнего уровня предоставляют доступ в сеть через протоколы различных служб: http, ftp, nfs и т. п.

Иерархичность драйверов облегчает решение большинства задач управления подсистемой ввода-вывода, таких как простота включения новых драйверов, поддержка нескольких файловых систем, динамическая загрузка-выгрузка драйверов и других.

При таком подходе повышается гибкость и расширяемость функций по управлению устройствами — вместо жёсткого набора функций, сосредоточенных в единственном драйвере, администратор ОС может выбрать требуемый набор функций, установив нужный высокоуровневый драйвер. Если различным приложениям необходимо работать с различными логическими моделями одного и того же физического устройства, то для этого достаточно установить в системе несколько драйверов на одном уровне, работающих над одним аппаратным драйвером.

Количество уровней драйверов в подсистеме ввода-вывода обычно не ограничивается каким-либо пределом, но на практике чаще всего используют от двух до пяти уровней драйверов — слишком большое количество уровней может снизить скорость операций ввода-вывода. Несколько драйверов, управляющих одним устройством, но на разных уровнях, можно рассматривать как набор отдельных драйверов или как один многоуровневый драйвер.

В унификацию драйверов большой вклад внесла операционная система UNIX. В ней все драйверы были разделены на два больших класса блок-ориентированные драйверы и байт-ориентированные драйверы.

Блок-ориентированные драйверы управляют устройствами прямого доступа, которые хранят информацию в блоках фиксированного размера, каждый из которых имеет собственный адрес. Самое распространённое внешнее устройство прямого доступа — диск. Адресуемость блоков приводит к тому, что для устройств прямого доступа появляется возможность кэширования данных в оперативной памяти, и это обстоятельство значительно влияет на общую организацию ввода-вывода для блок-ориентированных драйверов.

Устройства, с которыми работают байт-ориентированные драйверы, не адресуемы и не позволяют производить операцию поиска данных, они генерируют или потребляют последовательности байт. Примерами таких устройств, которые так же называют устройствами последовательного доступа, служат терминалы, строчные принтеры, сетевые адаптеры.

Для байт-ориентированного устройства невозможно разработать блок-ориентированный драйвер. Но для блок-ориентированного устройства разработка байт-ориентированного драйвера возможна и бывает полезна на практике. В этом случае байт-ориентированный драйвер располагается на более высоком уровне иерархии относительно блок-ориентированного драйвера.

47. Проблема эффективности при доступе к вращающимся накопителям информации (например, жёстким дискам)

При работе диска набор пластин вращается вокруг своей оси с высокой скоростью, подставляя по очереди под головки соответствующих дорожек все их сектора. Номер сектора, номер дорожки и номер цилиндра однозначно определяют положение данных на жёстком диске и, наряду с типом совершаемой операции — чтение или запись, — полностью характеризуют часть запроса, связанную с устройством, при обмене информацией в объёме одного сектора.

Для выполнения запроса на чтение или запись к диску последовательно выполняются следующие действия:

– каретка с головками перемещает головки таким образом, чтобы они оказались над заданной дорожкой;

– система ожидает, пока диск повернётся таким образом, чтобы под головкой оказался нужный сектор;

– выполняется чтение или запись данных и их передача контроллеру диска.

Как можно видеть, скорость выполнения всех операций связана с ограничениями механики — скоростями позиционирования головок

и вращение дисков. Как следствие, время выполнения любого запроса к диску на ввод-вывод складывается из трёх составляющих: времени позиционирования головок, времени ожидания поворота диска (в сумме эти два времени называются временем доступа) и времени считывания или записи сектора, равного времени прохода сектора под головкой.

Последнее время является постоянным, в то время как два других времени меняются в широких пределах в зависимости от момента появления запроса на выполнение операции и текущего положения головок.

При небольшой интенсивности запросов к диску, когда время доступа сравнимо с интервалами их возникновения или даже меньше их, с задержками в работе программ, возникающими из-за длительности операций доступа к диску, вполне можно мириться (уменьшить время доступа практически невозможно).

Однако при большой интенсивности запросов к диску из-за большого времени обслуживания каждого запроса возникнет очередь запросов, в которой одновременно может находиться много программ. Учитывая механические особенности обслуживания каждого запроса (прежде всего необходимость перемещать каретку с головками), можно попытаться переупорядочить запросы в очереди с тем, чтобы, ухудшив времена обслуживания отдельных «неудобных» запросов, значительно улучшить (сократить) времена обслуживания множества других запросов. Тем самым возможно повысить эффективность работы дисковой подсистемы.

48. Стратегии оптимизации среднего времени доступа к жёсткому диску

Алгоритм First Come First Served (FCFS)

Простейший алгоритм — первым пришёл, первым обслужен. Все запросы организуются в очередь FIFO и обслуживаются в порядке поступления. Алгоритм прост в реализации, но может приводить к достаточно длительному общему времени обслуживания запросов.

Алгоритм Short Seek Time First (SSTF)

Кратчайшее время поиска первым — выполняет первоочередное обслуживание запросов, данные для которых лежат рядом с текущей позицией головок, а уж затем далеко отстоящих.

Данный алгоритм может приводить к бесконечному откладыванию далеко отстоящих запросов.

Алгоритм SCAN

Простейший из алгоритмов сканирования — головки постоянно перемещаются от одного края диска до другого, по ходу дела обслуживая все встречающиеся запросы. По достижении другого края направление движения меняется, и всё повторяется снова. Недостатком алгоритма является примерно вдвое более частое обслуживание запросов в центре диска, чем по краям.

Алгоритм C-SCAN

Модификация алгоритма SCAN — циклическое сканирование. Обслуживание запросов выполняется только при движении головки в одном направлении (например, от края к центру). Далее происходит быстрый скачок головки в обратном направлении и цикл повторяется.

Модификации N-Step

Так называемые N-шаговые модификации предыдущих алгоритмов сканирования. Оба предыдущих типа алгоритмов могут использовать эти модификации. Идея заключается в том, что при каждом цикле движения головок выполняется не более N запросов к каждому цилиндру, а остальные переносятся на следующий цикл.

Модификации позволяют более равномерно обслуживать запросы, относящиеся к разным дорожкам, и исключают проблему бесконечного откладывания.

Схема Эшенбаха

Учитывает расположение запросов по секторам диска и требует, чтобы при каждом цикле сканирования на каждом цилиндре обслуживались только те запросы, которые расположены в разных секторах, причём только в течение одного оборота диска. Остальные запросы откладываются на следующий цикл сканирования.

49. Условия эффективного и неэффективного применения стратегий оптимизации среднего времени доступа к жёсткому диску

Применение стратегий оптимизации требует дополнительных накладных расходов системы, связанных с необходимостью вести очередь запросов, сортировать эту очередь, осуществлять из неё выборку. Поэтому применение стратегий эффективно лишь в тех случаях, когда:

– велико количество параллельно выполняющихся процессов, причём велика также и частота запросов на дисковые операции с их стороны;

– к времени выполнения каждого отдельного запроса не предъявляется жёстких требований;

– все (или большинство) запросов адресуется к одному диску.

Применение стратегий неэффективно (или даже вредно) в случаях, когда:

– каждый процесс осуществляет запросы к выделенному ему диску. Причём планирование особенно неэффективно, если выполняются последовательные запросы на чтение или запись одного большого файла;

– количество процессов невелико и запросы к диску имеют низкую интенсивность.

50. Эффективность функционирования операционной системы

Под эффективностью понимают способность ОС достигать целей и удовлетворительно решать задачи, для которых она проектировалась.

Не бывает эффективных и неэффективных ОС вообще. Существуют лишь ОС, решающие поставленные перед ними задачи с большей или меньшей эффективностью. В этом смысле эффективность — понятие не абсолютное (не выражаемое количественно), а относительное (качественное).

Например, пакетная ОС как правило ориентируется на максимальную загрузку ЭВМ и максимальную пропускную способность. Она эффективна при решении потока неинтерактивных (вычислительных) задач. Но средства интерактивного взаимодействия с пользователями у неё не развиты. И с точки зрения таких пользователей система неудобна, медлительна, то есть неэффективна.

Система разделения времени ориентирована на интерактивное взаимодействие с пользователями. Она способна эффективно обслуживать одновременные запросы большого количества пользователей. Но для этого в ней применён механизм планирования, требующий больших накладных расходов на переключение и обслуживание большого количества процессов. В смысле выполнения «чистых» вычислений она неэффективна.

Система реального времени должна обеспечить своевременную реакцию в пределах заданных интервалов времени на любую последовательность заранее предусмотренных событий. В этом эффективность её применения. Однако для реализации этой задачи система должна обладать соответствующими ресурсами, которые простаивают при отсутствии событий, то есть используются неэффективно.

С целью повышения эффективности могут создаваться ОС, сочетающие в себе черты и режимы работы нескольких разных типов ОС. Повышение эффективности применения ОС возможно также при подборе её компонентов (например, планировщика процессов), который будет лучше (быстрее, с меньшими накладными расходами) управлять вычислениями в данной вычислительной обстановке (при данном потоке заявок, при данной смеси задач и т. п.).

51. Цели и методы сбора информации об эффективности функционирования операционной системы и ЭВМ

Цели:

– до приобретения ЭВМ — оценить необходимую производительность ЭВМ для решения стоящих перед организацией (пользователем) задач, определить режимы, в которых будет эксплуатироваться ЭВМ;

– непосредственно в процессе приобретения ЭВМ — выбрать поставщика, конкретный состав аппаратных и программных средств, в том числе тип ОС;

– после приобретения, в процессе эксплуатации — оценить правильность выбора, эффективность реальной работы, необходимость тонкой настройки ОС и оборудования, необходимость коррекции состава оборудования; мониторинг изменения нагрузки с течением времени, разработка и реализация мер по подгонке существующей вычислительной установки под требования изменяющейся нагрузки.

Методы для вновь приобретаемой системы: расчёт; моделирование; эксперимент (прогон тестовых программ, эталонных программ) на предоставленной поставщиком аппаратуре.

Методы для уже приобретённой системы: журналирование, сбор статистики о работе системы с последующим анализом собранной информации; наблюдение за работой системы, мониторинг работы отдельных интерфейсов и системы в целом.

Мониторинг может осуществляться как чисто программными средствами (дополнительная нагрузка на систему, не все сечения можно контролировать), так и с применением специальных устройств (высокая стоимость, но небольшая нагрузка на систему и более глубокий контроль работы системы).

52. Оптимизация работы вычислительной системы

При проектировании ВС, стремятся обеспечить наиболее полное соответствие системы своему назначению. Степень соответствия системы своему назначению называется эффективностью (качеством) системы. Для сложных систем, какими являются ВС, эффективность не удаётся определить одной величиной, и поэтому её представляют набором величин, называемых характеристиками системы. Набор характеристик формируется таким образом, чтобы в своей совокупности они давали наиболее полное представление об эффективности системы. Основными характеристиками ВС являются производительность, время ответа, надёжность и стоимость.

Под оптимизацией ВС понимают улучшение некоторых её характеристик, важных в конкретной ситуации, возможно, за счёт ухудшения других, менее важных характеристик.

Например, в системах реального времени важнейшими характеристиками являются время ответа и надёжность. Для достижения наилучших показателей этих характеристик чаще всего приходится жертвовать стоимостью системы (она оказывается очень большой).

В случае ВС, которая уже эксплуатируется долгое время, может быть предпринята попытка добиться, например, лучшей производительности в связи с возросшей нагрузкой или увеличившимся количеством пользователей. В этом случае необходим анализ работы системы с определением узких мест, которые ограничивают возможности роста производительности. В результате могут быть приняты меры, направленные на ликвидацию обнаруженных узких мест — замена контроллеров управления периферийными устройствами, увеличение количества накопителей информации и т. п.

Однако в результате анализа может быть установлено, что экономические потери от недостаточной производительности системы значительно меньше, чем затраты на модернизацию системы или приобретение новой системы. В этом случае может оказаться, что существующая система является наиболее эффективной с экономической точки зрения.

Вопрос о необходимости оптимизации работы ВС и целях оптимизации должен решаться отдельно в каждом конкретном случае. Каких-либо общих рекомендаций на эту тему дать невозможно.

53. Программы с оверлейной структурой. Цель применения. Принципы построения и функционирования. Преимущества и недостатки

Оверлей — буквально «лежащий сверху» или просто «перекрытие». Это способ загрузки и выполнения программ, при котором в ОП загружается не вся программа, а только та её часть, которая необходима для выполнения какого-либо этапа её работы. По завершении этого этапа на её место загружается другая часть программы (перекрывает первую часть), и работа программы продолжается.

Важно подчеркнуть, что, несмотря на определённое сходство между задачами, решаемыми механизмом перекрытий и виртуальной адресацией, одно и в коем случае не является разновидностью другого. При виртуальной адресации решается задача отображения большого адресного пространства в ограниченную оперативную память. При использовании оверлея решается задача отображения большого количества объектов (модулей программы) в ограниченное адресное пространство.

Основная проблема при оверлейной загрузке состоит в следующем: прежде чем ссылаться на оверлейный адрес, надо понять, какой из оверлейных модулей в данный момент там находится. Эта проблема обычно решается построением дерева зависимостей оверлейных модулей и требованием, что каждый дочерний модуль может быть вызван только из своего родительского модуля. Передача управления из одного дочернего модуля другому может быть осуществлена только через родительский модуль (а если модули, передающие друг другу управление, не являются дочерними модулями одного родительского, то через родительский модуль родительских модулей этих модулей).

Собственно передача управления и при необходимости загрузка оверлейных модулей может быть реализована как через вызов специальной службы ОС, так и без привлечения ОС с использованием специального программного модуля — менеджера оверлея, — включаемого в состав прикладной программы из специальной библиотеки на этапе редактирования связей.

Распределение кода программы по оверлейным модулям и определение дерева зависимости оверлейных модулей обычно возлагается на программиста.

Преимущества применения программ с оверлейной структурой особенно заметны на системах с малой ОП. За счёт перекрытия разными модулями одного адресного пространства удаётся разрабатывать и выполнять программы, которые полностью в ОП не помещаются.

Недостатками оверлейного программирования являются:

– сложность программирования — построение структуры программы возлагается на программиста;

– не каждую программу можно эффективно закодировать с применением оверлея — в программе должны просматриваться независимо выполняющиеся части;

– на перезагрузку оверлеев (операция дискового ввода-вывода) тратится заметное время — программа выполняется медленнее, чем без применения оверлея.

54. Раскручивающиеся загрузчики. Назначение. Принцип многоступенчатой загрузки ОС

Обычно программу в ОП компьютера загружает загрузчик. Но сам загрузчик также является программой и, следовательно, каким-то образом когда-то чем-то должен загружаться в ОП. Более того, ОС также является лишь программой и, следовательно, тоже каким-то способом должна загружаться в ОП.

Задачу загрузки в ОП загрузчика и всей ОС решает специальный компонент, получивший название раскручивающего загрузчика.

Основная идея раскручивающего загрузчика состоит в том, чтобы минимальной по объёму программой, которую при необходимости и наличии технической возможности можно ввести в ОП вручную, инициировать процесс загрузки ОС, загружая с диска все необходимые программы и данные.

Обычно процесс раскручивающей загрузки состоит из нескольких этапов. Первоначальный раскручивающий загрузчик настолько

мал, что выполнить загрузку всей ОС не может. Вместо этого он лишь читает с диска известную ему стандартную запись, помещает её в ОП и передаёт ей управление.

В прочитанной записи находится более сложная программа загрузки, которая уже может запросить оператора о варианте загрузки ОС и устройстве (например, разделе диска), с которого необходимо выполнить загрузку. Она читает начальную запись загрузки с заданного устройства и передаёт ей управление.

В прочитанной записи уже содержится программа загрузчика конкретной ОС. Она находит ядро ОС, загружает его в ОП и передаёт ему управление.

Все описанные выше процессы выполняются с обращением к диску без использования файловых систем — в терминах дорожка, головка, сектор.

Ядро инициализирует необходимые системные таблицы (в том числе таблицы управления ОП), считывает с диска таблицы файловой системы и далее пытается найти (уже как файл) и загрузить предопределённый стартовый процесс, который будет определять далее всё поведение ОС с точки зрения пользователя.

В современных ЭВМ роль раскручивающего загрузчика играет программа, прошитая в ПЗУ. Причём прошита она таким образом, чтобы первая команда, которую будет выбирать процессор после сигнала начальной установки (Reset), оказалась первой командой этой программы. В результате всегда после команды Reset (при включении питания компьютера такая команда генерируется аппаратурой автоматически) начинает выполняться программа раскручивающего загрузчика.

На некоторых ЭВМ принята другая схема. Первой командой, выбираемой процессором из ПЗУ после команды Reset, является команда технологической программы монитора (пульта), предоставляющая пользователю возможность посредством простых команд с клавиатуры выполнять команды прямого управления аппаратурой компьютера: записать/считать данные указанной ячейки памяти, провести тест заданного устройства и т. п. Для инициализации раскручивающего загрузчика пользователь должен подать с клавиатуры специальную команду монитора, либо вручную записать в ОП программу загрузчика в машинных кодах и передать ей управление.

55. Проблема безопасности в операционных системах. Основные вопросы защиты

Есть несколько причин для реализации средств защиты. Наиболее очевидная — помешать внешним попыткам нарушить доступ к конфиденциальной информации. Не менее важно, однако, гарантировать, что каждый программный компонент в системе использует системные ресурсы только способом, совместимым с установленной политикой применения этих ресурсов. Такие требования абсолютно необходимы для надёжной системы. Кроме того, наличие защитных механизмов может увеличить надёжность системы в целом за счёт обнаружения скрытых ошибок интерфейса между компонентами системы.

Политика в отношении ресурсов может меняться в зависимости от приложения и с течением времени. Операционная система должна обеспечивать прикладные программы инструментами для создания и поддержки защищённых ресурсов. Здесь реализуется важный для гибкости системы принцип — отделение политики от механизмов. Механизмы определяют, как может быть сделано что-либо, тогда как политика решает, что должно быть сделано. Политика может меняться в зависимости от места и времени. Желательно, чтобы были реализованы, по-возможности, общие механизмы, тогда как изменение политики требует лишь модификации системных параметров или таблиц.

К сожалению, построение защищённой системы предполагает необходимость склонить пользователя к отказу от некоторых интересных возможностей. Например, письмо, содержащее в качестве приложения документ в формате Word, может включать макросы. Открытие такого письма может повлечь за собой запуск чужой программы, что потенциально опасно. То же самое можно сказать про Web-страницы, содержащие апплеты.

Знание возможных угроз, а также уязвимых мест защиты, которые эти угрозы обычно эксплуатируют, необходимо для того, чтобы выбирать наиболее экономичные средства обеспечения безопасности.

Считается, что безопасная система должна обладать свойствами конфиденциальности, доступности и целостности. Любое потенциальное действие, которое направлено на нарушение конфиденциальности, целостности и доступности информации, называется угрозой. Реализованная угроза называется атакой.

Конфиденциальная система обеспечивает уверенность в том, что секретные данные будут доступны только тем пользователям, которым этот доступ разрешён (такие пользователи называются авторизованными). Под доступностью понимают гарантию того, что авторизованным пользователям всегда будет доступна информация, которая им необходима. И наконец, целостность системы подразумевает, что неавторизованные пользователи не могут каким-либо образом модифицировать данные.

Защита информации ориентирована на борьбу с так называемыми умышленными угрозами, то есть с теми, которые, в отличие от случайных угроз (ошибок пользователя, сбоев оборудования и др.), преследуют цель нанести ущерб пользователям ОС.

Умышленные угрозы подразделяются на активные и пассивные. Пассивная угроза — несанкционированный доступ к информации без изменения состояния системы, активная — несанкционированное изменение системы. Пассивные атаки труднее выявить, так как они не влекут за собой никаких изменений данных. Защита против пассивных атак базируется на средствах их предотвращения.

Можно выделить несколько типов угроз. Наиболее распространённая угроза — попытка проникновения в систему под видом легального пользователя, например, попытки угадывания и подбора паролей. Более сложный вариант — внедрение в систему программы, которая выводит на экран слово login. Многие легальные пользователи при этом начинают пытаться входить в систему, и их попытки могут протоколироваться.

Угрозы другого рода связаны с нежелательными действиями легальных пользователей, которые могут, например, предпринимать попытки чтения страниц памяти, дисков и лент, которые сохранили информацию, связанную с предыдущим использованием. Защита в таких случаях базируется на надёжной системе авторизации. В эту категорию также попадают атаки типа отказ в обслуживании, когда сервер затоплен мощным потоком запросов и становится фактически недоступным для отдельных авторизованных пользователей.

Наконец, функционирование системы может быть нарушено с помощью программ-вирусов. Многопользовательские компьютеры меньше страдают от вирусов по сравнению с персональными, поскольку там имеются системные средства защиты.

Проектирование системы безопасности подразумевает ответы на следующие вопросы: какую информацию защищать, какого рода атаки на безопасность системы могут быть предприняты, какие средства использовать для защиты каждого вида информации?

Поиск ответов на данные вопросы называется формированием политики безопасности, которая помимо чисто технических аспектов включает также и решение организационных проблем. На практике реализация политики безопасности состоит в присвоении субъектам и объектам идентификаторов и фиксации набора правил, позволяющих определить, имеет ли данный субъект авторизацию, достаточную для предоставления к данному объекту указанного типа доступа.

Формируя политику безопасности, необходимо учитывать несколько базовых принципов.

1) Проектирование системы должно быть открытым. Нарушитель и так всё знает (криптографические алгоритмы открыты).

2) Не должно быть доступа по умолчанию. Ошибки с отклонением легитимного доступа будут обнаружены скорее, чем ошибки там, где разрешён неавторизованный доступ.

3) Нужно тщательно проверять текущее авторство. Так, многие системы проверяют привилегии доступа при открытии файла и не делают этого после. В результате пользователь может открыть файл и держать его открытым в течение недели и иметь к нему доступ, хотя владелец уже сменил защиту.

4) Давать каждому процессу минимум возможных привилегий.

5) Защитные механизмы должны быть просты, постоянны и встроены в нижний слой системы, это не аддитивные добавки.

6) Важна физиологическая приемлемость. Если пользователь видит, что защита требует слишком больших усилий, он от неё откажется. Ущерб от атаки и затраты на её предотвращение должны быть сбалансированы.

Приведённые соображения показывают необходимость продумывания и встраивания защитных механизмов на самых ранних стадиях проектирования системы.

56. Программирование для многопроцессорных структур

Мультипроцессорные комплексы позволяют воспользоваться преимуществами параллелизма. Вычислительные системы получают больше пользы от параллельной обработки благодаря мультипрограммному выполнению нескольких процессов, чем используя параллелизм в рамках одного процесса. Обнаружение параллелизма (распараллеливание), выполняемое программистами, языковыми трансляторами, аппаратными средствами или операционными системами — это сложная проблема. Независимо от того, каким образом в конце концов будет обнаруживаться параллелизм, мультипроцессорные

системы позволяют с успехом использовать его, одновременно выполняя параллельные ветви вычислений.

Параллелизм в программах может быть либо явным, либо неявным (скрытым). Явный параллелизм программист указывает в своей программе при помощи специальной конструкции, обозначающей параллельное вычисление, например COBEGIN/COEND.

В мультипроцессорной системе, рассчитанной на реализацию параллелизма, каждый из программных операторов может выполнять отдельный процессор, с тем, чтобы все вычисления завершались быстрее, чем при чисто последовательной работе.

Явное указание параллелизма налагает определённую ответственность на программиста. Это достаточно трудоёмкая процедура, причём в ряде случаев программист может ошибочно указать, что определённые операции можно выполнять параллельно, в то время как в действительности этого делать нельзя. Программист может упустить многие ситуации, допускающие распараллеливание. Весьма вероятно, что программист обнаружит параллелизм и закодирует его явным образом в наиболее очевидных ситуациях. Однако многие случаи параллелизма вручную в алгоритмах обнаружить трудно, поэтому они будут просто пропускаться.

Одно из довольно неприятных последствий явного указания параллелизма заключается в том, что программы могут оказаться более сложными для модификации. При внесении изменений в программу, имеющую большое число явных параллельных конструкций, легко можно допустить ошибки, причём, вообще говоря, довольно тонкие.

Реально, на что можно надеяться при решении подобной проблемы, — это на автоматическое обнаружение неявного параллелизма, т. е. параллелизма, присущего алгоритму, но не указанного явно программистом. В компиляторы, операционные системы и аппаратные средства компьютеров необходимо включать специальные механизмы распараллеливания. Это с гораздо большей вероятностью, чем явное указание параллелизма, обеспечит создание быстро выполняющихся и корректных программ.

Два распространённых способа, реализуемых в компиляторах для использования неявного параллелизма программ, — это расщепление цикла и редукция высоты дерева.

Применение указанных методов компиляции с целью оптимизации программ для выполнения на мультипроцессорных системах не обходится бесплатно. Дело в том, что за уменьшение количества

времени выполнения, затрачиваемого на данное вычисление, приходится платить увеличенными затратами времени и ресурсов в период компиляции. Такую взаимосвязь необходимо учитывать, тщательно оценивая необходимые затраты и возможные выгоды в каждом индивидуальном случае. Например, для производственного счета целесообразно добиваться минимального времени выполнения программ. Однако в условиях разработки, когда программа, возможно, будет выполняться всего один или два раза до внесения очередных изменений и повторной компиляции, затраты на оптимизацию могут значительно превысить получаемые выгоды.

В некоторых случаях целесообразно, чтобы процессоры выполняли вычисления, результаты которых могут так и не потребоваться, если в действительности существует вероятность того, что эти результаты все же будут использованы и ускорят вычисления. Лучше поручить некоторому процессору работу, которая то ли будет, то ли не будет использована в дальнейшем, чем оставить этот процессор бездействующим. Идея состоит в том, что если всё же результаты понадобятся, то вычисления можно будет выполнить гораздо быстрее.

57. Классификация многопроцессорных структур

Все многопроцессорные структуры можно разделить на два больших типа:

– с сильно связанными процессорами;

– со слабо связанными процессорами.

Среди структур с сильно связанными процессорами выделяют схемы:

– с общей шиной;

– с матрицей координатной коммутации;

– с многопортовой памятью.

В рамках этих структур вычислительный процесс может быть организован по принципу:

– главный — подчинённый;

– с выделенными мониторами;

– с симметричной организацией (SMP).

Структуры со слабо связанными процессорами предполагают наличие более-менее автономных вычислительных блоков, объединённых линиями связи. К таким структурам относят:

– локальные сети;

– региональные и глобальные сети;

– транспьютерные архитектуры;

– кластеры.

58. Мультипроцессорные операционные системы

Увеличение количества процессоров, а также усложнение связей с памятью и процессорами ввода-вывода значительно повышают стоимость аппаратуры комплекса. Поэтому многопроцессорная операционная система должна эффективно управлять дополнительными аппаратными средствами, с тем чтобы получаемые выгоды превосходили увеличенные исходные затраты.

Нельзя также игнорировать дополнительные затраты на программное обеспечение — построение мультипроцессорного вычислительного комплекса требует не только дополнительной аппаратуры, но и более сложной операционной системы.

Одно из основных различий между операционными системами мультипроцессорных и однопроцессорных вычислительных комплексов состоит в том, каким образом организуется и строится операционная система с учётом взаимодействия со многими процессорами. Существуют три основных варианта организации операционных систем для мультипроцессорных комплексов:

– главный — подчинённый;
– свой монитор в каждом процессоре;
– симметричная организация (процессоры идентичны).

Организацию главный — подчинённый реализовать легче всего, причём часто её можно создать просто путём расширения существующей мультипрограммной системы. Однако такая организация не обеспечивает оптимального использования аппаратуры комплекса.

При организации главный — подчинённый ОС выполняется только на одном конкретном процессоре, главном процессоре. На подчинённом процессоре (процессорах) могут выполняться только программы пользователей. Когда процесс на подчинённом процессоре требует внимания ОС, он генерирует сигнал и ждёт, чтобы главный процессор обработал его запрос. Если подчинённых процессоров много и они активно генерируют сигналы, то у главного процессора могут создаваться большие очереди.

Организация главный — подчинённый характеризуется меньшей надёжностью по сравнению с другими видами организации, поскольку выход главного процессора из строя вызывает катастрофический отказ всей системы.

При организации с раздельными мониторами (исполнительными программами) каждый процессор содержит собственную операционную систему, которая соответствующим образом реагирует запросы

от программ пользователей, работающих на этом процессоре. Поскольку некоторые таблицы содержат глобальную информацию для всей системы (например, список процессоров, известных системе), доступ к этим таблицам должен осуществляться под строгим контролем с применением методов взаимоисключения.

Организация с раздельными мониторами является в принципе более надёжной, чем организация главный — подчинённый. Отказ какого-то одного процессора здесь вряд ли станет катастрофическим отказом системы, однако рестарт системы с отказавшим процессором может оказаться достаточно сложным.

Каждый процессор управляет своими собственными ресурсами, например файлами и устройствами ввода-вывода. Реконфигурация оборудования ввода-вывода системы может потребовать подключения устройств ввода-вывода к другим процессорам с другими операционными системами. Такая процедура может быть достаточно сложной и потребовать значительных ручных усилий.

При организации с раздельными мониторами не предусматривается никакого взаимодействия процессоров при выполнении индивидуального процесса. Не исключается возможность, что некоторые из процессоров будут оставаться свободными, в то время как один процессор выполняет длинный процесс.

Симметричная организация мультипроцессорного вычислительного комплекса является наиболее сложной для реализации и в то же время наиболее эффективной. Здесь все процессоры идентичны. Операционная система управляет пулом идентичных процессоров, каждый из которых может управлять работой любого устройства ввода-вывода или обращаться к любому устройству памяти.

Поскольку программы операционной системы могут выполняться на многих процессорах одновременно, реентерабельный код и взаимоисключение являются для ОС обязательными. Благодаря симметричности системы имеется возможность более точно сбалансировать рабочую нагрузку, чем при других видах организации.

При симметричной организации особенно важное значение приобретают аппаратные и программные средства для разрешения конфликтных ситуаций. Конфликты между процессорами, пытающимися получить доступ к одной и той же памяти в одно и то же время, разрешаются, как правило, аппаратными средствами. Конфликты при доступе к системным таблицам разрешаются обычно программными средствами.

Симметричные мультипроцессорные комплексы являются в общем случае наиболее надёжными — отказ одного процессора приводит к тому, что операционная система исключает этот процессор из пула имеющихся процессоров и уведомляет об этом оператора. Комплекс может продолжать работать с несколько пониженным уровнем функциональных и скоростных возможностей (плавная деградация), пока вышедший из строя процессор не будет отремонтирован.

В симметричной системе процесс может в разные периоды времени выполняться на любом из эквивалентных процессоров. Все процессоры могут кооперироваться при выполнении конкретного процесса.

Сегодня многопроцессорная обработка реализована в таких ОС, как Linux, Solaris, Windows NT, и ряде других.

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

В сетевой операционной системе отдельной машины можно выделить несколько частей:

– 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 имён сетевых ресурсов; обработку запросов удалённого доступа к собственной файловой системе и базе данных; управление очередями запросов удалённых пользователей к своим периферийным устройствам;

– средства запроса доступа к удалённым ресурсам и услугам и их использования — клиентская часть ОС (редиректор). Эта часть

выполняет распознавание и перенаправление в сеть запросов к удалённым ресурсам от приложений и пользователей, при этом запрос поступает от приложения в локальной форме, а передаётся в сеть в другой форме, соответствующей требованиям сервера. Клиентская часть также осуществляет приём ответов от серверов и преобразование их в локальный формат, так что для приложения выполнение локальных и удалённых 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, надёжность передачи и т. п., то есть является средством транспортировки сообщений.

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

На практике сложилось несколько подходов к построению сетевых операционных систем.

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.

Однако более эффективным является 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.

60. Распределённые ОС

Распределённая система — совокупность независимых компьютеров, которая представляется пользователю единым компьютером, использование которого не намного сложнее, чем использование персональной ЭВМ.

Главная побудительная причина создания распределённых ОС — наличие огромного количества персональных компьютеров и необходимость совместной работы без ощущения неудобства от географического и физического распределения людей, данных и машин.

Сформулированы следующие принципы построения распределённых ОС.

1) Прозрачность (для пользователя и программы):

– прозрачность расположения — пользователь не должен знать, где расположены ресурсы;

– прозрачность миграции — ресурсы могут перемещаться без изменения их имён;

– прозрачность размножения — пользователь не должен знать, сколько копий ресурса существует;

– прозрачность конкуренции — множество пользователей разделяют ресурсы автоматически;

– прозрачность параллелизма — работа может распараллеливаться без участия пользователя.

2) Надёжность — доступность в любой момент времени, устойчивость к ошибкам, секретность (защищённость от вторжений).

3) Производительность — «мелкозернистый» и «крупнозернистый» параллелизм. Устойчивость к ошибкам требует дополнительных накладных расходов.

4) Масштабируемость.

В распределённых ОС применяются только децентрализованные алгоритмы со следующими чертами:

– ни одна машина не имеет полной информации о состоянии системы;

– машины принимают решения на основе только локальной информации;

– выход из строя одной машины не должен приводить к отказу алгоритма;

– не должно быть неявного предположения о существовании глобальных часов.

На данный момент распределённые ОС находятся в стадии проектирования и лабораторной отработки. Промышленного распространения пока не получили.

Список литературы

1. **Олифер В. Г., Олифер Н. А.** Сетевые операционные системы. — СПб.: Питер, 2001. — 544 с.
2. **Таненбаум Э.** Современные операционные системы. 2-е изд.: Пер. с англ. — СПб.: Питер, 2002. — 1040 с.
3. **Столлингс В.** Операционные системы. 4-е изд.: Пер. с англ. — М.: Издательский дом «Вильямс», 2002. — 848 с.

Анекдоты об экзаменах

* * *

Студент — объективно существующая реальность, плавающая на поверхности океана знаний.

* * *

СТУДЕНТ — Сонное, Теоретически Умное Дитя, Естественно, Не желающее Трудиться...

* * *

Студент прогулял — до экзамена осталась ночь. Никак не выучить 40 билетов.

— И не учи! — говорит внутренний голос. — Чувствую, что 26-й билет будет лежать крайним справа в верхнем ряду.

Студент доверился, выучил 26-й билет и лёг спать. Утром приходит на экзамен, берёт билет, который лежит крайним справа в верхнем ряду, глядь — а это № 31!

— Вот ни фига себе! — сказал внутренний голос.

* * *

На экзамене.

Профессор:

— Вы трое, прекратите передавать друг другу записки!

Студент:

— Это не записки, это мы в преферанс играем.

— Ну, тогда извините.

* * *

На экзаменах в разведшколу абитуриент Семёнов не ответил ни на один вопрос, в результате чего был принят сразу на второй курс.

* * *

— Пап, дай 500 рублей. — Зачем? — Да на четыре поменяю. — Это по какому курсу? — По философии.

* * *

Студент с преподавателем на экзамене.
Преподаватель:
— Ну, знаешь?
Студент:
— Знаю.
— Что знаешь? — Предмет знаю! — Какой предмет? — Который сдаю! — А какой сдаёшь? — Ну, это Вы уже придираетесь!

* * *

Студент сдаёт экзамен по физике. Сдаёт очень плохо. Профессор пытается его вытянуть, спрашивает:
— Ну скажите, хотя бы, при какой температуре кипит вода?
— Профессор, я не знаю при какой температуре вода кипит, но я знаю что при 40 градусах она превращается в водку!

* * *

Как сдают экзамены на мехмате?
Преподаватель:
— Столица Франции?
Студент:
— Париж!
Преподаватель:
— Молодец, пять!

А на юридическом?
Преподаватель:
— А не Париж ли столица Франции?
Студент:
— Париж!
Преподаватель:
— Молодец, пять!

А на военной кафедре?
Преподаватель:
— Париж — столица Франции!
Студент:
— Есть! Париж — столица Франции!
Преподаватель:
— Молодец, пять!

* * *

После экзамена студент поворачивается к своей соседке по столу и говорит:

— Я сдал совершенно пустой лист бумаги. Ни фига не смог вспомнить!

Девушка отвечает:

— Со мной то же самое. Чистый лист сдала. Надеюсь, преподаватель не подумает, что мы списали друг у друга.

* * *

То, что не понял на лекции, поймёшь на экзамене!

* * *

Политех. Экзамен по физике. Подходит очередь одной *ДЕВУШКИ*. Вместо вопроса профессор высыпал перед ней на стол кучку деталей и сказал:

— Резисторы — направо, конденсаторы — налево...

Дело кончилось пересдачей...

* * *

Преподаватель:

— Кто первым выйдет отвечать, тому оценку на балл выше.

Студент (вставая):

— Ладно, ставьте мне тройку, я ухожу...

* * *

— Профессор, а у меня будет автомат?

— Да, и кирзовые сапоги!!!

* * *

Возмутительный поступок совершил во время сессии студент-двоечник, фамилию которого даже не хочется называть. В отличие от успевающих студентов, он решил отблагодарить преподавателя не ценным подарком, а глубокими познаниями предмета...

* * *

Экзаменатор к студентам:

— А вы знаете, зачем я здесь сижу? Чтобы пополнить ряды Красной Армии.

* * *

Два студента перед сессией:

— Что читаешь?

— Квантовую механику.

— А чего книга вверх ногами?

— Да какая разница...

* * *

Отмазки для сессии (с переводом):

1) Я не уверен (я не знаю).

2) Не могу навскидку вспомнить (я не знаю).

3) Вчера у сестры была свадьба (я не знаю ничего).

4) Очевидно (я не знаю вывод).

5) Исходя из вышесказанного (вышесказанное здесь ни при чём).

6) Этого на лекциях не было (я на лекции не хожу).

7) Семестр был очень загруженный (вчера желание заглянуть в материал так и не появилось).

8) Я ещё и работаю (хочу зачёт нахаляву).

9) Было мало времени на подготовку (халява не прошла).

10) Мне нужно время, чтобы сосредоточиться (блин!! что делать?!).

11) Надо подумать (тянем время...).

12) М-М-М-М... (предлагаю забить на этот вопрос).

* * *

Почему-то фраза «профессор завалил студента на экзамене» никого не трогает, зато фраза «студент завалил профессора после экзамена» вызывает бурю эмоций.

* * *

Объявление у деканата: «Студенты, имеющие хвосты и не сдавшие языки, будут повешены на втором этаже.»

Содержание

Предисловие 3

Положение о проведении государственного экзамена по направлению «Информатика и вычислительная техника» 5

Вопросы к государственному экзамену по направлению «Информатика и вычислительная техника» 5

Конспект ответов на вопросы . . . .

Список литературы 83

Анекдоты об экзаменах 83

Методические указания

Государственный экзамен
по направлению 552800 (230100)
«Информатика и вычислительная техника»

Под редакцией составителя

Оригинал-макет выполнен в пакете teTEX с использованием кириллических шрифтов семейства LH.

Вёрстка в TEXe: А. В. Чернышов

По тематическому плану внутривузовских изданий учебной литературы на 2006 г., поз.

Лицензия ЛР № 020718 от 02.02.1998 г.
Лицензия ПД № 00326 от 14.02.2000 г.

Подписано к печати		Формат 60×88/16
Бумага 80 г/см^2 «Снегурочка»		Ризография
Объем 6 п. л.	Тираж 100 экз.	Зак. №

Издательство Московского государственного университета леса.
141005. Мытищи-5, Московская обл., 1-я Институтская, 1, МГУЛ.
Телефон: (095) 588-57-62
e-mail: izdat@mgul.ac.ru