1. Определение операционной системы (ОС).

Операционная система – это совокупность программ, обеспечивающая организацию вычислительного процесса на ЭВМ.

Операционная система позволяет запускать пользовательские программы; управляет всеми ресурсами компьютерной системы – процессором (процессорами), оперативной памятью, устройствами ввода вывода; обеспечивает долговременное хранение данных в виде файлов на устройствах внешней памяти; предоставляет доступ к компьютерным сетям.

Основные задачи ОС следующие:

– увеличение пропускной способности ЭВМ (за счет организации непрерывной обработки потока задач с автоматическим переходом от одной задачи к другой и эффективного распределения ресурсов ЭВМ по нескольким задачам);

– уменьшение времени реакции системы на запросы пользователей пользователями ответов на ЭВМ;

– упрощение работы разработчиков программных средств и сотрудников обслуживающего персонала ЭВМ (за счет предоставления им значительного количества языков программирования и разнообразных сервисных программ).

2. Назначение и основные функции ОС.

Основные функции ОС:

1. управление устройствами компьютера (ресурсами), т.е. согласованная работа всех аппаратных средств ПК: стандартизованный доступ к периферийным устройствам, управление оперативной памятью и др.
2. управление процессами, т.е. выполнение программ и их взаимодействие с устройствами компьютера.
3. управление доступом к данным на энергонезависимых носителях (таких как жесткий диск, компакт-диск и т.д.), как правило, с помощью файловой системы.
4. ведение файловой структуры.
5. пользовательский интерфейс, т.е. диалог с пользователем.

Дополнительные функции:

- параллельное или псевдопараллельное выполнение задач (многозадачность).

- взаимодействие между процессами: обмен данными, взаимная синхронизация.
- защита самой системы, а также пользовательских данных и программ от злонамеренных действий пользователей или приложений.
- разграничение прав доступа и многопользовательский режим работы (аутентификация, авторизация).

Состав операционной системы

В общем случае в состав ОС входят следующие модули:

- Программный модуль, управляющий файловой системой.
- Командный процессор, выполняющий команды пользователя.
- Драйверы устройств.
- Программные модули, обеспечивающие графический пользовательский интерфейс.
- Сервисные программы.
- Справочная система.
- Драйвер устройства (device driver) – специальная программа, обеспечивающая управление работой устройств и согласование информационного обмена с другими устройствами.

Командный процессор (command processor) – специальная программа, которая запрашивает у пользователя команды и выполняет их (интерпретатор программ).

Интерпретатор команд отвечает за загрузку приложений и управление информационным потоком между приложениями.

Для упрощения работы пользователя в состав современных ОС входят программные модули, обеспечивающие графический пользовательский интерфейс.

Процесс работы компьютера в определенном смысле сводится к обмену файлами между устройствами. В ОС имеется программный модуль, управляющий файловой системой.

Сервисные программы позволяют обслуживать диски (проверять, сжимать, дефрагментировать и др.), выполнять операции с файлами (копирование, переименование и др.), работать в компьютерных сетях.

Для удобства пользователя в состав ОС входит справочная система, позволяющая оперативно получить необходимую информацию о функционировании как ОС в целом, так и о работе ее отдельных модулей.

3. Основные понятия операционной системы.

Операцио́нная систе́ма — комплекс взаимосвязанных программ, предназначенных для управления ресурсами компьютера и организации взаимодействия с пользователем.

Пакетный режим

Необходимость оптимального использования дорогостоящих вычислительных ресурсов привела к появлению концепции «пакетного режима» исполнения программ. Пакетный режим предполагает наличие очереди программ на исполнение, причём система может обеспечивать загрузку программы с внешних носителей данных в оперативную память, не дожидаясь завершения исполнения предыдущей программы, что позволяет избежать простоя процессора.

Разделение времени и многозадачность

Разделение времени позволило создать «многопользовательские» системы, в которых один (как правило) центральный процессор и блок оперативной памяти соединялся с многочисленными терминалами. При этом часть задач (таких как ввод или редактирование данных оператором) могла исполняться в режиме диалога, а другие задачи (такие как массивные вычисления) — в пакетном режиме.

Разделение полномочий

Реализация разделения полномочий в операционных системах была поддержана разработчиками процессоров, предложивших архитектуры с двумя режимами работы процессора — «реальным» (в котором исполняемой программе доступно всё адресное пространство компьютера) и «защищённым» (в котором доступность адресного пространства ограничена диапазоном, выделенным при запуске программы на исполнение).

Масштаб реального времени

Применение универсальных компьютеров для управления производственными процессами потребовало реализации «масштаба реального времени» («реального времени») — синхронизации исполнения программ с внешними физическими процессами.

Включение функции масштаба реального времени позволило создавать решения, одновременно обслуживающие производственные процессы и решающие другие задачи (в пакетном режиме и/или в режиме разделения времени).

Файловые системы и структуры

Файловая система — способ хранения данных на внешних запоминающих устройствах.

4. Классификация ОС.

Операционная система составляет основу программного обеспечения ПК. Операционная система представляет комплекс системных и служебных программных средств, который обеспечивает взаимодействие пользователя с компьютером и выполнение всех других программ.

Операционные системы различаются особенностями реализации алгоритмов управления ресурсами компьютера, областями использования.

Так, в зависимости от алгоритма управления процессором, операционные системы делятся на:

1. Однозадачные и многозадачные.
2. Однопользовательские и многопользовательские.
3. Однопроцессорные и многопроцессорные системы.
4. Локальные и сетевые

По числу процессов, одновременно выполняемых под управлением системы:

· Однозадачные ОС поддерживают выполнение только одной программы в отдельный момент времени, то есть позволяют запустить одну программу в основном режиме.

· Многозадачные ОС (мультизадачные) поддерживают параллельное выполнение нескольких программ, существующих в рамках одной вычислительной системы на некотором отрезке времени, то есть позволяют запустить одновременно несколько программ, которые будут работать параллельно, не мешая друг другу.

При многозадачном режиме, в оперативной памяти находится несколько заданий пользователей, время работы процессора разделяется между программами, находящимися в оперативной памяти и готовыми к обслуживанию процессором, параллельно с работой процессора происходит обмен информацией с различными внешними устройствами.

Современные ОС поддерживают многозадачность, создавая иллюзию одновременной работы нескольких программ на одном процессоре. На самом деле за фиксированный период времени процессор обрабатывает только один процесс, а процессорное время делится между программами, организуя тем самым параллельную работу. Это замечание не относится к многопроцессорным системам, в которых в действительности в один момент времени могут выполняться несколько задач.

В зависимости от областей использования многозадачные ОС подразделяются на три типа:

1. Системы пакетной обработки (ОС ЕС).
2. Системы с разделением времени (Unix, Linux, Windows).
3. Системы реального времени (RT11).

Системы пакетной обработки предназначены для решения задач, которые не требуют быстрого получения результатов. Главной целью ОС пакетной обработки является максимальная пропускная способность или решение максимального числа задач в единицу времени.

Эти системы обеспечивают высокую производительность при обработке больших объемов информации, но снижают эффективность работы пользователя в интерактивном режиме.

В системах с разделением времени для выполнения каждой задачи выделяется небольшой промежуток времени, и ни одна задача не занимает процессор надолго. Если этот промежуток времени выбран минимальным, то создается видимость одновременного выполнения нескольких задач. Эти системы обладают меньшей пропускной способностью, но обеспечивают высокую эффективность работы пользователя в интерактивном режиме.

Системы реального времени применяются для управления технологическим процессом или техническим объектом, например, летательным объектом, станком и т.д

Одним из важнейших признаков классификации ЭВМ является разделение их на локальные и сетевые. Локальные ОС применяются на автономных ПК или ПК, которые используются в компьютерных сетях в качестве клиента.

В состав локальных ОС входит клиентская часть ПО для доступа к удаленным ресурсам и услугам. Сетевые ОС предназначены для управления ресурсами ПК включенных в сеть с целью совместного использования ресурсов. Они представляют мощные средства разграничения доступа к информации, ее целостности и другие возможности использования сетевых ресурсов.

5. Структура современных ОС.

Перед изучением структуры операционных систем следует рассмотреть режимы работы процессоров.

Современные процессоры имеют минимум два режима работы – привилегированный (supervisor mode) и пользовательский (user mode).

Отличие между ними заключается в том, что в пользовательском режиме недоступны команды процессора, связанные с управлением аппаратным обеспечением, защитой оперативной памяти, переключением режимов работы процессора. В привилегированном режиме процессор может выполнять все возможные команды.

Приложения, выполняемые в пользовательском режиме, не могут напрямую обращаться к адресным пространствам друг друга – только посредством системных вызовов.

Все компоненты операционной системы можно разделить на две группы – работающие в привилегированном режиме и работающие в пользовательском режиме, причем состав этих групп меняется от системы к системе.

Основным компонентом операционной системы является ядро (kernel). Функции ядра могут существенно отличаться в разных системах; но во всех системах ядро работает в привилегированном режиме (который часто называется режим ядра, kernel mode).

Термин "ядро" также используется в разных смыслах. Например, в Windows термин "ядро" (NTOS kernel) обозначает совокупность двух компонентов – исполнительной системы и собственно ядра.

Существует два основных вида ядер – монолитные ядра (monolithic kernel) и микроядра (microkernel). В монолитном ядре реализуются все основные функции операционной системы, и оно является, по сути, единой программой, представляющей собой совокупность процедур. В микроядре остается лишь минимум функций, который должен быть реализован в привилегированном режиме: планирование потоков, обработка прерываний, межпроцессное взаимодействие. Остальные функции операционной системы по управлению приложениями, памятью, безопасностью и пр. реализуются в виде отдельных модулей в пользовательском режиме.

Ядра, которые занимают промежуточные положение между монолитными и микроядрами, называют гибридными (hybrid kernel).

Примеры различных типов ядер:

– монолитное ядро – MS-DOS, Linux, FreeBSD;

– микроядро – Mach, Symbian, MINIX 3;

– гибридное ядро – NetWare, BeOS, Syllable.

Кроме ядра в привилегированном режиме (в большинстве операционных систем) работают драйверы (driver) – программные модули, управляющие устройствами.

В состав операционной системы также входят:

– системные библиотеки (system DLL – Dynamic Link Library, динамически подключаемая библиотека), преобразующие системные вызовы приложений в системные вызовы ядра;

– пользовательские оболочки (shell), предоставляющие пользователю интерфейс – удобный способ работы с операционной системой.

Пользовательские оболочки реализуют один из двух основных видов пользовательского интерфейса:

– текстовый интерфейс (Text User Interface, TUI), другие названия – консольный интерфейс (Console User Interface, CUI), интерфейс командной строки (Command Line Interface, CLI);

– графический интерфейс (Graphic User Interface, GUI).

6. Процессы и потоки.

Внутри каждого процесса могут выполняться один или несколько потоков, и именно поток является базовой единицей выполнения в Windows. Выполнение

потоков планируется системой на основе обычных факторов: наличие таких ресурсов, как CPU и физическая память, приоритеты, равнодоступность ресурсов и так далее. Начиная с версии NT4, в Windows поддерживается симметричная многопроцессорная обработка (Symmetric Multiprocessing, SMP), позволяющая распределять выполнение потоков между отдельными процессорами, установленными в системе.

С точки зрения программиста каждому процессу принадлежат ресурсы, представленные следующими компонентами:

• Одна или несколько потоков.
• Виртуальное адресное пространство, отличное от адресных пространств других процессов, если не считать областей памяти, распределенных явным образом для совместного использования (разделения) несколькими процессами. Заметьте, что разделяемые отображенные файлы совместно используют физическую память, тогда как разделяющие их процессы используют различные виртуальные адресные пространства.
• Один или несколько сегментов кода, включая код DLL.
• Один или несколько сегментов данных, содержащих глобальные переменные.
• Строки, содержащие информацию об окружении, например, информацию о текущем пути доступа к файлам.
• Куча процесса.
• Различного рода ресурсы, например, дескрипторы открытых файлов и другие кучи.

Поток разделяет вместе с процессом код, глобальные переменные, строки окружения и другие ресурсы. Каждый поток планируется независимо от других и располагает следующими элементами:
• Стек, используемый для вызова процедур, прерываний и обработчиков исключений, а также хранения автоматических переменных.
• Локальные области хранения потока (Thread Local Storage, SLT) — массивы указателей, используя которые каждый поток может создавать собственную уникальную информационную среду.
• Аргумент в стеке, получаемый от создающего потока, который обычно является уникальным для каждого потока.

• Структура контекста, поддерживаемая ядром системы и содержащая значения машинных регистров.

7. Состояние процесса.

1. Процесс выполняется в режиме задачи. При этом процессором выполняются прикладные инструкции данного процесса.
2. Процесс выполняется в режиме ядра. При этом процессором выполняются системные инструкции ядра операционной системы от имени процесса.
3. Процесс не выполняется, но готов к запуску, как только планировщик выберет его (состояние runnable). Процесс находится в очереди на выполнение и обладает всеми необходимыми ему ресурсами, кроме вычислительных.
4. Процесс находится в состоянии сна (asleep), ожидая недоступного в данный момент ресурса, например завершения операции ввода/вывода.
5. Процесс возвращается из режима ядра в режим задачи, но ядро прерывает его и производит переключение контекста для запуска более высокоприоритетного процесса.
6. Процесс только что создан вызовом *fork(2)* и находится в переходном состоянии: он существует, но не готов к запуску и не находится в состоянии сна.
7. Процесс выполнил системный вызов *exit(2)* и перешел в состояние зомби (zombie, defunct). Как такового процесса не существует, но остаются записи, содержащие код возврата и временную статистику его выполнения, доступную для родительского процесса. Это состояние является конечным в жизненном цикле процесса.

8. Операции над процессами.

(Я тут нашла видосик, можно его посмотреть и понятно всё *https://ru.coursera.org/lecture/os-v-razrabotke-po/opieratsii-nad-protsiessami-QTwiA**)***

Процесс самостоятельно не переходит из одного состояния в другое. Изменением состояния процессов занимается операционная система. Операции над процессами, производимые операционной системой удобно объединить в пары:

- *создание процесса – завершение процесса*;
- *приостановка процесса – запуск процесса*;
- *блокирование процесса – разблокирование процесса*;
- *изменение приоритета процесса*.

Операции *создание* и *завершение* процесса являются *одноразовыми*, так как применяются к процессу не более одного раза (некоторые системные процессы при работе вычислительной системы не завершаются никогда). Остальные операции, связанные с изменением состояния процессов, являются *многоразовыми*. Одноразовые операции приводят к изменению количества процессов, находящихся под управлением операционной системы и всегда связаны с выделением или освобождением определенных ресурсов. Многоразовые операции не приводят к изменению количества процессов и не обязаны быть связанными с выделением или освобождением ресурсов.

При операции запуск процесса операционная система из множества процессов, находящихся в состоянии «готовность», выбирает один. Выбор

осуществляется согласно заложенного в операционной системе алгоритма планирования.

Процесс, создавший новый процесс, называют *процессом родителем*, а созданный процесс – *потомком* (дочерним процессом или ребенком). Выделяют четыре основных события, которые приводят к созданию новых процессов:

- загрузка (инициализация) системы;
- выполнение изданного работающим процессом системного запроса на создание процесса;
- запрос пользователя на создание процесса;
- инициирование пакетного задания.

При загрузке операционной системы создаются несколько процессов, одни – *высокоприоритетные*, обеспечивающие взаимодействие с пользователем и выполняющие заданную работу, другие – *фоновые*, выполняющие особые функции. Например, один фоновый процесс обрабатывает приходящую электронную почту, активизируясь только при появлении писем, другой – обрабатывает запросы к *web*-страницам, активизируясь для обслуживания полученного запроса. Фоновые процессы, связанные с электронной почтой, *web*-страницами, новостями, выводом на печать и т.п. называются *демонами*.

9. Переключение контекста.

Переключение контекста – это процесс записи и восстановления состояния процесса или потока таким образом, чтобы в дальнейшем продолжить его выполнение с прерванного места. Этот механизм позволяет нескольким процессам разделить между собой ресурсы одного центрального процессора. Является особенностью многозадачных ОС.

Точное значение термина "переключение контекста" сильно зависит от области применения, чаще всего он означает "переключение потока или переключение процесса" или "только переключение процесса". Затраты ресурсов сильно варьируются в зависимости от того, что за собой влечет переключение контекста, от вызова подпрограммы для выполнения небольших пользовательских задач до запуска требовательного к ресурсам процесса.

В процесс переключения входит процедура планирования задачи – принятие решения, какой задаче передать управление.

При переключении контекста происходит сохранение и восстановление следующей информации:

- Регистровый контекст регистров общего назначения (в том числе флаговый регистр);
- Контекст состояния сопроцессора с плавающей точкой;
- Состояние регистров MMX/SSE (x86);
- Состояние сегментных регистров (x86);
- Состояние некоторых управляющих регистров (например, регистр CR3, отвечающий за страничное отображение памяти процесса) (x86).

Переключение контекста обычно требует больших вычислительных затрат, и основной этап при проектировании операционной системы заключается в оптимизации использования переключения контекста. Переход от одного процесса к другому требует определенного времени для сохранения состояния регистров, обновления различных таблиц и списков и т.д.

Выделяется три ситуации для переключения контекста:

- Истек квант времени.
- Поток с более высоким приоритетом стал готовым исполнить код.
- Запущенный поток должен ждать.

При переключении контекста состояние выполняемого в данный момент процесса должно быть каким-то образом сохранено, чтобы в дальнейшем это состояние могло быть восстановлено. Эта информация (состояние процесса) хранится в структуре данных, называемой блоком управления процессом (англ. process control block, PCB).

Блок управления процессом может храниться в стеке для каждого процесса в памяти ядра (в отличие от стека вызовов в пользовательском режиме) или в специально определенной операционной системой структуре данных. Дескриптор блока управления процессом добавляется в очередь готовых к выполнению процессов. Поскольку операционная система эффективно приостановила выполнение одного из процессов, она может затем переключать контекст, выбирая процесс из очереди готовых к выполнению процессов и восстанавливая его блок управления. При этом из блока управления запускается счетчик команд и может продолжаться

выполнение выбранного процесса. Такие параметры, как приоритеты процесса и потоков внутри него, будут влиять на то, какой процесс будет выбран из очереди.

10. Процессы в Unix/Linux.

Термин "процесс" впервые появился при разработке операционной системы Multix и имеет несколько определений, которые используются в зависимости от контекста. Процесс - это:

- программа на стадии выполнения
- "объект", которому выделено процессорное время
- асинхронная работа

Для описания состояний процессов используется несколько моделей. Самая простая модель - это модель трех состояний. Модель состоит из:

- состояния выполнения
- состояния ожидания
- состояния готовности

Выполнение - это активное состояние, во время которого процесс обладает всеми необходимыми ему ресурсами. В этом состоянии процесс непосредственно выполняется процессором.

Ожидание - это пассивное состояние, во время которого процесс заблокирован, он не может быть выполнен, потому что ожидает какое-то событие, например, ввода данных или освобождения нужного ему устройства.

Готовность - это тоже пассивное состояние, процесс тоже заблокирован, но в отличие от состояния ожидания, он заблокирован не по внутренним причинам (ведь ожидание ввода данных - это внутренняя, "личная" проблема процесса - он может ведь и не ожидать ввода данных и свободно выполняться - никто ему не мешает), а по внешним, независящим от процесса, причинам. Когда процесс может перейти в состояние готовности? Предположим, что наш процесс выполнялся до ввода данных. До этого момента он был в состоянии выполнения, потом перешел в состояние ожидания - ему нужно подождать, пока мы введем нужную для работы процесса информацию. Затем процесс хотел уже перейти в состояние выполнения, так как все необходимые ему данные уже введены, но не

тут-то было: так как он не единственный процесс в системе, пока он был в состоянии ожидания, его "место под солнцем" занято - процессор выполняет другой процесс. Тогда нашему процессу ничего не остается как перейти в состояние готовности: ждать ему нечего, а выполняться он тоже не может.

Из состояния готовности процесс может перейти только в состояние выполнения. В состоянии выполнения может находится только один процесс на один процессор. Если у вас n-процессорная машина, у вас одновременно в состоянии выполнения могут быть n процессов.

Из состояния выполнения процесс может перейти либо в состояние ожидания или состояние готовности. Почему процесс может оказаться в состоянии ожидания, мы уже знаем - ему просто нужны дополнительные данные или он ожидает освобождения какого-нибудь ресурса, например, устройства или файла. В состояние готовности процесс может перейти, если во время его выполнения, квант времени выполнения "вышел". Другими словами, в операционной системе есть специальная программа - планировщик, которая следит за тем, чтобы все процессы выполнялись отведенное им время. Например, у нас есть три процесса. Один из них находится в состоянии выполнения. Два других - в состоянии готовности. Планировщик следит за временем выполнения первого процесса, если "время вышло", планировщик переводит процесс 1 в состояние готовности, а процесс 2 - в состояние выполнения. Затем, когда, время отведенное, на выполнение процесса 2, закончится, процесс 2 перейдет в состояние готовности, а процесс 3 - в состояние выполнения.
Диаграмма модели трех состояний представлена на рисунке 1.

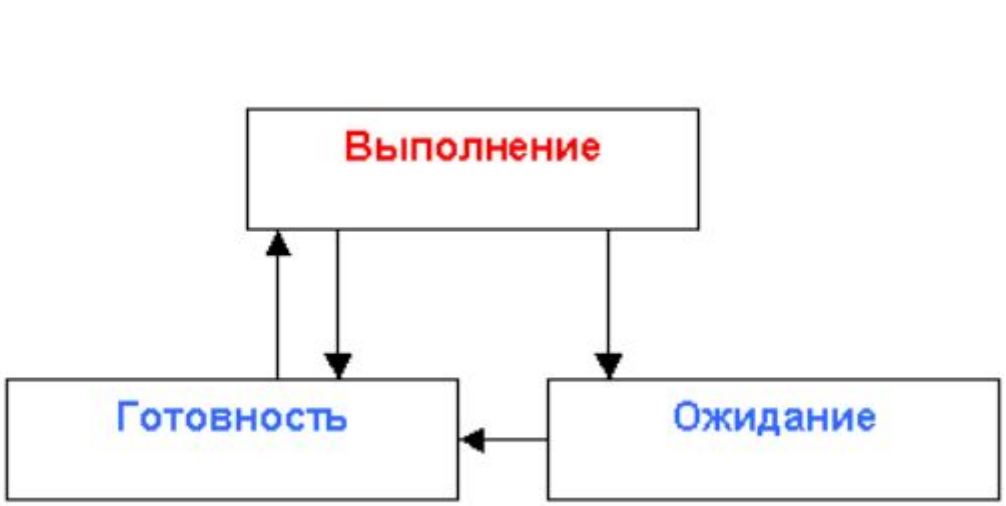


Рисунок 1. Модель трех состояний

Более сложная модель - это модель, состоящая из пяти состояний. В этой модели появилось два дополнительных состояния: рождение процесса и смерть процесса. Рождение процесса - это пассивное состояние, когда самого процесса еще нет, но уже готова структура для появления процесса. Как говорится в афоризме: "Мало найти хорошее место, надо его еще застолбить", так вот во время рождения как раз и происходит "застолбление" этого места. Смерть процесса - самого процесса уже нет, но может случиться, что его "место", то есть структура, осталась в списке процессов. Такие процессы называются зобми и о них мы еще поговорим в этой статье.

Диаграмма модели пяти состояний представлена на рисунке 2.

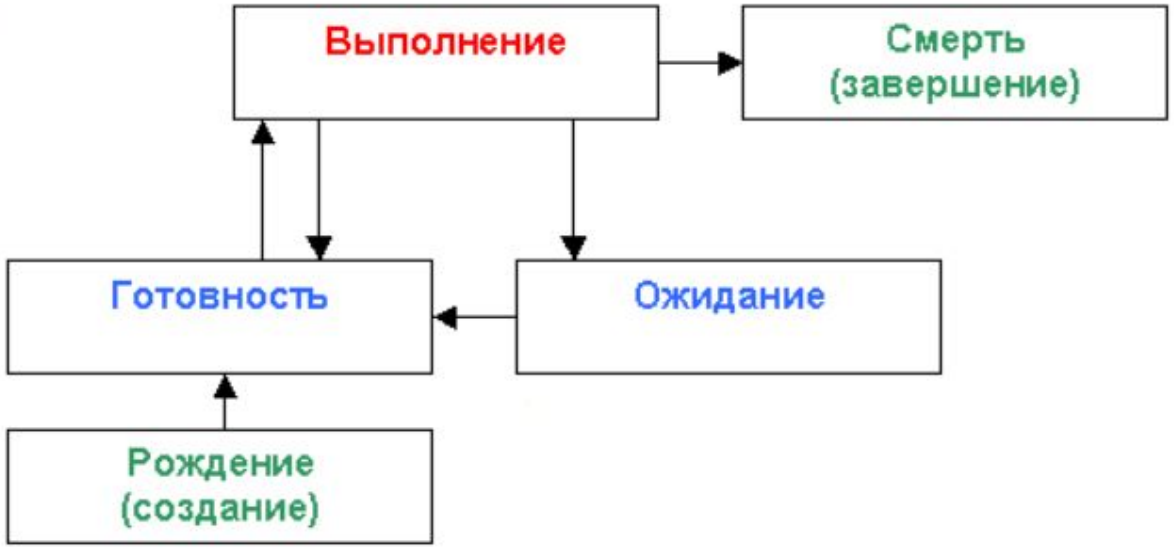


Рисунок 2. Модель пяти состояний

Над процессами можно производить следующие операции:

Создание процесса - это переход из состояния рождения в состояние готовности
Уничтожение процесса - это переход из состояния выполнения в состояние смерти
Восстановление процесса - переход из состояния готовности в состояние выполнения
Изменение приоритета процесса - переход из выполнения в готовность
Блокирование процесса - переход в состояние ожидания из состояния выполнения
Пробуждение процесса - переход из состояния ожидания в состояние готовности
Запуск процесса (или его выбор) - переход из состояния готовности в состояние выполнения
Для создания процесса операционной системе нужно:

Присвоить процессу имя
Добавить информацию о процессе в список процессов
Определить приоритет процесса
Сформировать блок управления процессом
Предоставить процессу нужные ему ресурсы
Подробнее о списке процессов, приоритете и обо всем остальном мы еще поговорим, а сейчас нужно сказать пару слов об иерархии процессов. Процесс не

может взяться из ниоткуда: его обязательно должен запустить какой-то процесс. Процесс, запущенный другим процессом, называется дочерним (child) процессом или потомком. Процесс, который запустил процесс называется родительским (parent), родителем или просто - предком. У каждого процесса есть два атрибута - PID (Process ID) - идентификатор процесса и PPID (Parent Process ID) - идентификатор родительского процесса.

Процессы создают иерархию в виде дерева. Самым "главным" предком, то есть процессом, стоящим на вершине этого дерева, является процесс init (PID=1).

11. Сигналы. Передача сигнала процессу.

Сигнал в операционных системах семейства Unix — асинхронное уведомление процесса о каком-либо событии, один из основных способов взаимодействия_между процессами. Когда сигнал послан процессу, операционная система прерывает выполнение процесса, при этом, если процесс установил собственный *обработчик сигнала*, операционная система запускает этот обработчик, передав ему информацию о сигнале, если процесс не установил обработчик, то выполняется обработчик по умолчанию.

Сигналы посылаются:

- из терминала, нажатием специальных клавиш или комбинаций (например, нажатие Ctrl-C генерирует SIGINT, Ctrl-\ SIGQUIT, а Ctrl-Z SIGTSTP);
- ядром системы:
  - при возникновении аппаратных исключений (недопустимых инструкций, нарушениях при обращении в память, системных сбоях и т. п.);
  - ошибочных системных вызовах;
  - для информирования о событиях ввода-вывода;

- одним процессом другому (или самому себе), с помощью системного вызова kill(), в том числе:
    - из shell, утилитой /bin/kill.

Сигналы не могут быть посланы завершившемуся процессу, находящемуся в состоянии «зомби».

12. Адресное пространство процесса.

(и тут видосик https://ru.coursera.org/lecture/os-v-razrabotke-po/adriesnoie-prostranstvo-protsiessa-5y17q**)**

Адресное пространство — совокупность всех допустимых адресов каких-либо объектов вычислительной системы — ячеек памяти, секторов диска, узлов сети и т. п., которые могут быть использованы для доступа к этим объектам при определенном режиме работы (состоянии системы).

Адресное пространство процесса состоит из диапазона адресов, которые выделены процессу, и, что более важно, в этом диапазоне выделяются адреса, которые процесс может так или иначе использовать. Каждому процессу выделяется "плоское" 32- или 64-битовое адресное пространство. Термин "плоское" обозначает, что адресное пространство состоит из одного диапазона адресов (например, 32-разрядное адресное пространство занимает диапазон адресов от 0 до 429496729).

Обычно для каждого процесса существует свое адресное пространство. Адрес памяти в адресном пространстве одного процесса не имеет никакого отношения к такому же адресу памяти в адресном пространстве другого процесса. Тем не менее несколько процессов могут совместно использовать одно общее адресное пространство. Такие процессы называются потоками.

13. Планирование процессов.

Система управления процессами обеспечивает прохождение процесса через компьютер. В зависимости от состояния процесса ему должен быть предоставлен тот или иной ресурс. Например, новый процесс необходимо разместить в основной памяти, следовательно, ему необходимо выделить часть адресного пространства. Процессу в состоянии «готовый» должно быть

предоставлено процессорное время. Выполняемый процесс может потребовать оборудование ввода-вывода и доступ к файлу.

Распределение процессов между имеющимися ресурсами носит название планирование процессов. Одним из методов планирования процессов, ориентированных на эффективную загрузку ресурсов, является метод очередей ресурсов. Новые процессы находятся во входной очереди, часто называемой очередью работ — заданий. Входная очередь располагается во внешней памяти, во входной очереди процессы ожидают освобождения ресурса — адресного пространства основной памяти. Готовые к выполнению процессы располагаются в основной памяти и связаны очередью готовых процессов. Процессы в этой очереди ожидают освобождения ресурса процессорное время. Процесс в состоянии ожидания завершения операции ввода-вывода находится в одной из очередей к оборудованию ввода-вывода. При прохождении через компьютер процесс мигрирует между различными очередями под управлением программы, которая называется планировщик (scheduler). Операционная система, обеспечивающая режим мультипрограммирования, обычно включает два планировщика — долгосрочный и краткосрочный. На уровень долгосрочного планирования выносятся редкие системные действия, требующие больших затрат системных ресурсов, на уровень краткосрочного планирования — частые и более короткие процессы. На каждом уровне существует свой объект и собственные средства управления им. Основное различие между долгосрочным и краткосрочным планировщиками заключается в частоте запуска, например, краткосрочный планировщик может запускаться каждые 100 мс, долгосрочный — 1 раз за несколько минут.

Долгосрочный планировщик решает, какой из процессов, находящихся во входной очереди, должен быть переведен в очередь готовых процессов в случае освобождения ресурсов памяти. Долгосрочный планировщик выбирает процесс из входной очереди с целью создания неоднородной мультипрограммной смеси. Это означает, что в очереди готовых процессов должны находиться в разной пропорции как процессы, ориентированные на ввод-вывод, так и процессы, ориентированные на преимущественную работу с CPU.

На уровне долгосрочного планирования объектом является не отдельный процесс, а некоторое объединение процессов по функциональному назначению, которое называется работой (приложением). Каждая работа рассматривается как независимая от других работ деятельность, связанная с использованием одной

или многих программ и характеризующаяся конечностью и определенностью. По мере порождения новых работ создается собственная виртуальная машина для их выполнения. Например, в ОС Windows 95 для каждого 32-разрядного приложения реализуется своя виртуальная машина. Распределение машин производится однократно в отличие от краткосрочного планирования, где процессор процессу может выделяться многократно.

Краткосрочный планировщик решает, какой из процессов, находящихся в очереди готовых процессов, должен быть передан на выполнение в CPU. В некоторых операционных системах долгосрочный планировщик может отсутствовать. Например, в системах разделения времени (time-sharing system) каждый новый процесс сразу же помещается в основную память. На уровне краткосрочного планирования объектом управления являются процессы, которые выступают как потребители центрального процессора для внутренних процессов или внешнего процессора для внешних процессов. Причинами порождения процесса могут быть процессы на том же уровне или сигналы, посылаемые от долгосрочного планировщика. Выделение процессора процессу производится многократно, с целью достижения эффекта мультипрограммирования, и такой процесс называется диспетчеризацией.

14. Вытесняющее и невытесняющее планирование.

Процесс планирования осуществляется частью операционной системы, называемой планировщиком. Планировщик может принимать решения о выборе для исполнения нового процесса из числа находящихся в состоянии готовность в следующих четырех случаях.

- Когда процесс переводится из состояния исполнение в состояние закончил исполнение.
- Когда процесс переводится из состояния исполнение в состояние ожидание.
- Когда процесс переводится из состояния исполнение в состояние готовность (например, после прерывания от таймера).

- Когда процесс переводится из состояния ожидание в состояние готовность (завершилась операция ввода-вывода или произошло другое событие).

В случаях 1 и 2 процесс, находившийся в состоянии исполнение, не может дальше исполняться, и операционная система вынуждена осуществлять планирование выбирая новый процесс для выполнения. В случаях 3 и 4 планирование может как проводиться, так и не проводиться, планировщик не вынужден обязательно принимать решение о выборе процесса для выполнения, процесс, находившийся в состоянии исполнение может просто продолжить свою работу. Если в операционной системе планирование осуществляется только в вынужденных ситуациях, говорят, что имеет место невытесняющее (nonpreemptive) планирование. Если планировщик принимает и вынужденные, и невынужденные решения, говорят о вытесняющем (preemptive) планировании. Термин "вытесняющее планирование" возник потому, что исполняющийся процесс помимо своей воли может быть вытеснен из состояния исполнение другим процессом.

Невытесняющее планирование используется, например, в MS Windows 3.1 и ОС Apple Macintosh. При таком режиме планирования процесс занимает столько процессорного времени, сколько ему необходимо. При этом переключение процессов возникает только при желании самого исполняющегося процесса передать управление (для ожидания завершения операции ввода-вывода или по окончании работы). Этот метод планирования относительно просто реализуем и достаточно эффективен, так как позволяет выделить большую часть процессорного времени для работы самих процессов и до минимума сократить затраты на переключение контекста. Однако при невытесняющем планировании возникает проблема возможности полного захвата процессора одним процессом, который вследствие каких-либо причин (например, из-за ошибки в программе) зацикливается и не может передать управление другому процессу. В такой ситуации спасает только перезагрузка всей вычислительной системы.

Вытесняющее планирование обычно используется в системах разделения времени. В этом режиме планирования процесс может быть приостановлен в любой момент исполнения. Операционная система устанавливает специальный таймер для генерации сигнала прерывания по истечении некоторого интервала

времени – кванта. После прерывания процессор передается в распоряжение следующего процесса. Временные прерывания помогают гарантировать приемлемое время отклика процессов для пользователей, работающих в диалоговом режиме, и предотвращают "зависание" компьютерной системы из-за зацикливания какой-либо программы.

15. Алгоритмы планирования.

FCFS (first come – first serve; первым пришел – первым обслуживается) - Процессы ставятся в очередь по мере поступления.

RR (round robin – карусель) – алгоритм с вытеснением. Каждому процессу выделяется некоторый квант времени на выполнения, а затем происходит переход к другому процессу. Остальные процессы ожидают.

SJF (Shortest-Job-First) – Сначала выполняются самые быстрые процессы. Может быть вытесняющим и невытесняющим. При вытесняющем планировании при поступлении учитывается появление новых процессов в очереди. При невытесняющем процессу отдаётся все процессорное время.

Алгоритмы SJF и гарантированного планирования представляют собой частные случаи приоритетного планирования. При приоритетном планировании каждому процессу присваивается определенное числовое значение – приоритет, в соответствии с которым ему выделяется процессор.

Планирование с использованием приоритетов может быть как вытесняющим, так и невытесняющим. При вытесняющем планировании процесс с более высоким приоритетом, появившийся в очереди готовых процессов, вытесняет исполняющийся процесс с более низким приоритетом. В случае невытесняющего планирования он просто становится в начало очереди готовых процессов.

16. Межпроцессное взаимодействие.

Межпроцессное взаимодействие (*Inter-process communication (IPC)*) — это набор методов для обмена данными между потоками процессов. Процессы могут быть запущены как на одном и том же компьютере, так и на разных, соединенных сетью. IPC бывают нескольких типов: «сигнал», «сокет», «семафор», «файл», «сообщение»…

Из механизмов, предоставляемых ОС и используемых для IPC, можно выделить:

· механизмы обмена сообщениями;
· механизмы синхронизации;
· механизмы разделения памяти;
· механизмы удалённых вызовов (RPC).

В идеальной многозадачной системе все процессы, выполняющиеся параллельно, независимы друг от друга (т.е., асинхронны). На практике такая ситуация маловероятна, поскольку рано или поздно возникают ситуации, когда параллельным процессам необходим доступ к некоторым общим ресурсам. Для этого необходимо введение на уровне ОС средств, предоставляющих такую возможность.

При выполнении параллельных процессов может возникать проблема, когда каждый процесс, обращающийся к разделяемым данным, исключает для всех других процессов возможность одновременного с ним обращения к этим данным - это называется взаимоисключением (mutual exclusion).

Ресурс, который допускает обслуживание только одного пользователя за один раз, называется критическим ресурсом. Если несколько процессов хотят пользоваться критическим ресурсом в режиме разделения времени, им следует синхронизировать свои действия таким образом, чтобы этот ресурс всегда находился в распоряжении не более чем одного их них.

Участки процесса, в которых происходит обращение к критическим ресурсам, называются критическими участками. Для организации коммуникации между одновременно работающими процессами применяются средства межпроцессного взаимодействия (Interprocess Communication - IPC).

Выделяются три уровня средств IPC:

· локальный;
· удаленный;
· высокоуровневый.

Средства локального уровня IPC привязаны к процессору и возможны только в пределах компьютера. Коммуникационное пространство этих IPC, поддерживаются только в пределах локальной системы. Из-за этих ограничений для них могут реализовываться более простые и более быстрые интерфейсы.

Удаленные IPC предоставляют механизмы, которые обеспечивают взаимодействие как в пределах одного процессора, так и между программами на

различных процессорах, соединенных через сеть. Сюда относятся удаленные вызовы процедур (Remote Procedure Calls - RPC), сокеты Unix, а также TLI (Transport Layer Interface - интерфейс транспортного уровня) фирмы Sun.

Под высокоуровневыми IPC обычно подразумеваются пакеты программного обеспечения, которые реализуют промежуточный слой между системной платформой и приложением. Эти пакеты предназначены для переноса уже испытанных протоколов коммуникации приложения на более новую архитектуру. Простые межпроцессные коммуникации можно организовать с помощью сигналов и каналов. Более сложными средствами IPC являются очереди сообщений, семафоры и разделяемые области памяти

17. Категории средств обмена информацией.

Процессы могут взаимодействовать друг с другом, только обмениваясь информацией. *По* объему передаваемой информации и степени возможного воздействия на поведение другого процесса все средства такого обмена можно разделить на три категории.

- *Сигнальные*. Передается минимальное количество информации – один бит, "да" или "нет". Используются, как правило, для извещения процесса о наступлении какого-либо события. Степень воздействия на поведение процесса, получившего информацию, минимальна. Все зависит от того, знает ли он, что означает полученный сигнал, надо ли на него реагировать и каким образом. Неправильная реакция на сигнал или его игнорирование могут привести к трагическим последствиям
- *Канальные*. "Общение" процессов происходит через линии связи, предоставленные операционной системой, и напоминает общение людей по телефону, с помощью записок, писем или объявлений. Объем передаваемой информации в единицу времени ограничен пропускной способностью линий связи. С увеличением количества информации возрастает и возможность влияния на поведение другого процесса.
- *Разделяемая память*. Два или более процессов могут совместно использовать некоторую область адресного пространства. Созданием *разделяемой памяти* занимается операционная система (если, конечно, ее об этом попросят). "Общение" процессов напоминает совместное проживание студентов в одной комнате общежития. Возможность обмена информацией максимальна, как, впрочем, и влияние на поведение другого процесса, но

требует повышенной осторожности. Использование *разделяемой памяти* для передачи/получения информации осуществляется с помощью средств обычных языков программирования, в то время как *сигнальным* и *канальным средствам коммуникации* для этого необходимы специальные системные вызовы. *Разделяемая память* представляет собой наиболее быстрый способ взаимодействия процессов в одной вычислительной системе.

18. Состязательные ситуации.

Некоторые операционные системы могут предоставлять для чтения и записи некие общие хранилища данных для совместно работающих процессов. Такое хранилище может либо размещаться в оперативной памяти, как структура данных ядра, либо быть представленным в виде общего файла. В качестве примера взаимодействия процессов рассмотрим простой пример — спулер печати, который присутствует практически в каждой операционной системе. При возникновении необходимости печати некоторого файла процесс помещает имя данного файла в специальный каталог спулера.

Другой процесс, который именуется демоном принтера, через определенные интервалы времени проверяет наличие файлов для печати. Когда такие файлы в спулере имеются, демон принтера печатает их и затем удаляет их имена из каталога спулера.

Представим, что в каталоге спулера есть большое количество областей памяти, пронумерованных с 0. Каждая такая область может хранить имя файла, подготовленного для печати. Кроме этого, есть также общие переменные: out, которая указывает на следующий файл для печати, и in, которая указывает на следующую свободную для записи область в каталоге спулера. Допустим возникновение следующей ситуации, что в некоторый момент времени области от 0 до 3 не заняты, и процессы А и Б одновременно решают поставить свои файлы в очередь на печать. Допустим, что процесс А считывает значение переменной in и сохраняет значение 7 в локальной переменной next_ free_slot, содержащей индекс следующей свободной области. После чего происходит прерывание по таймеру, и центральный процессор переключается на выполнение процесса Б. Процесс Б также считывает значение 7 из переменной

in и сохраняет его уже в своей локальной переменной next_ free_slot. К данному моменту у каждого процесса имеется информация о том, что следующей доступной областью для записи имени файла будет область с индексом 7.

Затем процесс Б сохраняет имя своего файла в область 7 и присваивает переменной in значение 8, после чего возвращается к выполнению других не связанных со спулером действий. Очевидно, что в некоторый момент выполнение процесса А возобновится, причем с того места, где он был приостановлен. Процесс А считывает значение переменной next_free_slot, в которой находится число 7 и записывает в область 7 имя своего файла, тем самым затирая имя файла процесса Б. Затем он также инкрементирует переменную next_free_slot, получая значение 8 и присваивая его переменной in. Демон печати не заметит никаких нестыковок в каталоге спулера, но в описанной ситуации файл процесса Б не получит вывода на печать.

Состязательной ситуацией называется такая ситуация, при которой два или более процесса работают с общими данными, а итоговый результат такой работы будет зависеть от того, какой процесс будет выполняться, в какой конкретный момент времени и в какую очередь он будет выполняться.

19. Критические области.

> Критическая секция (англ. *critical section*) — объект синхронизации потоков, позволяющий предотвратить одновременное выполнение некоторого набора операций (обычно связанных с доступом к данным) несколькими потоками. Критическая секция выполняет те же задачи, что и мьютекс.
>
> Отличие от mutex в том, что mutex работает на уровне ядра. Для windows в c++ объект называется CRITICAL_SECTION. Для linux – pthread_mutex_t;

20. Алгоритмы режима взаимного исключения.

Системы, состоящие из нескольких процессов, часто легче программировать, используя так называемые критические секции.

Когда процессу нужно читать или модифицировать некоторые разделяемые структуры данных, он, прежде всего, *входит в критическую секцию* для того, чтобы *обеспечить себе исключительное право использования этих данных*, при этом он уверен, что никакой процесс не будет иметь доступа к этому ресурсу одновременно с ним.

Это называется взаимным исключением.

В однопроцессорных системах критические секции защищаются семафорами, мониторами и другими аналогичными конструкциями. Рассмотрим, какие алгоритмы могут быть использованы в распределенных системах

Централизованный алгоритм

Наиболее очевидный и простой путь реализации взаимного исключения в распределенных системах - это применение тех же методов, которые используются в однопроцессорных системах.

Один из процессов выбирается в качестве координатора(например, процесс, выполняющийся на машине, имеющей наибольшее значение сетевого адреса).

1. Когда какой-либо процесс хочет войти в критическую секцию, он посылает сообщение с запросом к координатору, оповещая его о том, в какую критическую секцию он хочет войти, и ждет от координатора разрешение.
2. Если в этот момент ни один из процессов не находится в критической секции, то координатор посылает ответ с разрешением.
3. Если же некоторый процесс уже выполняет критическую секцию, связанную с данным ресурсом, то никакой ответ не посылается; запрашивавший процесс ставится в очередь, и после освобождения критической секции ему отправляется ответ-разрешение.

Этот алгоритм гарантирует взаимное исключение, но вследствие своей централизованной природы обладает низкой отказоустойчивостью.

Распределенный алгоритм

Когда процесс хочет войти в критическую секцию, он формирует сообщение, содержащее

- имя нужной ему критической секции,
- номер процесса и
- текущее значение времени.

Затем он посылает это сообщение всем другим процессам.

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

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

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

(Тут больше алгоритмов http://citforum.ru/operating_systems/sos/glava_13.shtml)

21. Механизмы синхронизации процессов.

Синхронизация процессов — приведение двух или нескольких процессов к такому их протеканию, когда определённые стадии разных процессов совершаются в определённом порядке, либо одновременно.

Синхронизация необходима в любых случаях, когда параллельно протекающим процессам необходимо взаимодействовать. Для её организации используются средства межпроцессного взаимодействия. Среди наиболее часто используемых средств — сигналы и сообщения, семафоры и мьютексы, каналы (англ. pipe), совместно используемая память.

Важным понятием при изучении способов синхронизации процессов является понятие критической секции (critical section) программы. Критическая секция – это часть программы, исполнение которой может привести к возникновению race condition для определенного набора программ. Чтобы исключить эффект гонок по отношению к некоторому ресурсу, необходимо организовать работу так, чтобы в каждый момент времени только один процесс мог находиться в своей критической секции, связанной с этим ресурсом. Иными словами, необходимо обеспечить реализацию взаимоисключения для критических секций программ. Реализация взаимоисключения для критических секций программ с практической точки зрения означает, что по отношению к другим процессам, участвующим во взаимодействии, критическая секция начинает выполняться как атомарная операция.

Рассмотрим самые простые механизмы синхронизации.

Семафор — примитив синхронизации работы процессов и потоков, в основе которого лежит счётчик, над которым можно производить две атомарные операции: увеличение и уменьшение значения на единицу. Служит для построения более сложных механизмов синхронизации и используется для синхронизации параллельно работающих задач, для защиты передачи данных через разделяемую память, для защиты критических секций, а также для управления доступом к аппаратному обеспечению.

Семафор представляет собой целую переменную, принимающую неотрицательные значения, доступ любого процесса к которой, за исключением момента ее инициализации, может осуществляться только через две атомарные операции: P и V. Классическое определение этих операций выглядит следующим образом:

P(S): пока S == 0 процесс блокируется;

S = S – 1;

V(S): S = S + 1;

Эта запись означает следующее: при выполнении операции P над *семафором* S сначала проверяется его значение. Если оно больше 0, то из S вычитается 1. Если оно меньше или равно 0, то процесс блокируется до тех пор, пока S не станет больше 0, после чего из S вычитается 1. При выполнении операции V над *семафором* S к его значению просто прибавляется 1. В момент создания *семафор* может быть инициализирован любым неотрицательным значением.

В двоичных семафорах S может принимать только значения 0 и 1 и используются для взаимного исключения одновременного нахождения двух или более процессов в своих критических областях.

Мьютексы — упрощённая реализация семафоров, аналогичная двоичным семафорам с тем отличием, что мьютексы должны отпускаться тем же процессом или потоком, который осуществляет их захват. Наряду с двоичными семафорами используются в организации критических участков кода. В отличие от двоичных семафоров, начальное состояние мьютекса не может быть захваченным и они могут поддерживать наследование приоритетов.

Другие механизмы.

Монитор обладает собственными переменными, определяющими его состояние. Значения этих переменных извне могут быть изменены только с помощью вызова функций-методов, принадлежащих монитору. В свою очередь, эти функции-методы могут использовать в работе только данные, находящиеся внутри монитора, и свои параметры.

Сообщения.

В качестве примера, две примитивные команды send/receive. Send – послать сообщение процессу. Receive – получить сообщение от процесса.

22. Объекты синхронизации.

Все нити, принадлежащие одному процессу, разделяют некоторые общие ресурсы - такие, как адресное пространство оперативной памяти или открытые файлы. Эти ресурсы принадлежат всему процессу, а значит, и каждой его нити. Следовательно, каждая нить может работать с этими ресурсами без каких-либо ограничений. Но... Если одна нить еще не закончила работать с каким-либо общим ресурсом, а система переключилась на другую нить, использующую этот же ресурс, то результат работы этих нитей может чрезвычайно сильно отличаться от задуманного. Такие конфликты могут возникнуть и между нитями, принадлежащими различным процессам. Всегда, когда две или более нитей используют какой-либо общий ресурс, возникает эта проблема.

Именно поэтому необходим механизм, позволяющий потокам согласовывать свою работу с общими ресурсами. Этот механизм получил название механизма синхронизации нитей (thread synchronization).

Этот механизм представляет собой набор объектов операционной системы, которые создаются и управляются программно, являются общими для всех нитей в системе (некоторые - для нитей, принадлежащих одному процессу) и используются для координирования доступа к ресурсам. В качестве ресурсов может выступать все, что может быть общим для двух и более нитей - файл на диске, порт, запись в базе данных, объект GDI, и даже глобальная переменная программы (которая может быть доступна из нитей, принадлежащих одному процессу).

Объектов синхронизации существует несколько, самые важные из них - это взаимоисключение (mutex), критическая секция (critical section), событие (event) и семафор (semaphore). Каждый из этих объектов реализует свой способ синхронизации. Также в качестве объектов синхронизации могут использоваться сами процессы и нити (когда одна нить ждет завершения другой нити или процесса); а также файлы, коммуникационные устройства, консольный ввод и уведомления об изменении.

Любой объект синхронизации может находиться в так называемом сигнальном состоянии. Для каждого типа объектов это состояние имеет различный смысл. Нити могут проверять текущее состояние объекта и/или ждать изменения этого состояния и таким образом согласовывать свои действия. При этом гарантируется, что когда нить работает с объектами синхронизации (создает их, изменяет состояние) система не прервет ее выполнения, пока она не завершит это действие. Таким образом, все конечные операции с объектами синхронизации являются атомарными (неделимыми).
Объект-критическая секция помогает программисту выделить участок кода, где нить получает доступ к разделяемому ресурсу, и предотвратить одновременное использование ресурса. Перед использованием ресурса нить входит в критическую секцию (вызывает функцию EnterCriticalSection). Если после этого какая-либо другая нить попытается войти в ту же самую критическую секцию, ее выполнение приостановится, пока первая нить не покинет секцию с помощью вызова LeaveCriticalSection. Используется только для нитей одного процесса. Порядок входа в критическую секцию не определен.

Существует также функция TryEnterCriticalSection, которая проверяет, занята ли критическая секция в данный момент. С ее помощью нить в процессе ожидания доступа к ресурсу может не блокироваться, а выполнять какие-то полезные действия.

Объекты-взаимоисключения (мьютексы, mutex - от MUTual EXclusion) позволяют координировать взаимное исключение доступа к разделяемому ресурсу. Сигнальное состояние объекта (т.е. состояние "установлен") соответствует моменту времени, когда объект не принадлежит ни одной нити и его можно "захватить". И наоборот, состояние "сброшен" (не сигнальное) соответствует моменту, когда какая-либо нить уже владеет этим объектом. Доступ к объекту разрешается, когда нить, владеющая объектом, освободит его.
Две (или более) нити могут создать мьютекс с одним и тем же именем, вызвав функцию CreateMutex. Первая нить действительно создает мьютекс, а следующие - получают дескриптор уже существующего объекта. Это дает возможность нескольким нитям получить дескриптор одного и того же мьютекса, освобождая программиста от необходимости заботиться о том, кто в действительности создает мьютекс. Если используется такой подход, желательно установить флаг bInitialOwner в FALSE, иначе возникнут определенные трудности при определении действительного создателя мьютекса.

Объекты-события используются для уведомления ожидающих нитей о наступлении какого-либо события. Различают два вида событий - с ручным и автоматическим сбросом. Ручной сброс осуществляется функцией ResetEvent. События с ручным сбросом используются для уведомления сразу нескольких нитей. При использовании события с автосбросом уведомление получит и продолжит свое выполнение только одна ожидающая нить, остальные будут ожидать дальше.
Функция CreateEvent создает объект-событие, SetEvent - устанавливает событие в сигнальное состояние, ResetEvent - сбрасывает событие. Функция PulseEvent устанавливает событие, а после возобновления ожидающих это событие нитей (всех при ручном сбросе и только одной при автоматическом), сбрасывает его. Если ожидающих нитей нет, PulseEvent просто сбрасывает событие.

Объект-семафор - это фактически объект-взаимоисключение со счетчиком. Данный объект позволяет "захватить" себя определенному количеству нитей. После этого "захват" будет невозможен, пока одна из ранее "захвативших" семафор нитей не освободит его. Семафоры применяются для ограничения количества нитей, одновременно работающих с ресурсом. Объекту при инициализации передается максимальное число нитей, после каждого "захвата" счетчик семафора уменьшается. Сигнальному состоянию соответствует значение счетчика больше нуля. Когда счетчик равен нулю, семафор считается не установленным (сброшенным).
Функция CreateSemaphore создает объект-семафор с указанием и максимально возможного начального его значения, OpenSemaphore – возвращает дескриптор существующего семафора, захват семафора производится с помощью ожидающих функций, при этом значение семафора уменьшается на единицу, ReleaseSemaphore - освобождение семафора с увеличением значения семафора на указанное в параметре число.

23. Механизмы межпроцессного взаимодействия в ОС Unix/Linux.

1. Канал (pipe) представляет собой средство связи стандартного вывода одного процесса со стандартным вводом другого. Когда процесс создает канал, ядро устанавливает два файловых дескриптора для пользования этим каналом. Один такой дескриптор используется, чтобы открыть путь ввода в канал (запись), в то время как другой применяется для получения данных из канала (чтение).
2. Именованные каналы во многом работают так же, как и обычные каналы, но все же имеют несколько заметных отличий:
    a. Именованные каналы существуют в виде специального файла устройства в файловой системе.
    b. Процессы различного происхождения могут разделять данные через такой канал.

c. Именованный канал остается в файловой системе для дальнейшего использования и после того, как весь ввод/вывод сделан.

3. Сигналы являются программными прерываниями, которые посылаются процессу, когда случается некоторое событие. Сигналы могут возникать синхронно с ошибкой в приложении, например SIGFPE (ошибка вычислений с плавающей запятой) и SIGSEGV (ошибка адресации), но большинство сигналов является асинхронными. Сигналы могут посылаться процессу, если система обнаруживает программное событие, например, когда пользователь дает команду прервать или остановить выполнение, или получен сигнал на завершение от другого процесса.

4. Очереди сообщений представляют собой связный список в адресном пространстве ядра. Сообщения могут посылаться в очередь по порядку и доставаться из очереди несколькими разными путями. Каждая очередь сообщений однозначно определена идентификатором IPC. Очереди сообщений как средство межпроцессной связи позволяют процессам взаимодействовать, обмениваясь данными. Данные передаются между процессами дискретными порциями, называемыми сообщениями. Процессы, использующие этот тип межпроцессной связи, могут выполнять две операции:послать или принять сообщение.

5. Семафоры являются одним из классических примитивов синхронизации. Семафор (semaphore) - это целая переменная, значение которой можно опрашивать и менять только при помощи неделимых (атомарных) операций. Двоичный семафор может принимать только значения 0 или 1. Считающий семафор может принимать целые неотрицательные значения. В приложениях как правило требуется использование более одного семафора, ОС должна представлять возможность создавать множества семафоров.

6. Сокеты обеспечивают двухстороннюю связь типа точка-точка *между двумя процессами. Они являются основными компонентами межсистемной и межпроцессной связи. Каждый сокет представляет собой конечную точку связи, с которой может быть совмещено некоторое имя. Он имеет определенный тип, и один процесс или несколько, связанных с ним процессов.*

Теперь другой источник, но менее подробно.

- *сигналы,* которые используются в качестве признака возникновения события;
- *конвейеры* (известные пользователям оболочек в виде оператора |) и FIFO-буферы, которые могут применяться для передачи данных между процессами;
- *сокеты,* которые могут использоваться для передачи данных от одного процесса к другому; данные при этом находятся на одном и том же базовом компьютере либо на различных хостах, связанных по сети;
- *файловая блокировка,* позволяющая процессу блокировать области файла с целью предотвращения их чтения или обновления содержимого файла другими процессами;
- *очереди сообщений,* которые используются для обмена сообщениями (пакетами данных) между процессами;
- *семафоры,* которые применяются для синхронизации действий процессов;
- *совместно используемая память,* позволяющая двум и более процессам совместно использовать часть памяти. Когда один процесс изменяет содержимое совместно используемой области памяти, изменения тут же могут быть видимы всем остальным процессам.

24. Потоки. Преимущества и недостатки использования потоков.

Поток — это по сути последовательность инструкций, которые выполняются параллельно с другими потоками. Каждая программа создает по меньшей мере один поток: основной, который запускает функцию main(). Программа, использующая только главный поток, является однопоточной; если добавить один или более потоков, она станет многопоточной.

Каждый процесс имеет ***основной,*** или ***первичный, поток.*** Под основным потоком процесса понимается программный поток управления или поток выполнения. Процесс может иметь несколько потоков выполнения и, соответственно, столько же потоков управления. Каждый поток, имея собственную последовательность инструкций, выполняется независимо от других, а все они — параллельно друг другу.

Преимущества:

- Так как множество потоков способно размещаться внутри одного EXE-модуля, это позволяет экономить ресурсы как внешней, так и внутренней памяти.
- Как правило, контекст потоков меньше, чем контекст процессов, а значит, время переключения между задачами-потоками меньше, чем между задачами-процессами.
- Для переключения контекста требуется меньше системных ресурсов.(При организации процесса для выполнения возложенной на него функции может оказаться вполне достаточно одного основного потока. Если же процесс имеет множество параллельных подзадач, то их асинхронное выполнение можно обеспечить с помощью нескольких потоков, на переключение контекста которых потребуются незначительные затраты системных ресурсов. При ограниченной доступности процессора или при наличии в системе только одного процессора параллельное выполнение процессов потребует существенных затрат системных ресурсов в связи с необходимостью обеспечить переключение контекста.)
- Достигается более высокая производительность приложения(Создание нескольких потоков повышает производительность приложения. При использовании одного потока запрос к устройствам ввода-вывода может остановить весь процесс. Если же в приложении организовано несколько потоков, то пока один

из них будет ожидать удовлетворения запроса ввода-вывода, другие потоки, которые не зависят от заблокированного, смогут продолжать выполнение. Тот факт, что не все потоки ожидают завершения операции ввода-вывода, означает, что приложение в целом не заблокировано ожиданием, а продолжает работать.).

- Для обеспечения взаимодействия между задачами не требуется никакого специального механизма.
- Программа имеет более простую структуру.(Потоки можно использовать, чтобы упростить структуру приложения. Каждому потоку назначается подзадача или подпрограмма, за выполнение которой он отвечает. Поток должен независимо управлять выполнением своей подзадачи. Каждому потоку можно присвоить приоритет, отражающий важность выполняемой им задачи Для приложения. Такой подход позволяет упростить поддержку программного кода.)

Недостатки:

Простота доступности потоков к памяти процесса имеет свои недостатки.

• Потоки могут легко разрушить адресное пространство процесса(Потоки могут легко разрушить информацию процесса во время «гонки» данных, если сразу несколько потоков получат доступ для записи одних и тех же данных. При использовании процессов это невозможно. Каждый процесс имеет собственные данные, и другие процессы не в состоянии получить к ним доступ, если не сделать это специально. Защита информации обусловлена наличием у процессов отдельных адресных пространств. Тот факт, что потоки совместно используют одно и то же адресное пространство, делает данные незащищенными от искажения.).

• Потоки необходимо синхронизировать при параллельном доступе (для чтения или записи) к памяти.

• Один поток может ликвидировать целый процесс или программу(Поскольку потоки не имеют собственного адресного пространства, они не изолированы. Если поток стал причиной

фатального нарушения доступа, это может привести к завершению всего процесса. Процессы изолированы друг от друга. Если процесс разрушит свое адресное пространство, проблемы ограничатся этим процессом. Процесс может допустить нарушение доступа, которое приведет к его завершению, но все остальные процессы будут продолжать выполнение. Это нарушение не окажется фатальным для всего приложения. Ошибки, связанные с некорректностью данных, могут не выйти за рамки одного процесса. Но ошибки, вызванные некорректным поведением потока, как правило, гораздо серьезнее ошибок, допущенных процессом.).

• Потоки существуют только в рамках единого процесса и, следовательно, не являются многократно используемыми.(Потоки зависят от процесса, в котором они существуют, и их невозможно от него отделить. Процессы отличаются большей степенью независимости, чем потоки. Приложение можно так разделить на задачи, порученные процессам, что эти процессы можно оформить в виде модулей, которые возможно использовать в других приложениях. Потоки не могут существовать вне процессов, в которых они были созданы и, следовательно, они не являются повторно используемыми.)

25. Программирование потоков в ОС Unix/Linux.

В Linux каждый поток является процессом, и для того, чтобы создать новый поток, нужно создать новый процесс. В чем же, в таком случае, заключается преимущество многопоточности Linux перед многопроцессностью? В многопоточных приложениях Linux для создания дополнительных потоков используются процессы особого типа. Эти процессы представляют собой обычные дочерние процессы главного процесса, но они разделяют с главным процессом адресное пространство, файловые дескрипторы и обработчики сигналов. Для обозначения процессов этого типа, применяется специальный термин – легкие процессы (lightweight processes). Прилагательное «легкий» в

названии процессов – потоков вполне оправдано. Поскольку этим процессам не нужно создавать собственную копию адресного пространства (и других ресурсов) своего процесса – родителя, создание нового легкого процесса требует значительно меньших затрат, чем создание полновесного дочернего процесса. Поскольку потоки Linux на самом деле представляют собой процессы, в мире Linux нельзя говорить, что один процесс содержит несколько потоков.

В Linux у каждого процесса есть идентификатор. Есть он, естественно, и у процессов-потоков. С другой стороны, спецификация POSIX 1003.1c требует, чтобы все потоки многопоточного приложения имели один идентификатор. Вызвано это требование тем, что для многих функций системы многопоточное приложение должно представляться как один процесс с одним идентификатором. Проблема единого идентификатора решается в Linux весьма элегантно. Процессы многопоточного приложения группируются в группы потоков (thread groups). Группе присваивается идентификатор, соответствующий идентификатору первого процесса многопоточного приложения. Именно этот идентификатор группы потоков используется при «общении» с многопоточным приложением.

26. Физическая организация памяти компьютера.

Запоминающие устройства компьютера разделяют, как минимум, на два уровня: основную (главную, оперативную, физическую) и вторичную (внешнюю) память.

Основная память представляет собой упорядоченный массив однобайтовых ячеек, каждая из которых имеет свой уникальный адрес (номер). Процессор извлекает команду из основной памяти, декодирует и выполняет ее. Для выполнения команды могут потребоваться обращения еще к нескольким ячейкам основной памяти. Обычно основная память изготавливается с применением полупроводниковых технологий и теряет свое содержимое при отключении питания.

Вторичную память (это главным образом диски) также можно рассматривать как одномерное линейное адресное пространство, состоящее из последовательности байтов. В отличие от оперативной памяти, она является

энергонезависимой, имеет существенно большую емкость и используется в качестве расширения основной памяти.

Эту схему можно дополнить еще несколькими промежуточными уровнями, как показано на рис. 8.1. Разновидности памяти могут быть объединены в иерархию по убыванию времени доступа, возрастанию цены и увеличению емкости.

Иерархия памяти

Рис. 8.1. Иерархия памяти

Многоуровневую схему используют следующим образом. Информация, которая находится в памяти верхнего уровня, обычно хранится также на уровнях с большими номерами. Если процессор не обнаруживает нужную информацию на i-м уровне, он начинает искать ее на следующих уровнях. Когда нужная информация найдена, она переносится в более быстрые уровни.

Локальность
Оказывается, при таком способе организации по мере снижения скорости доступа к уровню памяти снижается также и частота обращений к нему.

Ключевую роль здесь играет свойство реальных программ, в течение ограниченного отрезка времени способных работать с небольшим набором адресов памяти. Это эмпирически наблюдаемое свойство известно как принцип локальности или локализации обращений.

Свойство локальности (соседние в пространстве и времени объекты характеризуются похожими свойствами) присуще не только функционированию ОС, но и природе вообще. В случае ОС свойство локальности объяснимо, если учесть, как пишутся программы и как хранятся данные, то есть обычно в течение какого-то отрезка времени ограниченный фрагмент кода работает с ограниченным набором данных. Эту часть кода и данных удается разместить в памяти с быстрым доступом. В результате реальное время доступа к памяти определяется временем доступа к верхним уровням, что и обусловливает эффективность использования иерархической схемы. Надо сказать, что описываемая организация вычислительной системы во многом имитирует деятельность человеческого мозга при переработке информации. Действительно, решая конкретную проблему, человек работает с небольшим объемом информации, храня не относящиеся к делу сведения в своей памяти или во внешней памяти (например, в книгах).

Кэш процессора обычно является частью аппаратуры, поэтому менеджер памяти ОС занимается распределением информации главным образом в основной и внешней памяти компьютера. В некоторых схемах потоки между оперативной и внешней памятью регулируются программистом (см. например, далее оверлейные структуры), однако это связано с затратами времени программиста, так что подобную деятельность стараются возложить на ОС.

Адреса в основной памяти, характеризующие реальное расположение данных в физической памяти, называются физическими адресами. Набор физических адресов, с которым работает программа, называют физическим адресным пространством.

Логическая память

Аппаратная организация памяти в виде линейного набора ячеек не соответствует представлениям программиста о том, как организовано хранение программ и данных. Большинство программ представляет собой набор модулей, созданных независимо друг от друга. Иногда все модули, входящие в состав процесса, располагаются в памяти один за другим, образуя линейное пространство адресов. Однако чаще модули помещаются в разные области памяти и используются по-разному.

Схема управления памятью, поддерживающая этот взгляд пользователя на то, как хранятся программы и данные, называется сегментацией. Сегмент – область памяти определенного назначения, внутри которой поддерживается линейная адресация. Сегменты содержат процедуры, массивы, стек или скалярные величины, но обычно не содержат информацию смешанного типа.

По-видимому, вначале сегменты памяти появились в связи с необходимостью обобществления процессами фрагментов программного кода (текстовый редактор, тригонометрические библиотеки и т. д.), без чего каждый процесс должен был хранить в своем адресном пространстве дублирующую информацию. Эти отдельные участки памяти, хранящие информацию, которую система отображает в память нескольких процессов, получили название сегментов. Память, таким образом, перестала быть линейной и превратилась в двумерную. Адрес состоит из двух компонентов: номер сегмента, смещение внутри сегмента. Далее оказалось удобным размещать в разных сегментах различные компоненты процесса (код программы, данные, стек и т. д.). Попутно выяснилось, что можно контролировать характер работы с конкретным сегментом, приписав ему атрибуты, например права доступа или типы операций, которые разрешается производить с данными, хранящимися в сегменте.

Большинство современных ОС поддерживают сегментную организацию памяти. В некоторых архитектурах (Intel, например) сегментация поддерживается оборудованием.

Адреса, к которым обращается процесс, таким образом, отличаются от адресов, реально существующих в оперативной памяти. В каждом конкретном случае используемые программой адреса могут быть представлены различными способами. Например, адреса в исходных текстах обычно символические. Компилятор связывает эти символические адреса с перемещаемыми адресами (такими, как n байт от начала модуля). Подобный адрес, сгенерированный программой, обычно называют логическим (в системах с виртуальной памятью он часто называется виртуальным) адресом. Совокупность всех логических адресов называется логическим (виртуальным) адресным пространством.

Связывание адресов
Итак логические и физические адресные пространства ни по организации, ни по размеру не соответствуют друг другу. Максимальный размер логического адресного пространства обычно определяется разрядностью процессора (например, 232) и в современных системах значительно превышает размер физического адресного пространства. Следовательно, процессор и ОС должны быть способны отобразить ссылки в коде программы в реальные физические адреса, соответствующие текущему расположению программы в основной памяти. Такое отображение адресов называют трансляцией (привязкой) адреса или связыванием адресов

Связывание логического адреса, порожденного оператором программы, с физическим должно быть осуществлено до начала выполнения оператора или в момент его выполнения. Таким образом, привязка инструкций и данных к памяти в принципе может быть сделана на следующих шагах [Silberschatz, 2002].

Этап компиляции (Compile time). Когда на стадии компиляции известно точное место размещения процесса в памяти, тогда непосредственно генерируются физические адреса. При изменении стартового адреса программы необходимо перекомпилировать ее код. В качестве примера можно привести .com программы MS-DOS, которые связывают ее с физическими адресами на стадии компиляции.
Этап загрузки (Load time). Если информация о размещении программы на стадии компиляции отсутствует, компилятор генерирует перемещаемый код. В этом случае окончательное связывание откладывается до момента загрузки. Если стартовый адрес меняется, нужно всего лишь перезагрузить код с учетом измененной величины.
Этап выполнения (Execution time). Если процесс может быть перемещен во время выполнения из одной области памяти в другую, связывание откладывается до стадии выполнения. Здесь желательно наличие специализированного оборудования, например регистров перемещения. Их значение прибавляется к каждому адресу, сгенерированному процессом. Большинство современных ОС осуществляет трансляцию адресов на этапе выполнения, используя для этого специальный аппаратный механизм

27. Функции системы управления памятью.

Чтобы обеспечить эффективный контроль использования памяти, ОС должна выполнять следующие функции:

- отображение *адресного пространства* процесса на конкретные области физической памяти;
- распределение памяти между конкурирующими процессами;
- контроль доступа к *адресным пространствам* процессов;
- выгрузка процессов (целиком или частично) во внешнюю память, когда в *оперативной памяти* недостаточно места;
- учет свободной и занятой памяти.

Очень похожее с другого источника

Функциями ОС по управлению памятью являются:

· отслеживание свободной и занятой памяти,

· выделение памяти процессам и освобождение памяти при завершении процессов,

· вытеснение процессов из оперативной памяти на диск, когда размеры основной памяти не достаточны для размещения в ней всех процессов, и возвращение их в оперативную память, когда в ней освобождается место

· настройка адресов программы на конкретную область физической памяти.

28. Простейшие схемы управления памятью

ОС начали свое существование с применения очень простых методов управления памятью. Применявшаяся техника распространялась от статического распределения памяти (каждый процесс пользователя должен полностью поместиться в основной памяти, и система принимает к обслуживанию дополнительные пользовательские процессы до тех пор, пока все они одновременно помещаются в основной памяти), с промежуточным решением в виде "простого свопинга" (система по-прежнему располагает каждый процесс в основной памяти целиком, но иногда на основании некоторого критерия целиком

сбрасывает образ некоторого процесса из основной памяти во внешнюю память и заменяет его в основной памяти образом некоторого другого процесса).

Схема с фиксированными разделами.

Самым простым способом управления оперативной памятью является ее предварительное (обычно на этапе генерации или в момент загрузки системы) разбиение на несколько разделов фиксированной величины. По мере прибытия процесс помещается в тот или иной раздел.

Как правило, происходит условное разбиение физического адресного пространства. Связывание логических адресов процесса и физических происходит на этапе его загрузки в конкретный раздел.

Каждый раздел может иметь свою очередь или может существовать глобальная очередь для всех разделов.

Подсистема управления памятью сравнивает размер программы, поступившей на выполнение, выбирает подходящий раздел, осуществляет загрузку программы и настройку адресов.

В какой раздел помещать программу? Распространены три стратегии:

- Стратегия первого подходящего (First fit). Задание помещается в первый подходящий по размеру раздел.
- Стратегия наиболее подходящего (Best fit). Задание помещается в тот раздел, где ему наиболее тесно.
- Стратегия наименее подходящего (Worst fit). При помещении в самый большой раздел в нем остается достаточно места для возможного размещения еще одного процесса.

Моделирование показало, что с точки зрения утилизации памяти и уменьшения времени первые два способа лучше. С точки зрения утилизации первые два примерно одинаковы, но первый способ быстрее.

Связывание (настройка) адресов для данной схемы возможны как на этапе компиляции, так и на этапе загрузки.

Очевидный недостаток этой схемы число одновременно выполняемых процессов ограничено числом разделов.

Недостатком является то,что процесс не полностью занимает выделенный ему раздел или используются разделы,которые слишком малы для выполняемых пользовательских программ.

Оверлейная структура

Так как размер логического адресного пространства процесса может быть больше чем размер выделенного ему раздела (или больше чем размер самого большого раздела), иногда используется техника, называемая оверлей (overlay) или организация структуры с перекрытием. Основная идея - держать в памяти только те инструкции программы, которые нужны в данный момент времени.

Потребность в таком способе загрузки появляется, если логическое адресное пространство системы мало, например 1 мегабайт (MS-DOS) или даже всего 64 килобайта (PDP-11), а программа относительно велика. На современных 32-разрядных системах, где виртуальное адресное пространство измеряется гигабайтами, проблемы с нехваткой памяти решаются другими способами.

Коды ветвей оверлейной структуры программы находятся на диске как абсолютные образы памяти и считываются драйвером оверлеев при необходимости. Для конструирования оверлеев необходимы специальные алгоритмы перемещения и связывания. Совокупность файлов исполняемой программы дополняется файлом (обычно с расширением .odl), описывающим дерево вызовов внутри программы. Синтаксис подобного файла может распознаваться загрузчиком. Привязка к памяти происходит в момент очередной загрузки одной из ветвей программы.

Оверлеи не требуют специальной поддержки со стороны ОС. Они могут быть полностью реализованы на пользовательском уровне с

простой файловой структурой. ОС лишь делает несколько больше операций ввода-вывода. Типовое решение порождение линкером специальных команды, которые включают загрузчик каждый раз: когда требуется обращение к одной из перекрывающихся ветвей программы.

29. Виртуальная память.

Виртуальная память — метод управления памятью, которая реализуется с использованием аппаратного и программного обеспечения компьютера. Основная память представляется в виде непрерывного адресного пространства или набора смежных непрерывных сегментов. Операционная система осуществляет управление виртуальными адресными пространствами и соотнесением оперативной памяти с виртуальной. Программное обеспечение в операционной системе может расширить эти возможности, чтобы обеспечить виртуальное адресное пространство, которое может превысить объем оперативной памяти и таким образом иметь больше памяти, чем есть в компьютере. Виртуальная память позволяет модифицировать ресурсы памяти, сделать объём оперативной памяти намного больше, для того чтобы пользователь, поместив туда как можно больше программ, реально сэкономил время и повысил эффективность своего труда. “Открытие” виртуальной памяти внесло огромную контрибуцию в развитие современных технологий, облегчило работу как профессионального программиста, так и обычного пользователя, обеспечивая процесс более эффективного решения задач на ЭВМ.

К основным преимуществам виртуальной памяти относят:

1. избавление программиста от необходимости управлять общим пространством памяти,
2. повышение безопасность использования программ за счет выделения памяти,
3. возможность иметь в распоряжении больше памяти, чем это может быть физически доступно на компьютере.

Виртуальная память делает программирование приложений проще:

– скрывая фрагментацию физической памяти;

– устраняя необходимость в программе для обработки наложений в явном виде;

– когда каждый процесс запускается в своем собственном выделенном адресном пространстве, нет необходимости переместить код программы или получить доступ к памяти с относительной адресацией.

Виртуализация памяти может рассматриваться как обобщение понятия виртуальной памяти.

Почти все реализации виртуальной памяти делят виртуальное адресное пространство на страницы, блоки смежных адресов виртуальной памяти.

При работе машины с виртуальной памятью, используются методы страничной и сегментной организации памяти.

30. Сегментная, страничная и сегментно-страничная организация памяти.

Страничная память

В современных схемах управления памятью не принято размещать процесс в оперативной памяти одним непрерывным блоком.

В самом простом и наиболее распространенном случае страничной организации памяти (или paging) как логическое адресное пространство, так и физическое представляются состоящими из наборов блоков или страниц одинакового размера. При этом образуются логические страницы (page), а соответствующие единицы в физической памяти называют физическими страницами или страничными кадрами (page frames). Страницы (и страничные кадры) имеют фиксированную длину, обычно являющуюся степенью числа 2, и не могут перекрываться. Каждый кадр содержит одну страницу данных. При такой организации внешняя фрагментация отсутствует, а потери из-за внутренней фрагментации, поскольку процесс занимает целое число страниц, ограничены частью последней страницы процесса.

Логический адрес в страничной системе – упорядоченная пара (p,d), где p – номер страницы в виртуальной памяти, а d – смещение в рамках страницы p, на которой размещается адресуемый элемент. Заметим, что разбиение адресного пространства на страницы осуществляется вычислительной системой незаметно для программиста. Поэтому адрес является двумерным лишь с точки зрения

операционной системы, а с точки зрения программиста адресное пространство процесса остается линейным.

Описываемая схема позволяет загрузить процесс, даже если нет непрерывной области кадров, достаточной для размещения процесса целиком. Но одного базового регистра для осуществления трансляции адреса в данной схеме недостаточно. Система отображения логических адресов в физические сводится к системе отображения логических страниц в физические и представляет собой таблицу страниц, которая хранится в оперативной памяти. Иногда говорят, что таблица страниц – это кусочно-линейная функция отображения, заданная в табличном виде.

Сегментная и сегментно-страничная организация памяти

Существуют две другие схемы организации управления памятью: сегментная и сегментно-страничная. Сегменты, в отличие от страниц, могут иметь переменный размер. Идея сегментации изложена во введении. При сегментной организации виртуальный адрес является двумерным как для программиста, так и для операционной системы, и состоит из двух полей – номера сегмента и смещения внутри сегмента. Подчеркнем, что в отличие от страничной организации, где линейный адрес преобразован в двумерный операционной системой для удобства отображения, здесь двумерность адреса является следствием представления пользователя о процессе не в виде линейного массива байтов, а как набор сегментов переменного размера (данные, код, стек...).

Программисты, пишущие на языках низкого уровня, должны иметь представление о сегментной организации, явным образом меняя значения сегментных регистров (это хорошо видно по текстам программ, написанных на Ассемблере). Логическое адресное пространство – набор сегментов. Каждый сегмент имеет имя, размер и другие параметры (уровень привилегий, разрешенные виды обращений, флаги присутствия). В отличие от страничной схемы, где пользователь задает только один адрес, который разбивается на номер страницы и смещение прозрачным для программиста образом, в сегментной схеме пользователь специфицирует каждый адрес двумя величинами: именем сегмента и смещением.

Каждый сегмент – линейная последовательность адресов, начинающаяся с 0. Максимальный размер сегмента определяется разрядностью процессора (при 32-разрядной адресации это 232 байт или 4 Гбайт). Размер сегмента может меняться динамически (например, сегмент стека). В элементе таблицы сегментов помимо физического адреса начала сегмента обычно содержится и длина сегмента. Если размер смещения в виртуальном адресе выходит за пределы размера сегмента, возникает исключительная ситуация.

Логический адрес – упорядоченная пара v=(s,d), номер сегмента и смещение внутри сегмента.

В системах, где сегменты поддерживаются аппаратно, эти параметры обычно хранятся в таблице дескрипторов сегментов, а программа обращается к этим дескрипторам по номерам-селекторам. При этом в контекст каждого процесса входит набор сегментных регистров, содержащих селекторы текущих сегментов кода, стека, данных и т. д. и определяющих, какие сегменты будут использоваться при разных видах обращений к памяти. Это позволяет процессору уже на аппаратном уровне определять допустимость обращений к памяти, упрощая реализацию защиты информации от повреждения и несанкционированного доступа.
Сегментно-страничная и сегментная организация памяти позволяет легко организовать совместное использование одних и тех же данных и программного кода разными задачами. Для этого различные логические блоки памяти разных процессов отображают в один и тот же блок физической памяти, где размещается разделяемый фрагмент кода или данных.

31. Исключительные ситуации при работе с памятью.

Отображение виртуального адреса в физический осуществляется при помощи таблицы страниц. Для каждой виртуальной страницы запись в таблице страниц содержит номер соответствующего страничного кадра в оперативной памяти, а также атрибуты страницы для контроля обращений к памяти.

Страничное нарушение может происходить в самых разных случаях: при отсутствии страницы в оперативной памяти, при попытке записи в страницу с атрибутом "только чтение" или при попытке чтения или записи страницы с атрибутом "только выполнение". В любом из этих случаев вызывается обработчик страничного нарушения, являющийся частью ОС. Ему обычно передается причина возникновения исключительной ситуации и виртуальный адрес, обращение к которому вызвало нарушение.

Когда программа обращается к виртуальной странице, отсутствующей в основной памяти, ОС должна выделить страницу основной памяти, переместить в нее копию виртуальной страницы из внешней памяти и модифицировать соответствующий элемент таблицы страниц.

Время эффективного доступа к отсутствующей в оперативной памяти странице складывается из:

· обслуживания исключительной ситуации;

· чтения (подкачки) страницы из вторичной памяти;

· возобновления выполнения процесса, вызвавшего данный page fault.

32. Алгоритмы замещения страниц.

Существует большое количество разнообразных алгоритмов замещения страниц. Все они делятся на локальные и глобальные. Локальные алгоритмы, в отличие от глобальных, распределяют фиксированное или динамически настраиваемое число страниц для каждого процесса. Когда процесс израсходует все предназначенные ему страницы, система будет удалять из физической памяти одну из его страниц, а не из страниц других процессов. Глобальный же алгоритм замещения в случае возникновения исключительной ситуации удовлетворится освобождением любой физической страницы, независимо от того, какому процессу она принадлежала.

Глобальные алгоритмы имеют несколько недостатков. Во-первых, они делают одни процессы чувствительными к поведению других процессов. Во-вторых, некорректно работающее приложение может подорвать работу всей системы

Идеальный алгоритм заключается в том, что бы выгружать ту страницу, которая будет запрошена позже всех.

Но этот алгоритм не осуществим, т.к. нельзя знать какую страницу, когда запросят. Можно лишь набрать статистику использования.

Алгоритм NRU (Not Recently Used - не использовавшаяся в последнее время страница)

Используются биты обращения (R-Referenced) и изменения (M-Modified) в таблице страниц.

При обращении бит R выставляется в 1, через некоторое время ОС не переведет его в 0.

M переводится в 0, только после записи на диск.

Благодаря этим битам можно получить 4-ре класса страниц:

1. не было обращений и изменений (R=0, M=0)
2. не было обращений, было изменение (R=0, M=1)
3. было обращение, не было изменений (R=1, M=0)
4. было обращений и изменений (R=1, M=1)

Алгоритм FIFO (первая прибыла - первая выгружена)

Недостаток заключается в том, что наиболее часто запрашиваемая страница может быть выгружена.

Алгоритм "вторая попытка"

Подобен FIFO, но если R=1, то страница переводится в конец очереди, если R=0, то страница выгружается.

В таком алгоритме часто используемая страница никогда не покинет память.

Но в этом алгоритме приходится часто перемещать страницы по списку.

33. Модель рабочего множества.

34. Основы аппаратного и программного обеспечения ввода-вывода.

Ввод/вывод считается одной из самых сложных областей проектирования операционных систем, в которой сложно применить общий подход из-за изобилия частных методов.

Сложность возникает из-за огромного числа устройств ввода/вывода разнообразной природы, которые должна поддерживать ОС. При этом перед создателями ОС встает очень непростая задача — не только обеспечить эффективное управление устройствами ввода/вывода, но и создать удобный и эффективный виртуальный интерфейс устройств ввода/вывода, позволяющий прикладным программистам просто считывать или сохранять данные, не обращая внимание на специфику устройств и проблемы распределения устройств между выполняющимися задачами. Система ввода/вывода, способная объединить в одной модели широкий набор устройств, должна быть универсальной. Она должна учитывать потребности существующих устройств, от простой мыши до клавиатур, принтеров, графических дисплеев, дисковых накопителей, компакт-дисков и даже сетей. С другой стороны, необходимо обеспечить доступ к устройствам ввода/вывода для множества параллельно выполняющихся задач, причем так, чтобы они как можно меньше мешали друг другу.

Поэтому самым главным является следующий принцип: любые операции по управлению вводом/выводом объявляются привилегированными и могут выполняться только кодом самой ОС. Для обеспечения этого принципа в большинстве процессоров даже вводятся режимы пользователя и супервизора. Как правило, в режиме супервизора выполнение команд ввода/вывода разрешено, а в пользовательском режиме — запрещено. Использование команд ввода/вывода в пользовательском режиме вызывает исключение и управление через механизм прерываний передается коду ОС. Хотя возможны и более сложные системы, в которых в ряде случаев пользовательским программам разрешено непосредственное выполнение команд ввода/вывода.

Еще раз подчеркнем, что, прежде всего, мы говорим о мультипрограммных ОС, для которых существует проблема разделения ресурсов.

Одним из основных видов ресурсов являются устройства ввода/вывода и соответствующее программное обеспечение, с помощью которого осуществляется управление обменом данными между внешними устройствами и оперативной памятью. Помимо разделяемых устройств ввода/вывода (эти устройства допускают разделение посредством механизма доступа) существуют

неразделяемые устройства. Примерами разделяемого устройства могут служить накопитель на магнитных дисках, устройство для чтения компакт-дисков. Это устройства с прямым доступом. Примеры неразделяемых устройств — принтер, накопитель на магнитных лентах. Это устройства с последовательным доступом. Операционные системы должны управлять и теми и другими устройствами, предоставляя возможность параллельно выполняющимся задачам использовать различные устройства ввода/вывода.

Можно назвать три основные причины, по которым нельзя разрешать каждой отдельной пользовательской программе обращаться к внешним устройствам непосредственно:

1. Необходимость разрешать возможные конфликты доступа к устройствам ввода/вывода. Например, две параллельно выполняющиеся программы пытаются вывести на печать результаты своей работы. Если не предусмотреть внешнее управление устройством печати, то в результате мы можем получить абсолютно нечитаемый текст, так как каждая программа будет время от времени выводить свои данные, которые будут перемежаться данными другой программы. Другой пример: ситуация, когда одной программе необходимо прочитать данные с некоторого сектора магнитного диска, а другой — записать результаты в другой сектор того же накопителя. Если операции ввода/вывода не будут отслеживаться каким-то третьим (внешним) процессом-арбитром, то после позиционирования магнитной головки для первого запроса может тут же появиться команда позиционирования головки для второй задачи, и обе операции ввода/вывода не смогут быть выполнены корректно.

2. Желание увеличить эффективность использования этих ресурсов. Например, у накопителя на магнитных дисках время подвода головки чтения/записи к необходимой дорожке и обращение к определенному сектору может значительно (до тысячи раз) превышать время пересылки данных. В результате, если задачи по очереди обращаются к цилиндрам, далеко отстоящим друг от друга, то полезная работа, выполняемая накопителем, может быть существенно снижена.

3. Ошибки в программах ввода/вывода могут привести к краху всех вычислительных процессов, ибо часть операций ввода/вывода осуществляется

для самой операционной системы. В ряде ОС системный ввод/вывод имеет существенно более высокие привилегии, чем ввод/вывод задач пользователя. Поэтому системный код, управляющий операциями ввода/вывода, очень тщательно отлаживается и оптимизируется для повышения надежности вычислений и эффективности использования оборудования.

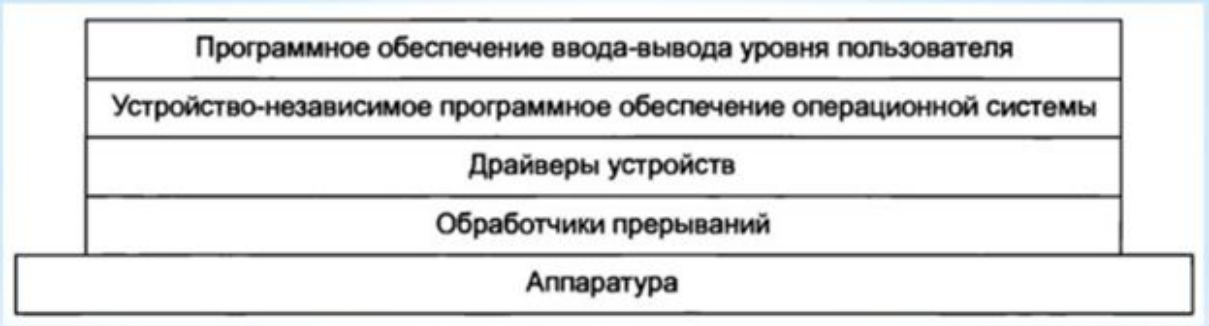
Итак, управление вводом/выводом осуществляется операционной системой, компонентом, который чаще всего называют супервизором ввода/вывода.

В перечень основных задач, возлагаемых на супервизор, входят следующие:

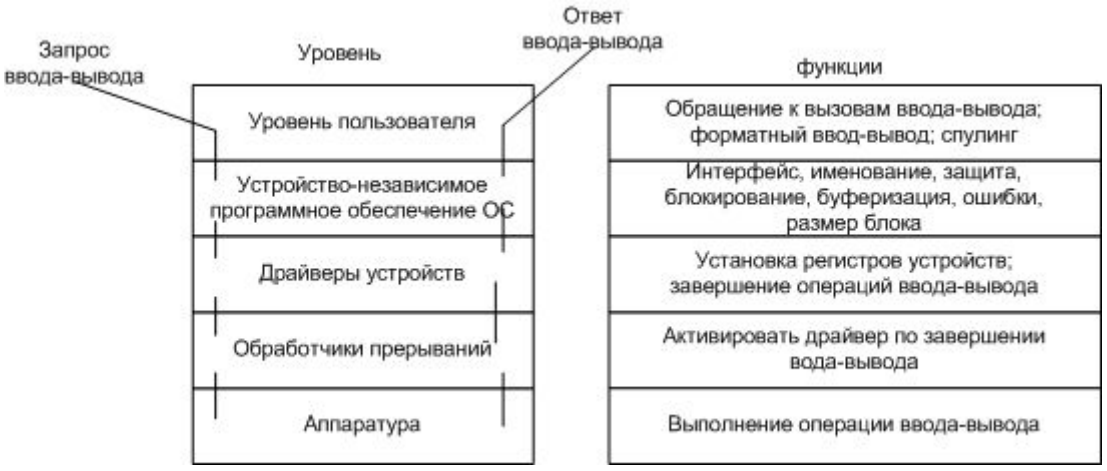
- супервизор ввода/вывода получает запросы на ввод/вывод от прикладных задач и от программных модулей самой операционной системы. Эти запросы проверяются на корректность, и если запрос выполнен по спецификациям и не содержит ошибок, он обрабатывается дальше, в противном случае пользователю (задаче) выдается соответствующее диагностическое сообщение о недействительности (некорректности) запроса;
- супервизор ввода/вывода вызывает соответствующие распределители каналов и контроллеров, планирует ввод/вывод (определяет очередность предоставления устройств ввода/вывода задачам, затребовавшим их). Запрос на ввод/ вывод либо тут же выполняется, либо ставится в очередь на выполнение;
- супервизор ввода/вывода инициирует операции ввода/вывода (передает управление соответствующим драйверам) и в случае управления вводом/выводом с использованием прерываний предоставляет процессор диспетчеру задач с тем, чтобы передать его первой задаче, стоящей в очереди на выполнение;
- при получении сигналов прерываний от устройств ввода/вывода супервизор идентифицирует их и передает управление соответствующей программе обработки прерывания (как правило, на секцию продолжения драйвера);
- супервизор ввода/вывода осуществляет передачу сообщений об ошибках, если таковые происходят в процессе управления операциями ввода/вывода;

- супервизор ввода/вывода посылает сообщения о завершении операции ввода/вывода запросившему эту операцию процессу и снимает его с состояния ожидания ввода/вывода, если процесс ожидал завершения операции.

35. Уровни программного обеспечения ввода-вывода.



У каждого уровня есть четко определенная рабочая функция и четко определенный интерфейс с примыкающими к нему уровнями. Функции и интерфейсы варьируются от системы к системе, поэтому последующее рассмотрение уровней, начинающееся с самого низшего из них, не имеет отношения к какой-либо конкретной машине.



Информации очень много, её сложно будет заполнить, я оставлю ссылку https://studfile.net/preview/4034529/page:5/

Ключевыми принципами организации ПО ввода-вывода являются:

· *Независимость от устройств*. Вид программы не должен зависеть от того, читает ли она данные с гибкого диска или с жесткого диска. Каждому устройству присваивается свое логическое имя (в MSDOS – LPT, PRN, COM1, COM2 и т. д.). Для именования устройств должны быть приняты единые правила.

· О*бработка ошибок*. Ошибки следует обрабатывать как можно ближе к аппаратуре. Если контроллер обнаруживает ошибку чтения, то он должен попытаться ее скорректировать. Если ему это не удается, то исправлением ошибок должен заняться драйвер устройства. Многие ошибки могут исчезать при повторных попытках выполнения операций ввода-вывода, например, ошибки, вызванные наличием пылинок на головках чтения или на диске. И только если нижний уровень не может справиться с ошибкой, он сообщает об ошибке верхнему уровню.

· *Использование блокирующих (синхронных) и неблокирующих (асинхронных) передач*. Большинство операций физического ввода-вывода выполняется как неблокирующие. При этом процессор, выдав команду на передачу данных, переходит на другую работу, пока не наступает прерывание. Это обеспечивает более полное использование процессора. При блокирующей передаче после поступления команды READ программа приостанавливается до тех пор, пока данные не попадут в буфер программы. Пользовательские программы намного легче писать, если операции ввода-вывода блокирующие. Поэтому ОС выполняет операции ввода-вывода как неблокирующие, но представляет их для пользовательских программ как блокирующие.

· Устранение проблем при работе с выделенными устройствами. Устройства ввода-вывода могут быть *разделяемыми* и *выделенными*. Диски – это разделяемые устройства, так как одновременный доступ нескольких пользователей к диску не представляет собой проблему. Принтеры – это выделенные

устройства, потому что нельзя смешивать строчки, печатаемые различными пользователями.

36. Пользовательский интерфейс: клавиатура, мышь, монитор.

37. Командный и графический интерфейс ОС Unix/Linux.

38. Файловая система.

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

В широком смысле понятие "файловая система" включает:

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

39. Именование файлов.

Полным, или абсолютным, называется имя файла, содержащее все каталоги до корня файловой системы. Относительные имена файлов не содержат полного пути и обычно привязываются к текущему каталогу.
Windows
Полное имя файла в Windows-системах состоит из буквы диска, после которого ставится двоеточие и обратная наклонная черта (обратный слеш), затем через обратные слеши перечисляются подкаталоги, в конце пишется имя файла. Пример:

C:\Windows\System32\calc.exe
Полное имя файла (включая расширение) в Windows может содержать до 260 символов, данное значение определено константой MAX_PATH в Windows API; например, полное имя файла максимально допустимой длины на диске C будет таким: «C:\<256 символов>NULL» (например, «C:\<254 символа>\<1 символ>NULL»). Однако, юникодовые версии некоторых функций позволяют использовать полные имена файлов длиной до 32767 символов, такие имена начинаются с префикса «\\?\». Пример:
\\?\C:\Windows\System32\calc.exe
При использовании префикса «\\?\» необходимо указывать абсолютный путь к файлу, относительные пути не допускаются. При использовании относительных путей максимальная длина полного имени файла определена константой MAX_PATH (260 символов).
UNIX[править | править код]
В UNIX и UNIX-подобных системах полный путь состоит из слеша (/), обозначающего корневой каталог, после которого через слеши перечисляются подкаталоги, в конце пишется имя файла. Пример:
/usr/local/bin/gcc
Пути, начинающиеся не с косой черты, считаются относительными и отсчитываются относительно рабочего каталога. Пример:
../mc при нахождении в каталоге /etc/apache2 эквивалентно /etc/mc.
Особое значение у путей, которые начинаются с тильды (~). Тильда обозначает домашний каталог текущего или указанного пользователя. Пример:
~/Desktop для пользователя user во многих системах и случаях эквивалентно /home/user/Desktop[1]
~admin/passwords для любого пользователя равносильно /home/admin/passwords[1]
~root/test для любого пользователя — то же самое, что и /root/test[1], потому что суперпользователь (root) имеет особый домашний каталог.
Многие операционные системы запрещают использование некоторых служебных символов.
Запрещённые символы Windows (в различных версиях):
- \ — разделитель подкаталогов
- / — разделитель ключей командного интерпретатора
- : — отделяет букву диска или имя альтернативного потока данных

• * — заменяющий символ (маска «любое количество любых символов»)
• ? — заменяющий символ (маска «один любой символ»)
• " — используется для указания путей, содержащих пробелы
• < — перенаправление ввода
• > — перенаправление вывода
• | — обозначает конвейер
• + — (в различных версиях) конкатенация

Частично запрещённые символы Windows:

• пробел — не допускается в конце имени файла;
• . — не допускается в конце имени файла кроме имён каталогов, состоящих из точек и доступа с префиксом «\\?\».

Символы, вызывающие проблемы в широко распространённых компонентах:

• % — в Windows используется для подстановки переменных окружения в интерпретаторе команд, вызывает проблемы при открытии файла через стандартный диалог открытия файла;
• ! — в Windows используется для подстановки переменных окружения в интерпретаторе команд, в bash используется для доступа к истории[1];
• @ — в интерпретаторах команд вызывает срабатывание функций, предназначенных для почты.

В именах файлов UNIX и некоторых UNIX-подобных ОС запрещен слеш (/) — разделитель подкаталогов — и символ конца С-строки (\0). Перечисленные выше символы (кроме слеша) использовать можно, но из соображений совместимости их лучше избегать.

На имя файла отводится 8 символов, а на его расширение — 3 символа. Имя от расширения отделяется точкой. Как имя, так и расширение могут включать только алфавитно-цифровые символы латинского алфавита и символ подчеркивания, имя не может начинаться с цифры.

Соглашение 8.3 не является стандартом, и потому в ряде случаев отклонения от правильной формы записи допускаются как операционной системой, так и ее приложениями. Сегодня имена файлов, записанные в соответствии с соглашением 8.3, считаются “короткими”.

Имя файла считается «длинным», если оно допустимо с точки зрения операционной системы Windows, но не соответствует соглашению 8.3. Имя считается длинным, если:

• оно имеет длину свыше восьми символов или длину расширения свыше трех символов;
• оно содержит специальные символы, например пробелы;
• оно содержит символы как верхнего, так и нижнего регистра. Для длинных имен файлов операционная система Windows XP также способна сформировать короткое имя, которое обеспечивает совместимость с другими операционными системами.

40. Атрибуты файла.

Атрибуты – *это информация, описывающая свойства файла, которые могут быть ему навязаны, чтобы заставить его выглядеть соответствующим образом и предупреждать о своих особенностях.Атрибут может находиться в двух состояниях: либо установленный, либо снятый. Атрибуты рассматриваются отдельно от других метаданных, таких как даты, расширения имени файла или права доступа. Каталоги и другие объекты файловой системы также могут иметь определённые атрибуты. Также существуют расширенные атрибуты файлов, хранящие данные другого типа.*

Примеры возможных атрибутов файла:
– владелец файла;
– тип файла (обычный файл, каталог и т.д.);
– создатель файла;
– пароль для доступа к файлу;
– информация о разрешенных операциях доступа к файлу;
– время создания, последнего доступа и последнего изменения;
– текущий размер файла;
– максимальный размер файла;
– признак «только для чтения»;
– признак «скрытый файл»;
– признак «системный файл»;
– признак «архивный файл»;
– признак «двоичный/символьный»;
– признак «временный» (удалить после завершения процесса);
– признак блокировки;

– длина записи в файле;
– указатель на ключевое поле в записи;
– длина ключа.

Набор атрибутов файла определяется спецификой файловой системы: в файловых системах разного типа для характеристики файлов могут использоваться разные наборы атрибутов.

Пользователь может получать доступ к атрибутам, используя средства, предоставленные для этих целей файловой системой. Обычно разрешается читать значения любых атрибутов, а изменять – только некоторые.

Значения атрибутов файлов могут непосредственно содержаться в каталогах, как это сделано в файловой системе MS-DOS.На рисунке представлена структура записи в каталоге, содержащая простое символьное имя и атрибуты файла. Здесь буквами обозначены следующие признаки файла:

– R – только для чтения, read only. Файл можно удалить или редактировать только после снятия атрибута или специального согласия на снятие атрибута;

– A – архивный, archive. Признак того, что файл после его создания или редактирования еще не вносился в резервный архив (утилита BACKUP или команда XCOPY). Данный атрибут не имеет никакого отношения к программам сжатия (упаковки) информации, он просто является обозначением: «при необходимости подлежит архивированию»;

– H – скрытый, hidden. При обычном просмотре имя файла не видно на экране, обычно присваивается некоторым файлам операционной системы, чтобы они не были случайно удалены;

– S – системный, system. Этот атрибут имеют некоторые файлы операционной системы.

41. Операции с файлами.

42. Защита файлов.

43. Иерархия каталогов.

Катало́г — каталог, директория, справочник, папка — объект в файловой системе, упрощающий организацию файлов. Типичная файловая система содержит большое количество файлов, и каталоги помогают упорядочить её путём их группировки. Каталог может быть реализован как специальный файл, где регистрируется информация о других файлах и каталогах на носителе информации

Корневой каталог - Каталог, прямо или косвенно включающий в себя все прочие каталоги и файлы файловой системы, называется корневым. В Unix-подобных ОС он обозначается символом / (дробь, слеш), в DOS и Windows исторически используется символ \ (обратный слеш), но с некоторого времени поддерживается и /.

Текущий каталог - текущим называется каталог, с которым работает ОС, если ей не указать другого каталога. Он обозначается точкой (.).

Подкаталог - Каталог, находящийся внутри текущего или любого другого последующего далее по иерархии каталога, называют подкаталогом.

Родительские каталог - одительским каталогом называется каталог, в котором находится текущий. Он обозначается двумя точками (..).

Может вот это лишнее, но бог его знает о чём этот вопрос вообще

Основные поддиректории корневого каталога

Перечисленные ниже директории являются общепринятыми и все современные Unix-подобные системы содержат в них только те данные, для которых предназначены эти каталоги.

home — данный каталог содержит личные каталоги для пользователей операционной системы.

bin — данный каталог содержит в себе исполняемые файлы (файлы готовые к запуску). Большинство встроенных команд Unix-подобных систем (например: rm; mv; ls; cd) находятся в директории bin.

lib — данный каталог является библиотекой, содержащей в себе программные коды для исполняемых файлов.

dev — данный каталог содержит в себе файлы устройств операционной системы.

proc — данный каталог предоставляет статистику Вашей системы.

etc — данный каталог является «сердцем» системных настроек операционной системы. В ней находятся: пароли пользователей, параметры загрузки, инструкции по работе в сети, а также многие другие.

tmp — данный каталог является местом хранения небольших по размеру временных файлов. Данная директория периодически очищается.

usr — данный каталог можно назвать «стволом дерева» иерархии директорий. В этой директории находятся жизненно важные данные операционной системы, но если в корневом каталоге находятся все первостепенные данные системы, то в этом каталоге находятся более пользовательские данные, которые всё же скрыты от простого пользователя.

boot — данный каталог содержит в себе загрузочные файлы ядра. Здесь содержатся данные о том, как должна запускаться Ваша операционная система.

root — данный каталог содержит в себе настройки и данные по супер-пользователю операционной системы.

mnt — данный каталог содержит в себе точку монтирования для временно подключаемых файловых систем к Unix-подобной системе.

sbin — данный каталог содержит в себе исполняемые файлы управления системой.

var — данный каталог содержит в себе переменные, где программы записывают информацию о своей проделанной работе. Например, регистрация в системе, слежение за действиями пользователя, кэш и другое. Все они создают и обслуживают программы системы Unix-подобной системы.

cdrom — данный каталог содержит в себе информацию и способы соединения с приводом CD-ROM;

44. Структура файловой системы.

файловая система - это совокупность алгоритмов и стандартов, задействуемых с целью организации эффективного доступа пользователя ПК к данным, размещенным на компьютере.

Файловая система обычно размещается на дисках или других устройствах внешней памяти, имеющих блочную структуру. Кроме блоков, сохраняющих каталоги и файлы, во внешней памяти поддерживается еще несколько служебных областей.

В ОС Windows наиболее распространенной на сегодняшний день является файловая система NTFS, заменившая устаревшую файловую систему FAT.

Кластер — упрощенно, минимальная ячейка на жестком диске для хранения информации, эдакая коробочка для хранения файлов. Кластер имеет вполне конкретные стандартизованные размеры равные 512 байт раньше и 4 096 байт в настоящее время. В одном кластере хранится только один файл, если он меньше размера кластера, то все равно занимает весь кластер. Когда файл не помещается целиком в одном кластере, то он записывается кусочками по разным кластерам, необязательно соседним.

Первоначально, вся информация в виде файлов записывалась в файловую систему Windows в одну кучу, однако с ростом количества информации и емкости дисков это стало очень неудобно. Попробуйте найти нужную вам вещь в коробке, среди десятков других. Выходом из этой ситуации стало создание древовидной структуры папок (директорий или каталогов) сильно облегчающих структурирование и поиск информации. Внутри каталога создаются подкаталоги, и файлы группируются по логическому принципу удобному пользователю.

Дальнейший рост емкости дисков привел к следующему очевидному шагу, разбить один физический носитель информации на несколько логических

разделов (дисков). Логически выделенная часть смежных блоков на диске называется раздел (partition). Такая структура файловой системы применяется в настоящее время в операционной системе Windows.
азделы бывают двух видов: первичный (основной) и дополнительный (расширенный). В первом секторе основного раздела располагается загрузочный сектор, обеспечивающий загрузку ОС с данного раздела жесткого диска. Всего на физическом диске может быть четыре раздела и только один из них расширенный. Дополнительный раздел представляет собой оболочку для любого количества других логических разделов. Это позволяет обойти ограничение, только четыре раздела на физическом диске.
(Описание каталогов в линухе https://losst.ru/ctruktura-fajlovoj-sistemy-linux)

45. Организация дискового пространства.

46. Управление свободным и занятым дисковым пространством.

Дисковое пространство, не выделенное ни одному файлу, также должно быть управляемым. В современных ОС используется несколько способов учета используемого места на диске. Рассмотрим наиболее распространенные.

Часто список свободных блоков диска реализован в виде битового вектора (bit map или bit vector). Каждый блок представлен одним битом, принимающим значение 0 или 1, в зависимости от того, занят он или свободен. Например, 001111001111000110000001 ... .

Главное преимущество этого подхода состоит в том, что он относительно прост и эффективен при нахождении первого свободного блока или n последовательных блоков на диске. Многие компьютеры имеют инструкции манипулирования битами, которые могут использоваться для этой цели. Например, компьютеры семейств Intel и Motorola имеют инструкции, при помощи которых можно легко локализовать первый единичный бит в слове.

Описываемый метод учета свободных блоков используется в Apple Macintosh.

Другой подход - связать в список все свободные блоки, размещая указатель на первый свободный блок в специально отведенном месте диска, попутно кэшируя в памяти эту информацию.

Подобная схема не всегда эффективна. Для трассирования списка нужно выполнить много обращений к диску. Однако, к счастью, нам необходим, как правило, только первый свободный блок.

Иногда прибегают к модификации подхода связного списка, организуя хранение адресов n свободных блоков в первом свободном блоке. Первые n-1 этих блоков действительно используются. Последний блок содержит адреса других n блоков и т. д.

Существуют и другие методы, например, свободное пространство можно рассматривать как файл и вести для него соответствующий индексный узел.

Размер логического блока играет важную роль. В некоторых системах (Unix) он может быть задан при форматировании диска. Небольшой размер блока будет приводить к тому, что каждый файл будет содержать много блоков. Чтение блока осуществляется с задержками на поиск и вращение, таким образом, файл из многих блоков будет читаться медленно. Большие блоки обеспечивают более высокую скорость обмена с диском, но из-за внутренней фрагментации (каждый файл занимает целое число блоков, и в среднем половина последнего блока пропадает) снижается процент полезного дискового пространства.

Для систем со страничной организацией памяти характерна сходная проблема с размером страницы.

Рассмотрение методов работы с дисковым пространством дает общее представление о совокупности служебных данных, необходимых для описания файловой системы. Структура служебных данных типовой файловой системы, например Unix, на одном из разделов диска, таким образом, может состоять из четырех основных частей .В начале раздела находится суперблок, содержащий общее описание файловой системы, например:тип файловой системы; размер

файловой системы в блоках; размер массива индексных узлов; размер логического блока.

Описанные структуры данных создаются на диске в результате его форматирования(например, утилитами format, makefs и др.). Их наличие позволяет обращаться к данным на диске как к файловой системе, а не как к обычной последовательности блоков.

Массив индексных узлов (ilist) содержит список индексов, соответствующих файлам данной файловой системы. Размер массива индексных узлов определяется администратором при установке системы. Максимальное число файлов, которые могут быть созданы в файловой системе, определяется числом доступных индексных узлов.

В блоках данных хранятся реальные данные файлов. Размер логического блока данных может задаваться при форматировании файловой системы. Заполнение диска содержательной информацией предполагает использование блоков хранения данных для файлов директорий и обычных файлов и имеет следствием модификацию массива индексных узлов и данных, описывающих пространство диска. Отдельно взятый блок данных может принадлежать одному и только одному файлу в файловой системе.

Директория или каталог - это файл, имеющий вид таблицы и хранящий список входящих в него файлов или каталогов. Основная задача файлов-директорий - поддержка иерархической древовидной структуры файловой системы. Запись в директории имеет определенный для данной ОС формат, зачастую неизвестный пользователю, поэтому блоки данных файла-директории заполняются не через операции записи, а при помощи специальных системных вызовов (например, создание файла).

Для доступа к файлу ОС использует путь (pathname), сообщенный пользователем. Запись в директории связывает имя файла или имя поддиректории с блоками данных на диске . В зависимости от способа выделения файлу блоков диска эта ссылка может быть номером первого блока

или номером индексного узла. В любом случае обеспечивается связь символьного имени файла с данными на диске.

Когда система открывает файл, она ищет его имя в директории. Затем из записи в директории или из структуры, на которую запись в директории указывает, извлекаются атрибуты и адреса блоков файла на диске. Эта информация помещается в системную таблицу в главной памяти. Все последующие ссылки на данный файл используют эту информацию. Атрибуты файла можно хранить непосредственно в записи директории

Список файлов в директории обычно не является упорядоченным по именам файлов. Поэтому правильный выбор алгоритма поиска имени файла в директории имеет большое влияние на эффективность и надежность файловых систем.

Существует несколько стратегий просмотра списка символьных имен. Простейшей из них является линейный поиск. Директория просматривается с самого начала, пока не встретится нужное имя файла. Хотя это наименее эффективный способ поиска, оказывается, что в большинстве случаев он работает с приемлемой производительностью. Метод прост, но требует временных затрат. Для создания нового файла вначале нужно проверить директорию на наличие такого же имени. Затем имя нового файла вставляется в конец директории. Для удаления файла нужно также выполнить поиск его имени в списке и пометить запись как неиспользуемую. Реальный недостаток данного метода - последовательный поиск файла. Информация о структуре директории используется часто, и неэффективный способ поиска будет заметен пользователями. Можно свести поиск к бинарному, если отсортировать список файлов. Однако это усложнит создание и удаление файлов, так как требуется перемещение большого объема информации.

Хеширование - другой способ, который может использоваться для размещения и последующего поиска имени файла в директории. В данном методе имена файлов также хранятся в каталоге в виде линейного списка, но дополнительно используется хеш-таблица. Хеш-таблица, точнее построенная на ее основе

хеш-функция, позволяет по имени файла получить указатель на имя файла в списке. Таким образом, можно существенно уменьшить время поиска.

47. Управление файловой системой и ее оптимизация.

Управление дисковым пространством

Обычно файлы хранятся на диске, поэтому управление дисковым пространством является основной заботой разработчиков файловой системы. Для хранения файла размером п байт возможно использование двух стратегий: выделение на диске п последовательных байтов или разбиение файла на несколько непрерывных блоков. Такая же дилемма между чистой сегментацией и страничной организацией присутствует и в системах управления памятью.

Как уже было показано, при хранении файла в виде непрерывной последовательности байтов возникает очевидная проблема: вполне вероятно, что по мере увеличения его размера потребуется его перемещение на новое место на диске

Размер блока

Как только принято решение хранить файлы в блоках фиксированного размера, возникает вопрос: каким должен быть размер блока?

Прежде всего, при размере блока 1 Кбайт только около 30-50 % всех файлов помещается в единичный блок, тогда как при размере блока 4 Кбайт количество файлов, помещающихся в блок, возрастает до 60-70 %. Судя по остальным данным, при размере блока 4 Кбайт 93 % дисковых блоков используется 10 % самых больших файлов. Это означает, что потеря некоторого пространства в конце каждого небольшого файла вряд ли имеет

какое-либо значение, поскольку диск заполняется небольшим количеством больших файлов (видеоматериалов), а то, что основной объем дискового пространства занят небольшими файлами, едва ли вообще имеет какое-то значение

Отслеживание свободных блоков

После выбора размера блока возникает следующий вопрос: как отслеживать свободные блоки?

Первый метод состоит в использовании связанного списка дисковых блоков, при этом в каждом блоке списка содержится столько номеров свободных дисковых блоков, сколько в него может поместиться. При блоках размером 1 Кбайт и 32-разрядном номере дискового блока каждый блок может хранить в списке свободных блоков номера 255 блоков. (Одно слово необходимо под указатель на следующий блок.) Рассмотрим диск емкостью 1 Тбайт, имеющий около 1 млрд дисковых блоков. Для хранения всех этих адресов по 255 на блок необходимо около 4 млн блоков. Как правило, для хранения списка свободных блоков используются сами свободные блоки, поэтому его хранение обходится практически бесплатно.

Другая технология управления свободным дисковым пространством использует битовый массив. Для диска, имеющего n блоков, требуется битовый массив, состоящий из n битов. Свободные блоки представлены в массиве в виде единиц, а распределенные — в виде нулей (или наоборот). В нашем примере с диском размером 1 Тбайт массиву необходимо иметь 1 млрд битов, для хранения которых требуется около 130 000 блоков размером 1 Кбайт каждый. Неудивительно, что битовый массив требует меньше пространства на диске, поскольку в нем используется по одному биту на блок, а не по 32 бита, как в модели,

использующей связанный список. Только если диск почти заполнен (то есть имеет всего несколько свободных блоков), для системы со связанными списками требуется меньше блоков, чем для битового массива.

Дисковые квоты

Чтобы не дать пользователям возможности захватывать слишком большие области дискового пространства, многопользовательские операционные системы часто предоставляют механизм навязывания дисковых квот. Замысел заключается в том, чтобы системный администратор назначал каждому пользователю максимальную долю файлов и блоков, а операционная система гарантировала невозможность превышения этих квот. Далее будет описан типичный механизм реализации этого замысла.

Когда пользователь открывает файл, определяется местоположение атрибутов и дисковых адресов и они помещаются в таблицу открытых файлов, находящуюся в оперативной памяти. В числе атрибутов имеется запись, сообщающая о том, кто является владельцем файла. Любое увеличение размера файла будет засчитано в квоту владельца.

Резервное копирование файловой системы

Резервное копирование на ленту производится обычно для того, чтобы справиться с двумя потенциальными проблемами:

♦ восстановлением после аварии;

♦ восстановлением после необдуманных действий (ошибок пользователей).

Производительность файловой системы

Доступ к диску осуществляется намного медленнее, чем доступ к оперативной памяти

Во многих файловых системах применяются различные усовершенствования, предназначенные для повышения производительности:

· Наиболее распространенным методом сокращения количества обращений к диску является блочное кэширование или буферное кэширование. В данном контексте кэш представляет собой коллекцию блоков, логически принадлежащих диску, но хранящихся в памяти с целью повышения производительности.

· Второй метод, улучшающий воспринимаемую производительность файловой системы, заключается в попытке получить блоки в кэш еще до того, как они понадобятся, чтобы повысить соотношение удачных обращений к кэшу.

· Еще одним важным методом является сокращение количества перемещений головок диска за счет размещения блоков с высокой степенью вероятности обращений последовательно, рядом друг с другом, предпочтительно на одном и том же цилиндре.

48. Различные виды файловых систем.

В процессе эволюции компьютеров, носителей информации и операционных систем возникало и пропадало большое количество файловых систем. В процессе такого эволюционного отбора, на сегодня для работы с жесткими дисками и внешними накопителями (флешки, карты памяти, внешние винчестеры, компакт диски) в основном используются следующие виды ФС:

1. NTFS
2. FAT32
3. Ext3

4. Ext4
5. NFS+
6. UDF
7. ISO9660

Последние две системы предназначены для работы с компакт дисками. Файловые системы Ext3 и Ext4 работают с операционными системами на основе Linux. NFS Plus – это ФС для операционных систем OS X, используемых в компьютерах фирмы Apple.

Самое большое распространение получили файловые системы NTFS и FAT32 и это не удивительно, т.к. они предназначены для операционных систем Windows, под управлением которых работает подавляющее большинство компьютеров в мире.Сейчас FAT32 активно вытесняется более продвинутой системой NTFS по причине ее большей надежности к сохранности и защите данных. К тому же последние версии ОС Windows просто не дадут себя установить, если раздел жесткого диска будет отформатирован в FAT32. Программа установки потребует отформатировать раздел в NTFS.

Файловая система NTFS поддерживает работу с дисками объемом в сотни терабайт и размером одного файла до 16 терабайт.

Файловая система FAT32 поддерживает диски до 8 терабайт и размер одного файла до 4Гб. Чаще всего данную ФС используют на флешках и картах памяти. Именно в FAT32 форматируют внешние накопители на заводе.

(файловые системы в линухе,вставлю потом краткое описание их https://mirror.yandex.ru/altlinux/4.0/Desktop/4.0.2/docs/linux_fstypes/index.html)

49. Управление файловой системой Unix/Linux.

Файловая система является краеугольным камнем операционной системы UNIX. Она обеспечивает логический метод организации, восстановления и управления информацией.

Строго говоря, следует различать физическую файловую систему, которая отвечает за управление дисковым пространством и размещение файлов в физических адресах диска и логическую файловую систему, которая обеспечивает логическую структуру хранения файлов – пространство имен файлов. ОС Unix и Linux могут работать с различными физическими файловыми системами, но логическое представление файловой системы в Unix/Linux всегда одинаково.

Все файлы, с которыми могут манипулировать пользователи, располагаются в файловой системе, представляющей собой дерево, промежуточные вершины которого соответствуют каталогам, и листья - файлам и пустым каталогам. Реально на каждом логическом диске (разделе физического дискового пакета) располагается отдельная иерархия каталогов и файлов. Для получения общего дерева в динамике используется "монтирование" отдельных иерархий к фиксированной корневой файловой системе в качестве ветвей общего дерева.

Каждый каталог и файл файловой системы имеет уникальное полное имя – имя, задающее полный путь, оно задает полный путь от корня файловой системы через цепочку каталогов к соответствующему каталогу или файлу). Каталог, являющийся корнем файловой системы (корневой каталог), в любой файловой системе имеет предопределенное имя "/" (слэш). Этот же символ используется как разделитель имен в пути. Полное имя файла, например, /bin/sh означает, что в корневом каталоге должно содержаться имя каталога bin, а в каталоге bin должно содержаться имя файла sh. Коротким или относительным именем файла называется имя (возможно, составное), задающее путь к файлу от текущего рабочего каталога (существует команда и соответствующий системный вызов, позволяющие установить текущий рабочий каталог).

В каждом каталоге содержатся два специальных имени, имя ".", именующее сам этот каталог, и имя "..", именующее "родительский" каталог данного каталога, т.е. каталог, непосредственно предшествующий данному в иерархии каталогов

50. Командный интерпретатор Unix/Linux.

Операционная система должна предоставлять удобный интерфейс пользователю, работающему за компьютером. В настоящее время получило

распространение два вида интерфейсов: графический и алфавитно-цифровой или текстовый. ОС Linux дает возможность использовать интерфейсы любого из этих видов.

Задачи администрирования системы обычно решаются в текстовом режиме. Основным посредником между пользователем и системой в текстовом режиме является командный интерпретатор. Вкратце его роль можно охарактеризовать так: интерпретатор должен постоянно ожидать ввода команд пользователя и при их получении выполнять соответствующие действия, как правило, выражающиеся в вызове других программ.

В действительности роль интерпретатора гораздо шире, он обладает значительным набором самых разнообразных возможностей, призванных сделать взаимодействие пользователя с компьютером предельно эффективным.

ОС Linux использует несколько различных видов интерпретаторов. Наиболее распространенными среди них являются:

- sh. Bourne Shell. Прообраз командных интерпретаторов сегодняшнего дня. В современных Linux-системах sh представляет собой символическую ссылку на файл bash;
- bash. Bourne-Again SHell. Основной командный интерпретатор ОС Linux. Представляет собой развитие ash и sh. Поддерживает богатый язык написания скриптов, удобный интерфейс для редактирования командной строки, автопродолжение команд и множество других полезных возможностей;
- tcsh. C Shell. Расширенная версия интерпретатора C Shell, использующегося в BSD-системах. Поддерживает функцию автозавершения текста и расширенные возможности редактирования;
- zsh. Очень развитый командный интерпретатор, объединяющий в себе возможности csh, bash с дополнительными, такими как: улучшенная поддержка автопродолжения, более развитые возможности редактирования, расширенные файловые шаблоны и ряд других;

- nash. Not A SHell. Предельно облегченная оболочка, предназначенная для интерпретации сценариев в linuxrc файлах, при загрузке с виртуального диска initrd. Не позволяет работать пользователю в интерактивном режиме.

http://xgu.ru/wiki/%D0%9A%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D0%BD%D1%8B%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D1%80%D0%B5%D1%82%D0%B0%D1%82%D0%BE%D1%80

51. Написание сценариев на языке SHELL.

Shell (шелл, он же «командная строка», он же CLI, он же «консоль», он же «терминал», он же «черное окошко с белыми буковками») -- это текстовый интерфейс общения с операционной системой

man — справка по командам и программам, доступным на вашей машине, а также по системным вызовам и стандартной библиотеке C.

#!/bin/sh – начало любого сценария

Проверенный и отлаженный shell-файл может быть вызван на исполнение, например, следующим способом:

$ chmod u+x shfil

$ shfil

$

Процедуре при ее запуске могут быть переданы аргументы. В общем случае командная строка вызова процедуры имеет следующий вид:

$ имя_процедуры $1 $2 ...$9

Некоторые вспомогательные операторы:

echo - вывод сообщений из текста процедуры на экран.

- для обозначения строки комментария в процедуре. (Строка не будет обрабатываться shell-ом).

banner - вывод сообщения на экран заглавными буквами (например для идентификации следующих за ним сообщений).

$banner 'hello ira'

Запуск set без параметров выводит список установленных системных переменных

shift - сдвинуть позиционные параметры влево на одну позицию

$имя_переменной -. на место этой конструкции будет подставлено значение переменной.

expr - вычисление выражений. a=`expr $a + 7`

Три формата команды экспортирования:

$export список имен локальных переменных

$export имя_лок_переменной=значение

$export (без параметров) - выводит перечень всех экспортированных локальных и переменных среды (аналог команды env).

Сравнение числовых значений:

$x=5

$[$x -lt 7]

$echo $?

test -ключ имя_файла

$[$X -eq $Y] - сравниваются значения чисел

$["$X" = "$Y"] - числа сравниваются как строки символов

Конструкции if then else

if

[$X -lt 10]

then

echo X is less 10

else

if

[$X -gt 10]

Then

Циклы

while список_команд1

do список_команд2

done

Цикл типа for:

for имя_ переменной [in список_значений]

do список_команд

done

Case:

echo -n 'Please, write down your age'

read age

case $age in

test $age -le 20) echo 'you are so young' ;;

test $age -le 40) echo 'you are still young' ;;

sleep t - приостанавливает выполнение процесса на t секунд

52. Аргументы сценария. Локальные переменные.

53. Управляющие конструкции языка SHELL.

Язык shell по своим возможностям приближается к высокоуровневым алгоритмическим языкам программирования. Операторы языка shell позволяют создавать собственные программы, не требуют компиляции, построения объектного файла и последующей компоновки, так как shell, обрабатывающий их, является транслятором интерпретирующего, а не компилирующего типа.

Shell имеет управляющие конструкции, которые позволяют осуществлять условное выполнение командной строки и циклы.

ЦЕЛИ

1. Использовать оператор test для:

– Проверки статуса файла (полномочия и размер).

– Сравнения символьных строк.

– Сравнения численных значений строк.

2. Использовать оператор if для условного выполнения командной строки.

3. Использовать управляющую конструкцию if-else для условного выполнения командной строки.

4. Использовать оператор exit для завершения процедур.

5. Использовать оператор while для циклического выполнения операторов и команд.

6 Использовать оператор continue для возвращения на вершину цикла.

7. Использовать оператор break для разрыва цикла, когда удовлетворены указанные условия.
8. Создать простую функцию внутри программы shell.

54. Инструментальные средства разработки Linux.

Для работы в Linux доступно потрясающее разнообразие средств разработки. Любому программисту, работающему в Linux, нужно ознакомиться с некоторыми наиболее важными из них.

Дистрибутивы Linux включают в себя множество серьезных и проверенных средств разработки; большинство из этих средств на протяжении нескольких лет входили в системы разработки под Unix. Средства разработки Linux не отличаются ненужными излишествами и броскостью; большинство из них представляют собой инструменты командной строки без графического интерфейса пользователя. За все годы их применения эти средства зарекомендовали себя с самой лучшей стороны, и их изучение лишним не будет.

Если вы уже знакомы с Emacs, vi, make, gdb, strace и ltrace, в этой главе ничего нового вы для себя не найдете. Тем не менее, в оставшейся части книги предполагаются хорошие знания какого-нибудь текстового редактора. Практически весь свободный исходный код Unix и Linux собирается при помощи make, а gdb — один из самых распространенных отладчиков, доступных для Linux и Unix. Утилита strace (или подобная утилита под названием trace либо truss) доступна в большинстве систем Unix; утилита ltrace была изначально написана для Linux и в большинстве систем недоступна (на момент написания книги).

Однако не стоит думать, что для Linux нет графических средств разработки; на самом деле, все как раз наоборот. Этих средств огромное количество.

На момент написания этой книги привлекали внимание две интегрированных среды разработки (Integrated Development Environment — IDE), которые могут

входить в используемый вами дистрибутив: KDevelop (http://kdevelop.org/), часть среды рабочего стола KDE, и Eclipse (http://eclipse.org/), межплатформенная среда, основанная на Java, которая первоначально, была разработана IBM, а теперь поддерживается крупным консорциумом. Однако в этой книге мы не будем останавливаться на рассмотрении упомянутых сред, поскольку они сопровождаются детальной документацией.

Даже несмотря на то, что для работы в Linux доступны многочисленные IDE, они пользуются не такой популярностью, как на других платформах. Даже если среда IDE применяется, все же более практичным считается написание программного обеспечения с открытым исходным кодом без ее задействования. Все это делается для того, чтобы другие программисты, которые захотят сделать свой вклад в ваш проект, не были стеснены вашим выбором IDE. Среда KDevelop поможет собрать проект, который будет использовать стандартные инструменты Automake, Autoconf и Libtool, используемые в многочисленных проектах с открытым исходным кодом.

Сами по себе стандартные средства Automake, Autoconf и Libtool играют важную роль в процессе разработки. Они были созданы для помощи в построений приложений таким образом, чтоб эти приложения могли быть почти автоматически перенесены в другие операционные системы. Ввиду того, что эти средства сложны, в настоящей книге мы рассматривать их не будем. Кроме того, эти средства регулярно изменяются; электронные версии GNU Autoconf, Automake и Libtool

55. Утилита Make.

> make — утилита предназначенная для автоматизации преобразования файлов из одной формы в другую. Правила преобразования задаются в скрипте с именем Makefile, который должен находиться в корне рабочей директории проекта. Сам скрипт состоит из набора правил, которые в свою очередь описываются:

1) целями (то, что данное правило делает);

2) реквизитами (то, что необходимо для выполнения правила и получения целей);

3) командами (выполняющими данные преобразования).

На всякий случай статья:

https://habr.com/ru/post/211751/

56. Безопасность операционных систем.

57. Угрозы. Атаки.

58. Формальные модели защищенных систем.

Наибольшее развитие получили два подхода, каждый из которых основан на своем видении проблемы безопасности и нацелен на решение определенных задач, — это формальное моделирование политики безопасности и криптография. Причем эти различные по происхождению и решаемым задачам подходы дополняют друг друга: криптография может предложить конкретные методы защиты информации в виде алгоритмов идентификации, аутентификации, шифрования и контроля целостности, а формальные модели безопасности предоставляют разработчикам основополагающие принципы, лежащие в основе архитектуры защищенной системы и определяющие концепцию ее построения.

Модель политики безопасности — формальное выражение политики безопасности. Формальные модели используются достаточно широко, потому что только с их помощью можно доказать безопасность системы, опираясь при этом на объективные и неопровержимые постулаты математической теории.

Основная цель создания политики безопасности ИС и описания ее в виде формальной модели — это определение условий, которым должно подчиняться поведение системы, выработка критерия безопасности и проведение формального доказательства соответствия системы этому критерию при соблюдении установленных правил и ограничений. На практике это означает, что только соответствующим образом уполномоченные пользователя получат доступ к информации и смогут осуществлять с ней только санкционированные действия.

Среди моделей политик безопасности можно выделить два основных класса: дискреционные (произвольные) и мандатные (нормативные). В данном подразделе в качестве примера изложены основные положения наиболее распространенных политик безопасности, основанных на контроле доступа субъектов к объектам.

Дискреционная модель Харрисона—Руззо — Ульмана. Модель безопасности Харрисона—Руззо —Ульмана, являющаяся классической дискреционной моделью, реализует произвольное управление доступом субъектов к объектам и контроль за распространением прав доступа.

В рамках этой модели система обработки информации представляется в виде совокупности активных сущностей — субъектов (множество S), которые осуществляют доступ к информации, пассивных сущностей — объектов (множество O), содержащих защищаемую информацию, и конечного множества прав доступа R = {r1, ..., rn}, означающих полномочия на выполнение соответствующих действий (например, чтение, запись, выполнение).

Причем для того чтобы включить в область действия модели и отношения между субъектами, принято считать, что все субъекты одновременно являются и объектами. Поведение системы моделируется с помощью понятия «состояние». Пространство состояний системы образуется декартовым произведением множеств составляющих ее объектов, субъектов и прав — OSR. Текущее состояние системы Q в этом пространстве определяется тройкой, состоящей из множества субъектов, множества объектов и матрицы прав доступа M, описывающей текущие права доступа субъектов к объектам, — Q = (S, O, M).

Строки матрицы соответствуют субъектам, а столбцы — объектам, поскольку множество объектов включает в себя множество субъектов, матрица имеет вид прямоугольника. Любая ячейка матрицы M[s, o] содержит набор прав субъекта s к объекту o, принадлежащих множеству прав доступа R. Поведение системы во времени моделируется переходами между различными состояниями.

Критерий безопасности модели Харрисона — Руззо — Ульмана формулируется следующим образом.

Для заданной системы начальное состояние Q0 = (S0, O0, M0) является безопасным относительно права r, если не существует применимой к Q0 последовательности команд, в результате которой право r будет занесено в ячейку матрицы M, в которой оно отсутствовало в состоянии Q0.

Нужно отметить, что все дискреционные модели уязвимы по отношению к атаке с помощью «троянского коня», поскольку в них контролируются только операции доступа субъектов к объектам, а не потоки информации между ними. Поэтому, когда «троянская» программа, которую нарушитель подсунул некоторому пользователю, переносит информацию из доступного этому объекта в объект, доступный нарушителю, то формально никакое правило дискреционной политики безопасности не нарушается, но утечка информации происходит.

Таким образом, дискреционная модель Харрисона—Руззо — Ульмана в своей общей постановке не дает гарантий безопасности системы, однако именно она послужила основой для целого класса моделей политик безопасности, которые используются для управления доступом и контроля за распределением прав во всех современных системах.

Типизованная матрица доступа. Другая дискреционная модель, получившая название «Типизованная матрица доступа» (Type Access Matrix — далее TAM), представляет собой развитие модели Харрисона—Руззо—Ульмана, дополненной концепцией типов, что позволяет несколько смягчить те условия, для которых возможно доказательство безопасности системы.

Состояние системы описывается четверкой Q = (S, O, t, M), где S, O и M обозначают соответственно множество субъектов, объектов и матрицу доступа, а t: O → T — функция, ставящая в соответствие каждому объекту некоторый тип.

Перед выполнением команды происходит проверка типов фактических параметров, и, если они не совпадают с указанными в определении, команда не выполняется. Фактически введение контроля типов для параметров команд приводит к неявному введению дополнительных условий, так как команды могут быть выполнены только при совпадении типов параметров.

Смысл элементарных операций совпадает со смыслом аналогичных операций их классической модели Харрисона — Руззо — Ульмана с точностью до использования типов.

Таким образом, ТАМ является обобщением модели Харрисона—Руззо—Ульмана, которую можно рассматривать как частный случай ТАМ с одним-единственным типом, к которому относятся все объекты и субъекты. Появление в каждой команде дополнительных неявных условий, ограничивающих область применения команды только сущностями соответствующих типов, позволяет несколько смягчить жесткие условия классической модели, при которых критерий безопасности является разрешимым.

Несмотря на различающиеся подходы, суть всех моделей безопасности одинакова, поскольку они предназначены для решения одних и тех же задач. Целью построения модели является получение формального доказательства безопасности системы, а также определения достаточного критерия безопасности.

Безопасность системы определяется в равной степени тремя факторами: свойствами самой модели, ее адекватностью угрозам, воздействующим на систему, и тем, насколько корректно она реализована. Поскольку при существующем разнообразии теоретических наработок в области теории информационной безопасности выбор модели, адекватной заданным угрозам, не

является проблемой, последнее, решающее слово остается за ее реализацией в защищенной системе.

59. Оранжевая книга безопасности.

Критерии определения безопасности компьютерных систем (англ. *Trusted Computer System Evaluation Criteria*) — стандарт Министерства обороны США, устанавливающий основные условия для оценки эффективности средств компьютерной безопасности, содержащихся в компьютерной системе. Критерии используются для определения, классификации и выбора компьютерных систем, предназначенных для обработки, хранения и поиска важной или секретной информации.

Политики

Политики безопасности должны быть подробными, чётко определёнными и обязательными для компьютерной системы. Есть две основных политики безопасности:

- Мандатная политика безопасности — обязательные правила управления доступом напрямую, основанные на индивидуальном разрешении, разрешении на доступ к информации и уровне конфиденциальности запрашиваемой информации. Другие косвенные факторы являются существенными и окружающими. Эта политика также должна точно соответствовать закону, главной политике и прочим важным руководствам, в которых устанавливаются правила.
  - Маркирование — системы, предназначенные для обязательной мандатной политики безопасности, должны предоставлять и сохранять целостность меток управления доступом, а также хранить метки, если объект перемещён.
- Дискреционная политика безопасности — предоставляет непротиворечивый набор правил для управления и ограничения доступа, основанный на идентификации тех

пользователей, которые намерены получить только необходимую им информацию.

Ответственность

Индивидуальная ответственность в независимости от политики должна быть обязательной. Есть три требования по условиям ответственности:

· Аутентификация — процесс, используемый для распознавания индивидуального пользователя.

· Авторизация — проверка разрешения индивидуальному пользователю на получение информации определённого рода.

· Аудит — контролируемая информация должна избирательно храниться и защищаться в мере, достаточной для отслеживания действий аутентифицированного пользователя, затрагивающих безопасность.

Гарантии

Компьютерная система должна содержать аппаратные и/или программные механизмы, которые могут независимо определять, обеспечивается ли достаточная уверенность в том, что система исполняет указанные выше требования. Вдобавок уверенность должна включать гарантию того, что безопасная часть системы работает только так, как запланировано. Для достижения этих целей необходимо два типа гарантий и соответствующих им элементов:

· Механизмы гарантий

  o Операционная гарантия — уверенность в том, что реализация спроектированной системы обеспечивает осуществление принятой стратегии защиты системы. Сюда относятся системная архитектура, целостность системы, анализ скрытых каналов, безопасное управление возможностями и безопасное восстановление.

  o Гарантия жизненного цикла — уверенность в том, что система разработана и поддерживается в соответствии с

формализованными и жёстко контролируемыми критериями функционирования. Сюда относятся тестирование безопасности, задание на проектирование и его проверка, управление настройками и соответствие параметров системы заявленным.

- Гарантии непрерывной защиты — надёжные механизмы, обеспечивающие непрерывную защиту основных средств от преступных и/или несанкционированных изменений.

Документирование

В каждом классе есть дополнительный набор документов, который адресован разработчикам, пользователям и администраторам системы в соответствии с их полномочиями. Эта документация содержит:

- Руководство пользователя по особенностям безопасности.
- Руководство по безопасным средствам работы.
- Документация о тестировании.
- Проектная документация

https://habr.com/ru/sandbox/9033/

60. Политики безопасности.

61. Разграничение доступа к ресурсам компьютера.

62. Аутентификация и авторизация в ОС.

Идентификация субъекта доступа заключается в том, что субъект сообщает системеидентифицирующую информациюо себе (имя, учетный номер и т.д.) и таким образом идентифицирует себя.

Аутентификация(authentification) предотвращает доступ к сети нежелательных лиц и разрешает вход для легальных пользователей. Процесс аутентификации следует отличать от процесса идентификации.

Термин «аутентификация» в переводе с латинского означает «установление подлинности». Аутентификацию следует отличать от идентификации. Идентификаторы пользователей используются в системе с теми же целями, что и идентификаторы любых других объектов, файлов, процессов, структур данных, но они не связаны непосредственно с обеспечением безопасности. Идентификация заключается в сообщении пользователем системе своего идентификатора, в то время как аутентификация – это процедура доказательства пользователем того, что он есть тот, за кого себя выдает, в частности, доказательство того, что именно ему принадлежит введенный им идентификатор.

В процедуре аутентификации участвуют две стороны: одна сторона доказывает свою аутентичность, предъявляя некоторые доказательства, а другая сторона - аутентификатор - проверяет эти доказательства и принимает решение. В качестве доказательства аутентичности используются самые разнообразные приемы:

аутентифицируемый может продемонстрировать знание некоего общего для обеих сторон секрета: слова (пароля) или факта (даты и места события, прозвища человека и т. п.);

аутентифицируемый может продемонстрировать, что он владеет неким уникальным предметом (физическим ключом), в качестве которого может выступать, например, электронная магнитная карта;

аутентифицируемый может доказать свою идентичность, используя собственные биохарактеристики: рисунок радужной оболочки глаза или отпечатки пальцев, которые предварительно были занесены в базу данных аутентификатора.

Сетевые службы аутентификации строятся на основе всех этих приемов, но чаще всего для доказательства идентичности пользователя используются

пароли. Простота и логическая ясность механизмов аутентификации на основе паролей в какой-то степени компенсирует известные слабости паролей. Это, во-первых, возможность раскрытия и разгадывания паролей, а, во-вторых, возможность «подслушивания» пароля путем анализа сетевого трафика. Для снижения уровня угрозы от раскрытия паролей администраторы сети, как правило, применяют встроенные программные средства для формирования политики назначения и использования паролей: задание максимального и минимального сроков действия пароля, хранение списка уже использованных паролей, управление поведением системы после нескольких неудачных попыток логического входа и т. п. Перехват паролей по сети можно предупредить путем их шифрования перед передачей в сеть.

В качестве объектов, требующих аутентификации, могут выступать не только пользователи, но и различные устройства, приложения, текстовая и другая информация. Так, например, пользователь, обращающийся с запросом к корпоративному серверу, должен доказать ему свою легальность, но он также должен убедиться сам, что ведет диалог действительно с сервером своего предприятия. Другими словами, сервер и клиент должны пройти процедуру взаимной аутентификации. Здесь мы имеем дело с аутентификацией на уровне приложений. Аутентификация данных означает доказательство целостности этих данных, а также того, что они поступили именно от того человека, который объявил об этом. Для этого используется механизм электронной подписи.

Авторизация субъекта доступа происходит после успешной идентификации и аутентификации. При авторизации субъекта операционная система выполняет действия, необходимые для того, чтобы субъект мог начать работу в системе. Например, авторизация пользователя в операционной системе UNIX включает в себя порождение процесса, являющегося операционной оболочкой, с которой в дальнейшем будет работать пользователь. В операционной системе Windows NT авторизация пользователя включает в себя создание маркера доступа пользователя, создание рабочего стола и запуск на нем от имени авторизуемого пользователя процессаUserinit, инициализирующего индивидуальную программную среду пользователя. Авторизация субъекта не относится напрямую к подсистеме защиты операционной системы. В процессе авторизации решаются чисто технические задачи, связанные с организацией начала работы в системе уже идентифицированного и аутентифицированного субъекта доступа.

Процедуры авторизации реализуются программными средствами, которые могут быть встроены в операционную систему или в приложение, а также могут поставляться в виде отдельных программных продуктов. При этом программные системы авторизации могут строиться на базе двух схем:

централизованная схема авторизации, базирующаяся на сервере;

децентрализованная схема, базирующаяся на рабочих станциях.

В первой схеме сервер управляет процессом предоставления ресурсов пользователю. Главная цель таких систем – реализовать «принцип единого входа». В соответствии с централизованной схемой пользователь один раз логически входит в сеть и получает на все время работы некоторый набор разрешений по доступу к ресурсам сети. Система Kerberosc ее сервером безопасности и архитектурой клиент-сервер является наиболее известной системой этого типа. Системы TACACS и RADIUS, часто применяемые совместно с системами удаленного доступа, также реализуют этот подход.

63. Средства защиты от несанкционированного доступа.

Для хранения паролей в UNIX использовали следующую схему. Пользователь вводит пароль, затем он шифруется и помещается в файл. Затем при попытке входа вводимый пароль хешируется и происходит поиск такого пароля в файле. Для защиты от атак с подбором нужного пароля используется соль, случайное число, которое вычисляется при каждом изменении.

Один из подходов заключается в использовании брандмауэров (firewall), которые являются всего лишь современной адаптацией средневекового вспомогательного защитного средства: выкапывания глубокого рва вокруг своего замка. Эта конструкция заставляет всех входить или выходить из замка, проходя по 9.10. Средства защиты 755 единственному подъемному мосту, где они могут быть проверены полицией, следящей за режимом входа-выхода. Тот же прием возможен и при работе в сети: у

компании может быть множество локальных сетей, соединенных в произвольном порядке, но весь информационный поток, направленный в компанию или из нее, принудительно направлен через электронный подъемный мост — брандмауэр.