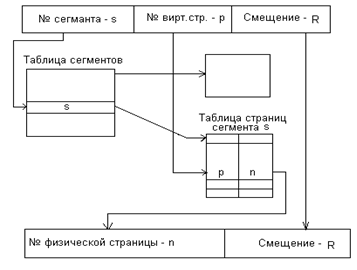
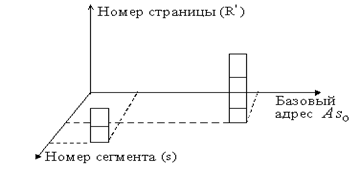
**Постройте схему сегметно-страничной структуризации и схему механизма преобразования виртуального**

**адреса в физический адрес**

Выполняются следующие

действия для получения

такой структуризации:

1. исходное пространство

структурируют фиксированными

страницами;

2. сегмент рассматривается как непрерывная по-

следовательность номеров страниц (размер сегмента — количество страниц);

3. каждому сегменту присваивается уникальный номер 𝑆;

4. в пределах каждого сегмента осуществляют перенумерацию

страниц, начиная с 0 и возрастая;

В итоге, сегменту назначается базовый адрес 𝐴𝑠0. В итоге адрес указывается с помощью четырёх координат

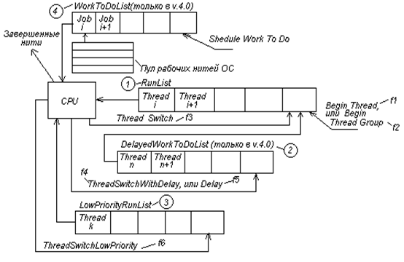
(𝑆,𝐴𝑠0,𝑅′,𝑅). R – номер страницы в пределах сегмента, R’ – смещение в пределах страницы. Базовый адрес

страницы 𝐴б=𝐴𝑠0+𝑅∗𝐿

Если размер страницы кратен 2, то к базовому адресу страницы присоединяется операцией конкатенации

размер смещения.

Механизм преобразования виртуального адреса в физический - для каждого сегмента создаётся своя таблица страниц. Для каждого процесса — своя таблица сегментов. При этом адрес таблицы сегментов процесса загружается в специальный регистр процессора, когда процесс переходит в активное состояние.

**Изобразите схему системы очередей планирования и опишите назначение очередей**

При создании нити с помощью функции f1

или f2 она попадает в конец очереди

1 (RunList (список запуска), содер-

жит готовые к выполнению нити).

Планировщик выбирает первую нить из

очереди 1 и запускает её на выполнение при

условии, что очередь 4 пуста.

Если нити требуются ещё итерации, то

она помещается в конец одной из очере-

дей в зависимости от того, какой вызов

управления был использован.

В конец очереди 1 при вызове f3, в

конец очереди 2 при вызове f4 или f5, в

конец очереди 3 при вызове f6.

Нити находятся в очереди 2, после за-

вершения попадают в очередь 1.

Нити из очереди 3 допускаются только

в том случае, если очередь 4 пустая. Обы-

чно в эту очередь попадают нити, выполняющие фоновую работу (не срочную).

Очередь 4 — самая приоритетная. Планировщик разрешает выполняться подряд только определённому количеству нитей из очереди 4, а затем запускает очередь 1. Очередь 4 была введена для повышения производительности NLM-приложений (Novell Load Module).

Рассмотренный механизм организации многонитевой работы сочетается со средствами синхронизации (семафорами и сигналами).

**Изобразите схему взаимодействия файловой системы и опишите назначение уровней**

Файл – поименованная целостная совокупность данных на внешнем носителе, на который наложена

структура.

Файловая система – часть ОС, обеспечивающая выполнение операций над файлами.

****

На уровне идентификации: - по символьному имени файла определяется его уникальное имя (если в ОС один файл может иметь несколько символьных имён); - по уникальному имени определить атрибуты файла;

- сравнение полномочий пользователя или процесса с правами доступа к файлу.

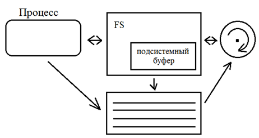
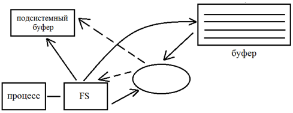
На логическом уровне:

- определяются координаты логической записи в файле; - алгоритм зависит от модели логической организации.

На физическом уровне:

- определяется номер физического блока (или блоков), который содержит требуемую логическую запись.

**Изобразите схему кэширования диска и опишите данный процесс**

**** В некоторой ФС при работе с внешними устройствами используется подсистема буферизации, которая работает по принципу КЭШ-памяти.

Запрос к внешнему устройству, в котором адресация осуществляется блоками, может быть перехвачен от систем буферизации. Такая система представляет собой буферный пул – совокупность однородных динамически распределенных блоков ОП одинаковой длины. Каждый буфер пула имеет размер, равный 1ому блоку.

Механизм работы: при поступлении запроса на чтение некоторого блока подсистемой буфер просматривает сначала свой буферный пул: - в случае обнаружения нужного блока копирует его в буфер запрашиваемого процесса (т.е. без обращения к устройству); - если блок не обнаружен, то он считывает и одновременно передает процессу и записывает его в буфер подсистемы.

Может быть, управление чтением нескольких блоков. При отсутствии свободного буфера на диск вытесняется самая редко используемая информация. Например, в Windows размер кэша может меняться в зависимости от текущей ситуации. При интенсивной работе в сети Windows увеличивает размер кэша, а при снижении интен-сивности и запуске большого числа приложений кэш автоматически уменьшается.

Достоинства: фрагментация меньше, срок службы диска увеличивается, скорость увеличивается.

**Приведите примеры управления КЭШ-ем**

Кэш — промежуточный буфер с быстрым доступом, содержащий информацию, которая может быть запрошена с наибольшей вероятностью. Доступ к данным в кэше осуществляется быстрее, чем выборка исходных данных из более медленной памяти или удалённого источника, однако её объём существенно ограничен по сравнению с хранилищем исходных данных.

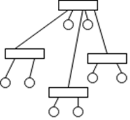
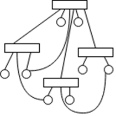
Работа кэша: при поступлении запроса на чтение некоторого блока подсистема буферизации сначала просматри-вает свой буфферный пул. В случае обнаружения нужного блока система копирует его в буфер запрашивающего процесса (то есть без обращения к диску) и если блок не обнаружен, то он считывается, записывается в буфер и передается процессу. Если буфер переполнен, то на диск вытесняется самая редко используемая информация.

Применение разных типов кеширования

- ленивый кэш – это самый простой вид кеширования, но его нужно использовать осторожно, так как отдает устаревшие данные. Можно при каждой записи сбрасывать ленивый кеш, чтобы поддерживать актуальность данных, но тогда затраты на реализацию будут сравнимы с более сложными типами кеширования. Такой тип кеширования можно использовать для данных, которые почти никогда не меняются. Другой вариант использова-ния – делать ленивый кеш с небольшим временем устаревания для стабильной работы при всплесках нагрузки. Такой тип кеширования позволит быстрее всех дать ответ.

- синхронизированный кэш – это самый полезный тип кеширования, так как отдает свежие данные и позволяет реализовать многоуровневый кеш. Такой тип кеширования встроен в протокол HTTP. Сервер отдает метку измене-ния, а клиент кеширует у себя результат и в последующем запросе передает эту метку. Сервер может дать ответ, что состояние не изменилось и можно использовать кэшированный на клиенте объект. Сервер в свою очередь, по-лучив метку, может переспросить у хранилища: были ли изменения или нет. Этот тип кеширования не избавляет от накладных расходов на общение между системами. Поэтому часто дополняется другими типами кеширования, что-бы ускорить работу.

- кэш сквозной записи. Если есть система распределенного кеширования (memcached, Windows Sever App Fabric, Azure Cache), то можно использовать кеш сквозной записи. Рукопашная реализация синхронизации кешей между узлами сама по себе отдельный большой проект, потому не стоит заниматься ей в рамках разработки приложения. Не стоит пытаться кешировать все в синхронизированном кеше, иначе большая часть кода приложения будет зани-маться перестройкой кеша. Также не стоит забывать, что системы распределенного кеширования также требуют общения между системами, что может сказываться на быстродействии.

**Опишите способы логической организации файловой системы**

Иерархия каталогов может быть деревом или сетью. Каталоги образу-

ют дерево, если файлу разрешено входить только в один каталог, и сеть

- если файл может входить сразу в несколько каталогов. В Windows

каталоги образуют древовидную структуру, а в UNIX - сете-

вую. Как и любой другой файл, каталог имеет символьное имя и

однозначно идентифицируется составным именем, содержащим

цепочку символьных имён всех каталогов, через которые проходит путь

от корня до данного каталога.

1-ая картинка – сеть, 2-ая – дерево

Пользователь работает на уровне логических записей.

Логическая запись — это наименьший элемент данных, который доступен пользователю для выполнения различных операций.

**Приведите классификацию процессов для современной операционной системы, опишите их основное назначение и приведите примеры (на примере Linux)**

1) Системные процессы являются частью ядра и всегда находятся в оперативной памяти. Такие процессы не имеют соответствующих им программ в виде исполняемых файлов и запускаются особым образом при инициализации ядра системы.

Примеры: shed (диспетчер свопинга), vhand (диспетчер страничной развертки), bdfflush (диспетчер буферного кэша), kmadaemon (диспетчер памяти ядра), init (является прородителем всех остальных процессов, но не является частью ядра, а его запуск происходит из специального исполняемого файла).

2) Демоны – не интерактивные процессы (программы), которые запускаются путем загрузки в память соответст-вующих им программ (исполняемых файлов), и они выполняются в фоновом режиме. Обычно демоны запускаются при инициализации системы, но после инициализации ядра. Они обеспечиваются работу различных систем, например системы терминального доступа, системы печати, системы сетевого доступа. Демоны не связаны ни с одним из пользовательских сеансов работы. Не могут непосредственно управлять пользователем. Большую часть временно спять, пока тот или иной процесс не запросит определенную услугу.

3) Прикладные процессы – все остальные процессы. Как правило порождаются пользовательским сеансом работы. Важным пользовательским процессом является интерпретатор, который обеспечивает работу Linux. Записывается после регистрации пользователя в системе, а завершение этого процесса приводит к отключению системы.

**Приведите примеры структур файлов**

Файлы с последовательной структурой. Файл рассматривается как одномерный массив составных элементов, называемых записями. Длина логической записи может быть как постоянная, так и переменная. Каждая логическая запись имеет свой логический номер в структуре файла. Доступ к элементам последовательный, т.е. после обра-ботки i-ой записи доступна только i+1. Для организации такого доступа достаточно иметь только 1 указатель на запись.

Файлы с индексно-последовательной структурой. Существует ряд методов, основанных на идентификации записи в файлах по некоторому ключу (т.е. его называют идентификационным отличительным признаком). Структура файла усложняется, но сокращается число обращений к диску. Здесь кроме полезных данных вводится ещё учетная информация. В данной структуре поиск элементов файла сначала проводится в прямом, а затем в последовательном порядке. Происходит упорядочивание всех записей по значению ключей, выделяются группы записей, ключи которых расположены в 1 файле и могут располагаться в пределах одной дорожки на диске. Для более быстрого поиска используют специальную структуру, например индекс дорожки. Каждый элемент диска описывает отдельную группу записи. Идентификатор может содержать ссылку на начальный адрес группы.

По идентификатору находят первую запись, затем внутри группы последовательным анализом ключей находят нужную запись. Недостаток: проблема расширения файла (требуется упорядоченность ключей). Для решения этой проблемы вводят область переполнения, в которую заносят новые записи, а из файла заносят ссылки.

Связный список блоков. В начале каждого блока содержится указатель на следующий блок и так далее. Дос-тоинства: Фрагментация практически отсутствует, файл может меняться во время работы, адрес файла задается номер первого блока. Недостатки: усложняется алгоритм распределения, сложность доступа к произвольной

части файла и увеличивается время поиска.

Перечень номеров блоков. Номера блоков, которые занимает файл, просто перечисляются. Достоинства: снима-ются проблемы динамического расширения, почти отсутствует фрагментация. Недостатки: увеличивается время доступа, то есть усложняется алгоритм распределения.

Пример. В UNIX реализован вариант, который позволяет обеспечить фиксированную длину индекса независимо от размера файла. Каждый файл в системе имеет дескриптор, в составе которого хранится список, который содержит 13 номеров блоков, которые могут принадлежать файлам.

Связный список индексов. В связном списке индексов с каждым блоком связывается индекс, расположенный в отдельной области диска. В DOS таблица называется FAT (File Allocation Table). Это позволяет отслеживать сос-тояние разных участков дискового пространства. Достоинства: файл — это ссылка на первый блок/индекс, фраг-ментация практически отсутствует, файл может расширяться, но и устраняются недостатки — для обращения к произвольному месту нужно прочитать индексы. Недостаток: больше трата памяти.

**Опишите свойства распределенных файловых систем и основные решаемые вопросы**

В основе лежит модель – клиент-сервер. Клиент – машина, которая обращается к некоторому файлу. Сервер – машина, хранящая файлы и обеспечивающая к ним доступ. Распределенные файловые системы имеют ряд важных свойств. Каждая конкретная система может обладать всеми или частью этих свойств.

- сетевая прозрачность (клиенты должны иметь возможность обращаться к удаленным файлам пользуясь теми же самыми операциями, что и для доступа к локальным файлам);

- прозрачность размещения (имя файла не должно определять его местоположения в сети);

- независимость размещения (имя файла не должно меняться при изменении его физического месторасположения);

- мобильность пользователя (пользователи должны иметь возможность обращаться к разделяемым файлам из любого узла сети);

- устойчивость к сбоям (система должна продолжать функционировать при неисправности отдельного компонента (сервера или сегмента сети), однако это может приводить к деградации производительности или к исключению доступа к некоторой части файловой системы);

- масштабируемость (система должна обладать возможностью масштабирования в случае увеличения нагрузки, кроме того, должна существовать возможность постепенного наращивания системы путем добавления отдельных компонентов);

- мобильность файлов (должна быть возможность перемещения файлов из одного месторасположения в другое на работающей системе).

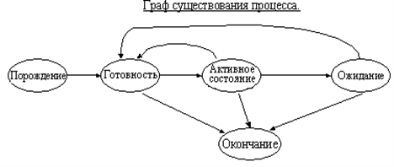
Имеется несколько важных вопросов, которые рассматриваются при разработке распределенных файловых систем:

- пространство имен (некоторые распределенные файловые системы обеспечивают однородное пространство имен (каждый клиент использует одно и то же путевое имя для доступа к данному файлу), другие системы позволяют клиенту создавать свое пространство имен путем монтирования разделяемых поддеревьев к произвольным каталогам в иерархии файлов);

- операции с сохранением и без сохранения состояний (сервер, сохраняющий состояния, обеспечивает хранение информации об операциях клиента между запросами и использует эту информацию о состоянии для корректного обслуживания последующих запросов);

- семантика разделения (распределенная файловая система должна определить семантику, которая применяется, когда несколько клиентов одновременно обращаются к одному файлу);

- методы удаленного доступа (в простой модели клиент-сервер используется метод удаленного обслуживания, при котором каждое действие инициируется клиентом, а сервер просто представляет собой агента, который выполняет заявки клиента).

**Постройте граф существования процесса и опишите условия перехода из одного состояния в другое**

Порождение — подготавливаются все условия для исполне-

ния. Готовность — все ресурсы предоставлены (кроме процес-

сора). Здесь процесс ждёт, когда диспетчер (планировщик)

его выберет. Активное состояние — процесс обладает

всеми ресурсами и использует

непосредственно процессор. Окон-

чание — процесс завершается. Может

быть аварийным или нормальным. Ожидание — процесс может

быть прерван по ряду причин (два вида):

1 род прерываний — процесс сам является причиной прерывания:

- при попытке получить ресурс или отказаться от ресурса;

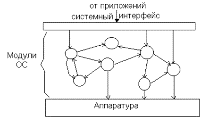
- при порождении, уничтожении или другом действии к другим процессам;

- алгоритмическое переполнение, попытка обращения к защищённой памяти и т. п.

2 род прерываний — процесс не является причиной прерывания:

- процесс может быть вытеснен более приоритетным процессом или по окончании кванта времени;

- другие прерывания, связаны с необходимостью проведения синхронизации между параллельными процессами.

**Сформулируйте концепции проектирования операционных систем**

1.Монолитные ОС. ОС в виде набора процедур, где каждая может вызывать лю-

бую другую, т. е. отсутствуют правила вертикального управления.

Для построения монолитной ОС необходимо было скомпилировать все не-

обходимые процедуры и связать в единый объектный файл с помощью

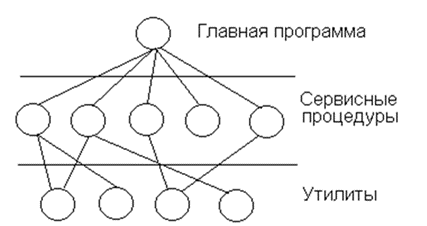
компоновщика (Unix, Novel NetWare).

2. Структурированная монолитная ОС. При обращении к системным вызо-

вам параметры помещаются в строго отведённом месте, а затем выполняется

специальная команда прерывания — вызов ядра (супервизора). Команда перек-

лючает машину из режима пользования в режим ядра, проверяя параметры вызова, индексируется таблица

ссылок на процедуры и вызывается соответствующий процесс. В этой модели для каждого системного вызова имеется 1 сервисная процедура.

3. Многоуровневые иерархическое ОС. Уровни:

1 — распределение времени процессора, переключение процессов по

прерыванию или кванта времени.

2 — выполнение функции виртуальной оперативной памяти.

3 — управление связью между консолью оператора и процессора.

4 — управление устройствами ввода-вывода и буферизация потока.

5 — пользовательские программы.

6 — процесс системного оператора.

Каждый уровень может взаимодействовать только непосредственно с соседним.

4. Микроядерные ОС. Здесь компоненты используют средства микроядра для обмена сообщениями между компонентами: ядро проверяет "законность" сообщений, пересылку сообщений между компонентами, обеспечивает доступ к аппаратуре. Такой подход позволяет использовать микроядерные ОС в распределённых средах, т. к. микроядру без разницы, поступило сообщение от локального или распределённого процесса. Однако пересылка сообщений проводится медленнее обычных вызовов функций. Важная задача — оптимизация пересылки сообщений.

**Изобразите структуру современной операционной системы и опишите назначение основных компонентов**

 Структура операционной системы на примере MS-DOS.

BIOS осуществляет инициализацию вектора прерывания

нижнего уровня и считывает память NSB. BIOS содержит

драйверы стандартных периферийных устройств, тестовые

программы аппаратуры, программы начальной загрузки.

NSB — non-system bootstrap, внесистемный загрузчик, стар-

товый сектор физического жёсткого диска (вторичный загруз-

чик). Считывает в память и запускает SB

SB — системный загрузчик. Стартовый сектор каждого ло-

гического жёсткого диска. Он считывает в память EMBIOS,

BMDOS. Затем запускает EMBIOS.

EMBIOS — extension module BIOS, модуль расширения (файл IO.SYS). Он

осуществляет определение состояния оборудования, конфигурирует

DOS по указаниям в файле config.sys, также осуществляет инициализа-

цию и переустановку некоторых векторов прерывания нижнего уровня.

Запускает BMDOS.

BMDOS — basic module, центральный компонент, реализующий основные функции (управление ресурсами и программой), Основу составляют обработчики прерываний верхнего уровня. Осуществляет считывание в память CI и запускает CI — ответчик за поддержку пользовательского интерфейса (интерпретатор команд).

CI отвечает за поддержку пользовательского интерфейса. состоит из двух уровней:

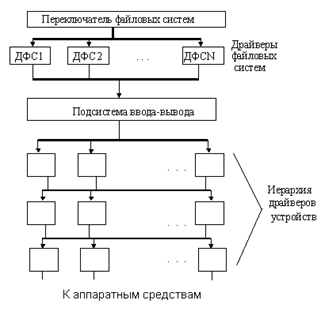
1) резидентного — содержит обработчик прерываний и код подгрузки транзитного модуля;

2) транзитного — отвечает за подгрузку и выполнение команд.

Внешние драйвера устройств — файлы управления периферийными устройствами.

DOS Shell («Оболочка DOS») — файловый менеджер

Утилита — обслуживающая программа, предоставляющая пользователю сервисные услуги (FORMAT.COM, CHKDSK.COM, FDISK.COM).

**Приведите схему современной файловой системы и опишите назначение основных блоков**

Современная ОС может работать сразу с несколькими ФС, имеет

многоуровневую структуру. Приложение обращается только через пере-

ключатели (в Windows такой переключатель называется установочный

диспетчер ФС (IFS)).

Переключатель преобразует запросы в формат следующего уровня

конкретной ФС. Каждый ДФС (драйвер) поддерживает определенную

организацию ФС. ДФС позволяет сразу нескольким приложениям выпол-

нять операции над файлом. Подсистема ввода/вывода отвечает за загрузку,

инициализацию, управление всеми моделями нижнего уровня (драйверы

портов). Постоянно присутствует в памяти и обеспечивает совместную

работу иерархии драйверов устройств. Каждый уровень драйверов устро-

йств представляет собой определенный тип: драйвер жестких дисков, драй-

веры, перехвата запроса к блочным устройствам, драйверы портов, управ-

ляющие конкретными адаптерами.

**Опишите организацию работы процесса на примере современной операционной системы**

Процесс — заявка на потребление

системных ресурсов. Процесс, как

логическая единица, предполагает два

аспекта — он выполняет операции, и

он является носителем данных. В нём

можно выделить две части: программа,

по которой он работает в активном

состоянии; дескриптор процесса.

Дескриптор процесса содержит та-

куюинформацию о процессе, которая

необходима ядру в течение всего жиз-

ненного цикла процесса, независимо

от того, находится ли он в активном

или пассивном состоянии, находится

ли образ процесса в оперативной

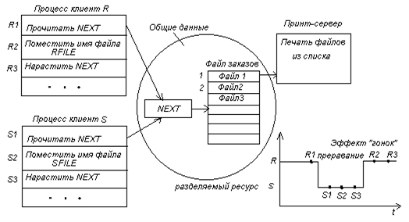
памяти или выгружен на диск. Дескрипторы отдельных процессов объединены в список, образующий таблицу процессов. Память для таблицы процессов отводится динамически в области ядра. На основании информации, содержащейся в таблице процессов, операционная система осуществляет планирование и синхронизацию процессов.

Контекст процесса — это информация, о процессе, необходимая в активном состоянии. Контекст процесса содер-жит менее оперативную, но более объёмную часть информации о процессе, необходимую для возобновления вы-полнения процесса с прерванного места: содержимое регистров процессора, коды ошибок, выполняемых процессо-ром системных вызовов, информацию о всех открытых данным процессом файлов и незавершённых операциях ввода-вывода (указатели на структуры file) и другие данные, характеризующие состояние вычислительной среды в момент прерывания. Контекст, так же, как и дескриптор процесса, доступен только программам ядра, то есть нахо-дится в виртуальном адресном пространстве операционной системы, однако он хранится не в области ядра, а непо-средственно примыкает к образу процесса и перемещается вместе с ним, если это необходимо, из оперативной па-мяти на диск. В UNIX для процессов предусмотрены два режима выполнения: привилегированный режим — "сис-тема" и обычный режим — "пользователь". В режиме "пользователь" запрещено выполнение действий, связанных с управлением ресурсами системы, в частности, корректировка системных таблиц, управление внешними устрой-ствами, маскирование прерываний, обработка прерываний. В режиме "система" выполняются программы ядра, а в режиме "пользователь" — оболочка и прикладные программы.

Организация процессов в операционной системе UNIX. Существует два уровня — верхний и нижний. Исполь-зуются приоритеты (распределение процессов происходит по приоритетной схеме), осуществляется динамическое изменение приоритетов в зависимости от характеристик процессов. Например, на основе оценки соотношения t\_использованное/t\_прогнозируемое : при повышении соотношения приоритет уменьшается. Такой алгоритм об-ладает отрицательной обратной связью (если процесс захватил много времени, то его приоритет уменьшается, и наоборот). Перераспределение времени происходит детерменировано с дискретностью примерно 1 раз в секунду. Этот период называют квантном мультиплексирования процессора.

Для синхронизации используются события и специальные файлы для передачи данных между процессами. Такие файлы могут наследоваться порождёнными процессами. Файл прекращает своё существование, когда прекращают существовать все процессы, имеющий его дескриптор. Количество файлов таких не должно превышать максималь-но допустимое число дескрипторов. Файлы используются между родственными процессами.

**Обоснуйте необходимость синхронизации процессов на примере эффекта «гонок»**

 Существует два типа конфликтов: при

взаимодействии процессов между

собой и конфликты с общеразделя-

емыми ресурсами.

Многие процессы находятся в зави-

симости от других процессов. Пример

процесса печати файлов. Она печатает

по очереди все файлы, имена которых другие процессы

записывают по порядку в общедоступный

файл заказов. Имеется особая перемен-

ная next, она доступна всем процессам и

содержит номер первой свободной

позиции файла заказов. В итоге

пострадает процесс S. Такую ситуацию

называют «эффект гонок»:

- критическая секция – часть программы, в которой осуществляется доступ к разделенным данным;

- взаимные исключения – обеспечивает, чтобы в каждый момент в критической секции находится не более одного процесса.

Самый простой способ позволить процессу находиться в критической секции выполнять все прерывания. Опасно доверять управление системой пользовательскому процессу. Другой способ – использование блокирующих переменных. С каждым разделяемым ресурсом связана двоичная переменная. Если она ноль, то ресурс занят, единица – свободен. Однако операции проверки и установки, блокируемые переменной, должны быть неделимы. Запрещать все прерывания при выполнении данных операций.

Сообщения используются в распределенных системах, когда процессы имеют собственную ОП, а использовать разделяемую невозможно. Ограничения: Объем сообщения меньше ёмкости приведенного буфера, принимающий процесс не может обработать сообщения быстрее, чем они создаются обрабатывающим процессом.

Использование двоичной переменной позволяет организовать доступ критических ресурсов только к одному потоку, а симофор – синхронную работу нескольких потоков.

**Опишите способы разрешения конфликтов**

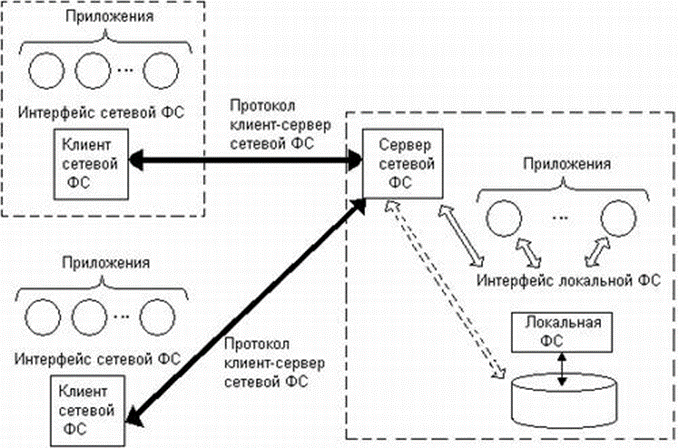
Взаимоблокировка: ситуация, когда конкуренция за ресурсы может привести к взаимоблокировке, т.е. все процессы не могут продолжать работу – дедлоки, клинчи и тупики. Взаимоблокировка — это тупиковая ситуация, характеризующаяся тем, что группа процессов ожидает события, которое может вызвать только другой процесс из этой же группы.

Способы восстановления: 1) через откат в ближайшую точку, когда всё было хорошо; 2) при помощи принудительной выгрузки ресурсов; 3) путем уничтожения процессов, которые в этом виноваты.

Проблемы синхронизации и взаимоисключения должны быть решены при помощи управления событиями и памятью. Синхронизация подразумевает сигнализацию между процессами по определенным протоколам. Протокол – набор правил и соглашений. Такая операция не зависит от времени и её принято называть событием.

Событие – объект синхронизации, используемый для информирования потоков о том, что событие произошло (что-то произошло).

В ОС допускается возникновение нескольких событий. Чтобы их отличать, каждому событию присваивается идентификатор, который называется флагом события (он определяет возможность синхронизации).

**Изобразите схему модели организации сетевой файловой системы и опишите принцип работы**

Клиенты сетевой ФС — это программы, которые работают

на многочисленных компьютерах, подключенных к сети. Эти

программы обслуживают запросы приложений на доступ к

файлам, хранящимся на удаленном компьютере. В качестве

таких приложений часто выступают графические или символь-

ные оболочки ОС.

Клиент сетевой ФС передает по сети запросы другому программному компоненту

- серверу сетевой ФС, работающему на удаленном компьютере.

Сервер, получив запрос, может выполнить его либо самосто-

ятельно, либо, что является более распространенным вариантом,

передать запрос локальной файловой системе для отработки.

После получения ответа от локальной файловой системы сервер

передает его по сети клиенту, а тот, — приложению, обративше-

муся с запросом.

Приложения обращаются к клиенту сетевой ФС, используя определенный программный интерфейс, который в данном случае является интерфейсом сетевой файловой системы. Этот интерфейс стараются сделать как можно более похожим на интерфейс локальной файловой системы, чтобы соблюсти принцип прозрачности. При полном совпадении интерфейсов приложение может обращаться к локальным и удаленным файлам и каталогам с помощью одних и тех же системных вызовов, совершенно не принимая во внимание места хранения данных.

Клиент и сервер сетевой файловой системы взаимодействуют друг с другом по сети по определенному протоколу. В случае совпадения интерфейсов локальной и сетевой файловых систем этот протокол может быть достаточно простым — в его функции будет входить ретрансляция серверу запросов, принятых клиентом от приложений, с которыми тот затем будет обращаться к локальной файловой системе.