Лекции по курсу
Операционные системы*

Преподаватель: Истратов Анатолий Юрьевич
TeX: Лана Гаварина
Оцифр. иллюстр.: Полуяктов Анатолий
Правки: Даниил Ковалев, Полуяктов Анатолий

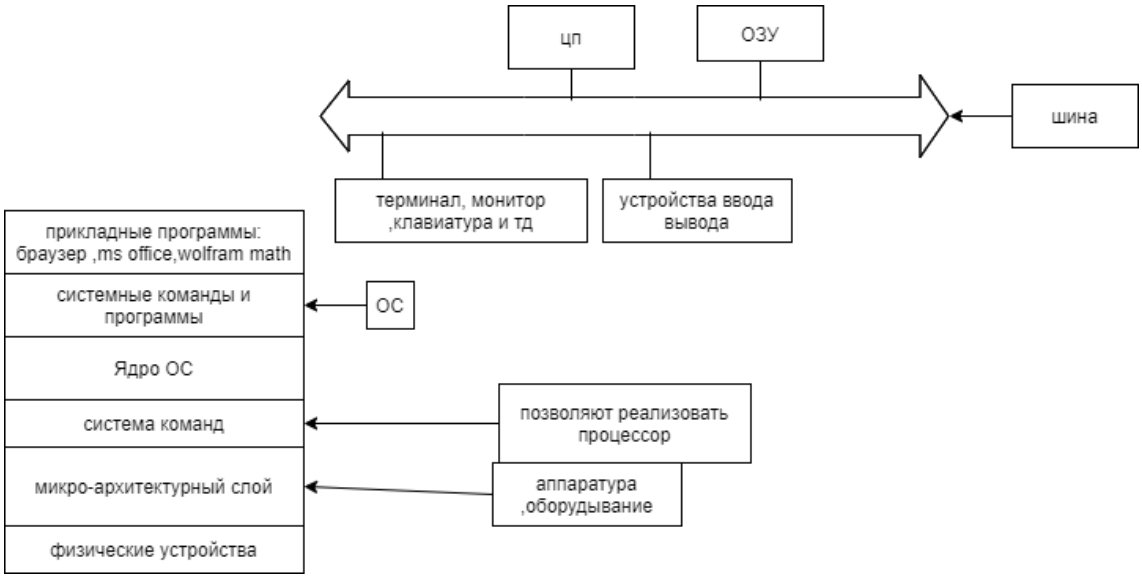
МИЭМ НИУ ВШЭ, 2020

*Семинары не оцифрованы, но все, что было на них теоретического, есть в билетах. По поводу опечаток/неточностей пишите ВК.

Содержание

1 Введение в ОС

Операционная система — набор программ и управляющих структур (таблиц), который обеспечит возможность использования аппаратуры компьютера и предоставит интерфейс для управления информацией и реализации ПО.



/На лекции также была картинка: ЦП + ОЗУ = компьютер, ЦП + ОЗУ + терминал, принтер, НМД, = вычислительная машина/

1.0.1 Функции ОС

1. Обеспечивает разделение ресурсов компьютера между программами;
2. Эффективное выполнение операций ввода-вывода;
3. Определяет интерфейс пользователя;
4. Восстановление информации и вычислительного процесса в случае ошибок.

1.0.2 Ресурсы

ОС управляет ресурсами:

1. Память (оперативная и внешняя);
2. Процессорное время (только для некоторых ОС);
3. Программы;
4. Периферийное оборудование, устройства ввода-вывода.

Типы ресурсов:

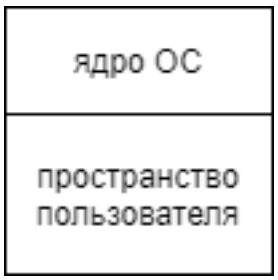
1. Выгружаемые (можно отнять у процесса, пример: ОП) / невыгружаемые (нельзя отнять у процесса, пример: принтер, компакт-диск, внешняя память)
2. Исчерпаемые (например, память, она ограничена) / неисчерпаемые (круговой конвейер, когда, например, с помощью " в Unix выход одной программы подается на вход другой)

ОС взаимодействует с:

1. Пользователями;
2. Системными и прикладными программами;
3. Аппаратными средствами.

1.0.3 Ядро ОС

Ядро ОС — та часть кода ОС, которая резидентно (постоянно) находится в ОП компьютера.



Функции ядра:

1. Управление устройствами ввода-вывода;
2. Управление прерываниями;
3. Управление файловой системой;
4. Управление процессами;
5. Распределение и перераспределение оперативной и внешней памяти.

1.1 Управление процессами

Процесс — программа в стадии выполнения вместе с текущими значениями счетчиков команд, регистров, переменных.

У каждого процесса есть адресное пространство — список адресов в памяти, в которые он может писать, и из которых можно читать. Оно содержит саму программу, данные к ней и ее стек.

Квант времени — по умолчанию равен 100 мс. Если за это время программа не завершилась, срабатывает прерывание, процесс ставится в очередь.

1.1.1 Состояния процесса

В период существования процесс проходит через ряд дискретных состояний:

1. Выполняется, если ему предоставлен (выделен) ЦП
2. Готов, если мог бы сразу использовать предоставленный ЦП
3. Блокирован, если ожидает появления некоторого события

Приоритеты процессов постоянно пересчитываются. В Unix 4 очереди на выполнение процессов, в Windows 8 (или даже 16).

Когда в системе поступает на выполнение программа, ОС создает процесс. Присваивает ему имя, начальный приоритет, другие атрибуты дескриптора процесса (см. далее). Эта информация заносится в таблицу процессов в первую свободную запись.

Сам процесс записывается в список готовых к выполнению. Со временем он окажется первым в этом списке, и ОС выделит ему ЦП для выполнения. Происходит смена состояния процесса (готов $\rightarrow$ выполняется). После отработки кванта времени (или преждевременном завершении процесса) состояние меняется (выполняется $\rightarrow$ готов).

1.1.2 Дескриптор процесса

Представителем процесса в ОС является дескриптор процесса (= блок управления, БУП).

Фоновый процесс — процесс, не управляемый терминалами (не привязанный к терминалу). Он не может читать из терминала и писать в него. Для создания такого процесса из командной строки в конце команды нужно поставить символ **&**, из программы его можно создать при помощи системного вызова setsid. Такие процессы также называют демонами.

Дескриптор содержит:

1. Идентификатор (= имя)
2. Приоритет (число, меняется)
3. Состояние (готов, выполняется, блокирован)
4. Указатели адресов памяти (начальное/конечное значение памяти + объем)
5. Указатели выделенных ресурсов
6. Машинные регистры (область сохраненных регистров)

ОС ведет таблицу, и каждая запись таблицы характеризует один процесс.

1.1.3 Операции над процессами

1. Создание
   (a) Присвоение имени (pid) процессу
   (b) Включение этого pid в список имен процессов, известных ОС
   (c) Определение начального приоритета процесса
   (d) Формирование дескриптора процесса
   (e) Выделение процессу требуемых ресурсов. Если нужных ресурсов нет, то процесс создастся и будет в состоянии блокировки, пока ресурс не освободится
2. Уничтожение
   (a) Выполнение системного вызова exit
   (b) Удаление записи из таблицы дескрипторов процессов
3. Возобновление(рестарт)
   (a) Восстановление образа процесса с помощью машинного регистра
4. Изменение приоритета (Прибавление или вычитание числа)
5. Блокирование
6. Пробуждение (вывод из блокировки)
7. Запуск (выбор процесса на запуск)
   (a) Сортировка очереди готовых к выполнению процессов по приоритетам

Процессы называются параллельными, если существуют одновременно. Параллельные процессы называются асинхронными, если работают совершенно независимо друг от друга. Процесс-отец и процесс-сын НЕ асинхронные, если используется *wait*, иначе все-таки асинхронные.

У любого процесса есть хотя бы один поток управления (последовательность команд для выполнения конкретного действия) и один счетчик команд.

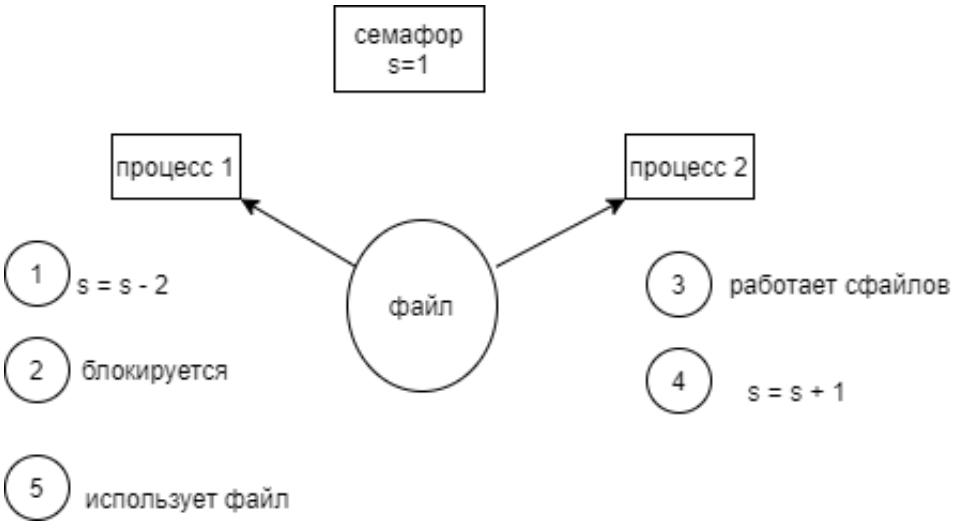
В некоторых современных ОС реализована возможность создания нескольких потоков управления в одном процессе. Такие потоки получили названия нитей (threads). У всех нитей процесса одно общее адресное пространство, но

у каждой личные счетчик команд, регистры и состояния. Информация о нитях записывается в таблицу в ОС. Каждая ее запись описывает один поток.

Волокно / fiber — поток, выполняемый в пространстве пользователя, а не ядра. Это свойственно только ОС Windows. Такой поток быстрее обычного.

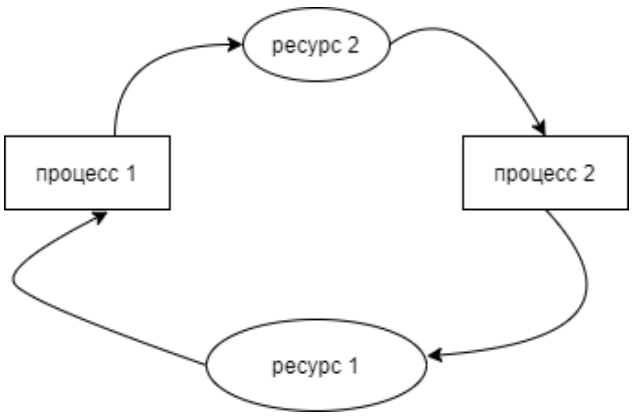
Задание — набор из одного или нескольких процессов, управляемых (выполняемых) как единое целое.

Семафор — системная переменная, которая является объектом ядра ОС и используется в основном для синхронизации процессов. Может принимать неотрицательные целочисленные значения.



Мьютекс / mutex — упрощенная версия семафора, которая может принимать только значения 0 и 1.

Процесс находится в состоянии тупика (deadlock, клинч), если он ожидает событие, которое никогда не произойдет:



1.2 Управление файловой системой

Файл — именованный набор данных, состоящий из записей, с заданной структурой, наложенной пользователем.

Символьный набор — это когда каждому символу противопоставляется численный код. Символьный набор кем-то предлагается и утверждается сообществом. Иногда это называется способом кодировки. Известные символьные наборы: ASCII, UNICODE, EBCDIC и др.

Файлы состоят из записей. Запись — одна строка (т.е. все символы, находящиеся между соседними переводами строк, в том числе пробельные), группа взаимосвязанных полей. Поле — группа последовательных байтов (или слов).

1.2.1 Дескриптор файла

- Символьное имя. Максимальная длина имени — 252(+4 под номер дескриптора);
- Размещение файла во внешней памяти (может содержаться в нескольких блоках, и все их надо указать в таблице);
- Тип файла (обычный, каталог, канал, устройство, гнездо...);
- Права доступа (описатель защиты);
- Размер файла;
- Время и дата последней модификации (создание — тоже модификация);
- Тип организации файла;
- Диспозиция файла (постоянный, временный (гнезда, семафоры, именные каналы), рабочий). Временные файлы живут, пока жива ОС, и удаляются после перезагрузки. Рабочие живут, пока жив процесс.

1.2.2 Организация файлов

Организация файлов подразумевает под собой способ расположения записей файлов во внешней памяти. Различают следующие виды организации файлов:

1. Последовательная — записи файла располагаются в физическом порядке.
2. Индексно-последовательная — записи располагаются в логической последовательности в соответствии со значениями ключей, содержащихся в каждой записи. Это характерно в основном для баз данных.
3. Прямая организация — записи могут быть раскиданы по всему диапазону

запоминающего устройства, доступ осуществляется по прямым указателям.

4. Библиотечная — файл, состоящий из последовательных подфайлов, например каталог, оглавление архива.

Файлами управляет ОС. Их структура, именование, защита, использование, доступ к ним относятся к той части ОС, которая называется файловой системой.

ОС содержит (предоставляет), как правило, следующие средства:

1. Средства доступа к файлам
2. Средства управления файлами (удаление, переименование, создание, копирование и т.д.)
3. Средства размещения файлов
4. Средства сохранения целостности файлов

Необходимая ОС информация для выполнения различных операций с файлом содержится в дескрипторе файла (блоке управления файлом). ОС ведет таблицу дескрипторов файлов, где представлена вся информация о файлах, находящихся на диске.

Дескриптор — структура данных, которая обычно хранится во внешней памяти и передается в ОП только во время открытия файла. Эти дескрипторы — индексные (а не пользовательские).

1.2.3 Реализация файловой системы

1. Неразрывные файлы

Файл занимает непрерывную цепочку блоков.

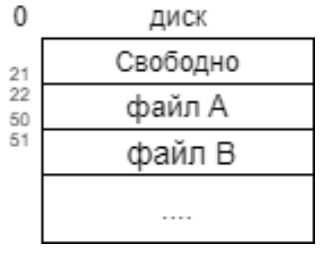
Плюс: быстрое чтение за одну операцию.

Минусы:

– Появление пустых окон при удалении файлов (проблема фрагментации). В эти окна могут не уместится новые файлы.

– Сложности с внесением изменений на диск

– Сложности с увеличением файлов



2. Список

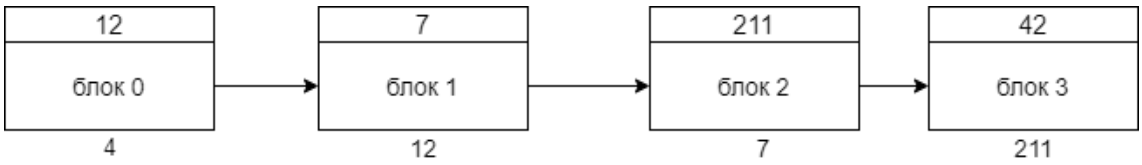
Файл хранится в виде однонаправленного связного списка блоков диска. В начале каждого блока находится указатель на следующий блок. В основной части — данные. В каталоге хранится указатель на следующий блок

Минусы:

– Долгий поиск

– Если один блок испортится, сломается все

– Долгий доступ



3. Список с индексацией

Указатели на следующий блок хранятся в отдельной таблице FAT (File Allocation Table), а не в блоке. Таблицу ведет ОС. Она должна быть загружена постоянно в память.

0	
1	
2	
3	
4	12
5	
…	…
12	7
…	…

Недостатки:

Хранение таблицы в оперативной памяти.

500Мб диска требует таблицы 2 Мб

Пример:

FAT

4. i-узел

Индексный дескриптор / узел.

С каждым файлом связана структура данных, i-узел (индексный дескриптор), содержащая все атрибуты файла. ОС ведет таблицу i-узлов, и каждая запись этой таблицы соответствует определенному файлу. Если сделать на файл символьную ссылку, то у него будет два i-узла (или больше). Каждый i-узел находится в оперативной памяти только тогда, когда соответствующий файл открыт. Если размер i-узла n байт, открыто k файлов, то таблица займет в памяти $n \cdot k$ байт. В Unix-системах $n = 64$ байта, и $n \cdot k$ значительно меньше, чем FAT-таблица (для списка с индексацией), и не зависит от объема диска.

5. New Technology File System (NTFS)

Файл состоит из атрибутов, каждый из которых представляется набором байтов. Большинство файлов имеет три атрибута: имя файла, идентификатор файла (64 бита), данные (если очень большой файл, может быть несколько потоков данных. Максимальная длина потока — 2^{64} байт).

Каждый том (дисковый раздел) диска (C:, D:, E:, etc.) содержит файлы, каталоги, битовые и байтовые массивы данных, таблицы с данными. Организован как линейная последовательность блоков (в Microsoft их называют кластерами). Обращение к блокам осуществляется по их смещению от начала тома.

Основной структурой данных в каждом томе является таблица MFT (Master File Table), состоящая из записей размером в 1 КБ. Каждая запись MFT описывает один файл или каталог. Максимальное количество записей в таблице MFT — 2^{48} записей. Каждая запись таблицы содержит атрибуты файла:

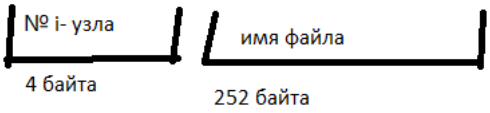
- Имя файла (256 байт)
- Идентификатор файла (64 байта)
- Данные: права доступа, имя тома, имя каталога, номера блоков диска, содержащие данные файла

Если файл очень большой, создается дополнительная запись, чтобы вместить весь список блоков файла.

Список свободных записей учитывается с помощью битовой карты. Ее размер 2^{48} бит. Первая запись MFT описывает саму MFT, т.е. расположение блоков таблицы. Номер первого блока файла MFT (в загрузочном блоке диска) — 0.

При открытии файла ОС оперирует полным именем файла. Если указывается неполное, то поиск происходит в текущем каталоге. Запись в каталоге содержит информацию, необходимую для нахождения блоков диска. Это может быть дисковый адрес всего файла для неразрывных файлов (номер 1-го блока), номер первого блока (обе схемы со списками), номер дескриптора i-узла.

Имя i-узла состоит из 256 байт:



1.3 Разделение оперативной памяти

ОС делит память на блоки размером, равным степеням 2 (512, 1024, 2048 байт) для учета свободных областей. Если блоки имеют одинаковый размер, их называют страницами, иначе — сегментами.

Учет свободных/занятых блоков осуществляется двумя способами.

1. **Битовая карта**

Набор битов: 1 — блок занят, 0 — блок свободен. Применяется при страничной памяти.

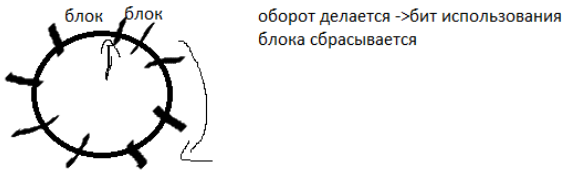
2. **Связный список блоков**

Основная в UNIX. Связный список представляет собой набор записей, каждая из которых отображает область свободной памяти. Запись состоит из: указателя на начало, длины области и указателя на следующую область.

1.3.1 Стратегии

1. **Стратегии выборки:**
   Ставят своей целью определять, *когда* следует втолкнуть очередной блок данных в оперативную память.
   - Выбор по требованию
   - Упреждающая выборка (никаких требований нет, процесс записан в память)
2. **Стратегии размещения**
   Ставят своей целью определять, *куда* следует втолкнуть очередной блок данных в оперативную память.
   - учитывая битовую карту
   - учитывая связный список свободных блоков
3. **Стратегии замещения**
   Ставят своей целью определять, *какой блок данных следует вытолкнуть* из оперативной памяти, чтобы освободить место для нового.
   - Самый простой алгоритм: вытолкнуть блок, к которому обращение производилось раньше остальных
   - FIFO — первый вариант: надо следить за порядком загрузки. Недостаток — можно вытолкнуть блок,который используется в данный момент

- FIFO — второй вариант. Перед удалением проверяет, не используется ли блок в данный момент
- LRU (Least Recently Used) — удаляет блок, использовавшийся меньше всего
- Aging (алгоритм старения)
- NRU (Not Recently Used) — удаляет не использовавшийся в последнее время блок
- Часы ("стрелка часов": при повторном проходе стрелки, если блок используется, то он не убирается из RAM без учета частоты использования. В MacOS 2 такие "стрелки")



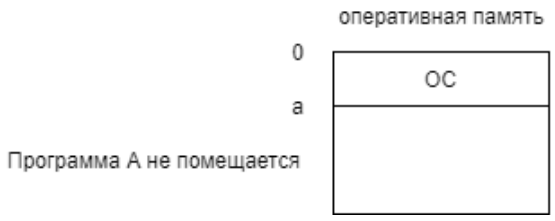
1.3.2 Концепции

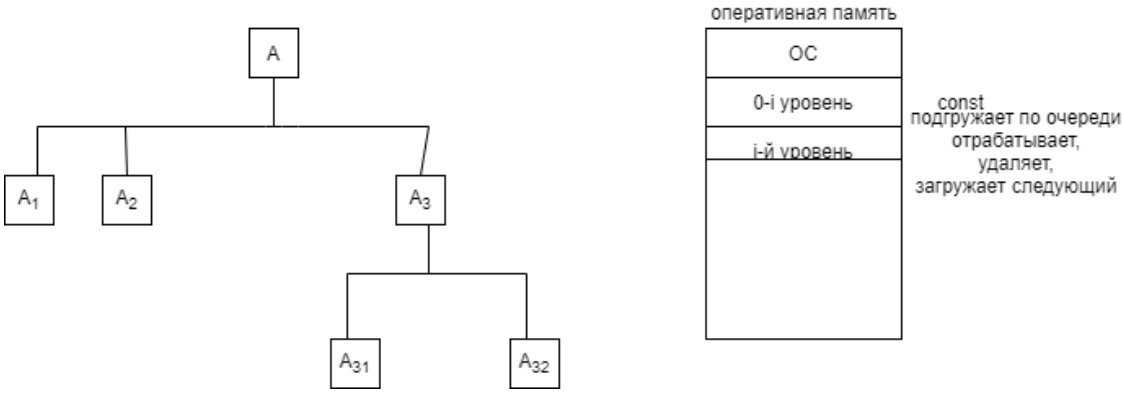
В современных ОС используются 2 концепции распределения памяти.

1. **Связная** — программа занимает один сплошной блок памяти.
   + не тратит время на поиск
   – не очень эффективно использует память
2. **Несвязная** — программа блоками разбросана по всей RAM.
   + память используется эффективно
   – дополнительные расходы на ведение учета, где и какой блок находится

Оверлейные перекрытия.

Если очередная часть программы не помещается, то необходимо разбить ее на еще более мелкие блоки. Это делает компоновщик. Программист же указывает компоновщику, в какой последовательности нужно запускать части программы.

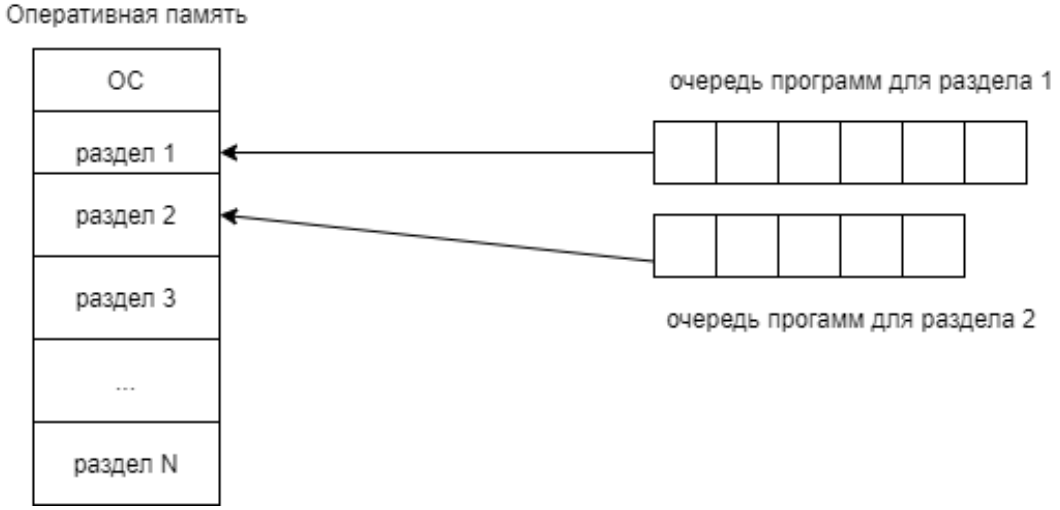




1.3.3 Подходы

1. Организация связной памяти с **разделами фиксированного размера**

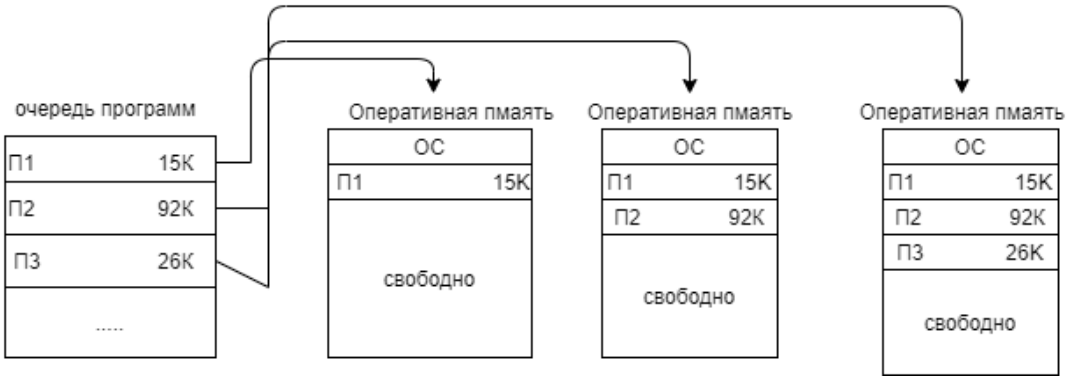
Вся RAM делится на разделы фиксированного размера. Для каждого такого раздела формируется очередь из программ, если нет перемещающихся загрузчика/компилятора/ассемблера. Если они есть, то очередь единая. Каждая программа может размещаться в тот раздел, размер которого это допускает.



2. Организация связной памяти с **разделами переменного размера**

Каждому заданию выделяется столько памяти, сколько оно требует.

Если освободился участок памяти, то новая программа может в него не влезть. Если таких "окон" много, то будут простои, следовательно, память будет использоваться неэффективно (фрагментация памяти).

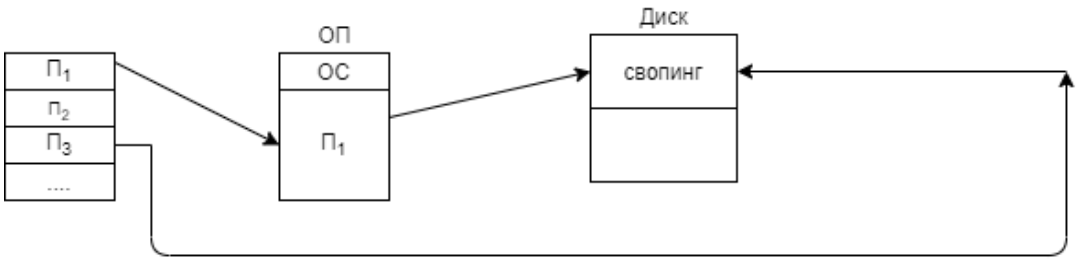


Если П4 не может влезть, берется следующая из очереди и т.д. После отработки программ появляются свободные окошки(фрагментация памяти) → в какой-то момент времени система приостановится → происходит сжатие (дефрагментация)/

Минус такого подхода в том, что подобная система не может быть реализована в режиме реального времени.

3. Свопинг / swapping (также для несвязного)

Не требует, чтобы программа постоянно находилась в оперативной памяти. После отработки своего кванта времени программа выгружаеся на диск, и на ее место загружается первая программа из очереди. На диске должен быть *буфер*.



со временем все засвопится

Программа может неограниченное количество раз перекачиваться во внешнюю память и обратно.

4. Виртуальная память (аналогично свопингу)(также для несвязного)

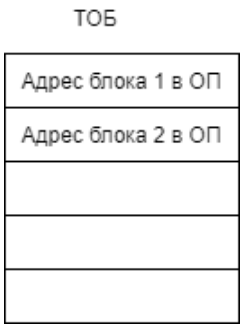
Используются не реальные, а виртуальные адреса памяти, за счет чего расширяется адресное пространство процесса.

Во время выполнения программы ОС должна преобразовывать ее виртуальные адреса в реальные. Чтобы сохранить объем информации отображения,

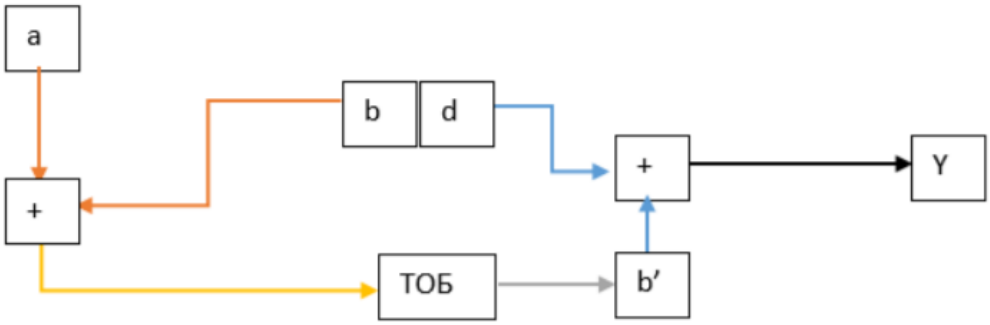
элементы информации группируются в блоки, и ОС следит за тем, где в оперативной памяти размещаются блоки виртуальной памяти.

Виртуальный адрес v представляется в виде упорядоченной пары $(b,\ d)$, где b — номер блока виртуальной памяти, d — смещение относительно начального адреса блока виртуальной памяти.

Каждый процесс имеет таблицу отображения блоков, которую ОС ведет в реальной оперативной памяти (таблица страниц).



Реальный адрес a этой таблицы в оперативной памяти загружается в специальный регистр ЦП. Номер блока b суммируется с базовым адресом a таблицы отображения блоков, образуя реальный адрес b' строки таблицы. Затем d и b' суммируются, образуя реальный адрес памяти.



Все 4 подхода могут быть применены для связной концепции, и только два — для несвязной.

1.4 Управление устройствами ввода-вывода

Это одна из важнейших функций ядра ОС.

Ядро ОС должно:

1. Перехватывать прерывания
2. Обрабатывать ошибки
3. Обеспечивать интерфейс с остальной частью ОС
4. Давать устройствам ввода-вывода команды

1.4.1 Типы УВВ

1. **Блочные**

Позволяют манипулировать блоками.

Устройства, хранящие информацию в виде адресуемых блоков фиксированного размера: от 512 КБ (= сектор диска) до 32768 КБ (= цилиндр диска). Каждый блок может быть прочитан/записан вне зависимости от остальных блоков. Важное свойство блочного устройства: каждый блок может быть прочитан независимо от остальных. Пример: накопитель на магнитном диске.

2. **Байтовые (символьные)**

   Принимают/предоставляют поток символов без какой-либо структуры. Не являются адресуемыми. Примеры: сетевая интерфейсная плата, принтер, мышь.

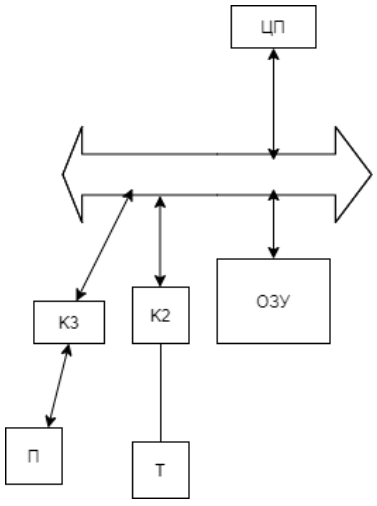
Программа пользователя общается с абстрактными устройствами. Зависимую от устройств часть поддерживает ядро ОС (драйверы устройств).

1.4.2 Части УВВ

Устройства ввода-вывода состоят из двух частей:

1. **Механическая** — это само устройство.
2. **Электронная = контроллер**.

   Принимает форму печатной платы, которая сопряжена с системной шиной компьютера или вставляется в слот расширения объединительной платы.



К2, К3 — контроллеры, П и Т — некие УВВ, например, принтер и телетайп

Работа контроллера — *преобразование* последовательного потока битов в байтовую последовательность и *коррекция ошибок*.

У каждого контроллера есть несколько регистров, с помощью которых с ним может общаться ЦП.

Регистры формируют свое адресное пространство.

В дополнение к регистрам часто используются прерывания, с помощью которых контроллер может сообщить процессору, готовы ли к работе его регистры.

Interrupt ReQuest line (IRQ) — один из видов контроллера.

Прерывание является электрическим сигналом.

Линия запроса аппаратного прерывания является одним из входов контроллера. Количество таких линий ограничено. Каждая из линий аппратного прерываниий связывается с вектором прерываний (указывает на адрес программ прерывания).

Обмен информацией с УВВ с точки зрения ОС осуществляется так: ОС записывает команду в регистры контроллера (обычно в регистры адреса и данных). Передав команду контроллеру, процессор может продолжить работу, не отвлекаясь на устройство. Затем, после выполения устройством команды, контроллер инициирует прерывание, чтобы привлечь внимание ОС для проверки результата.

Далее ОС проверяет результат и переносит его в оперативную память.

1.4.3 ПО системы управления УВВ

ПО системы управления вводом-выводом для всех ОС разрабатывается, исходя из принципов:

1. Независимость от УВВ
2. Единообразие именования устройств
3. Многослойность

Уровни такого ПО:

1. **Обработчик прерываний** — нижний уровень.
   Программа, содержащая последовательность команд обработки прерывания для какого-либо УВВ. (далее — подробнее)

2. **Драйвер УВВ**
   1) программа, которая позволяет осуществлять обмен информацией между УВВ и оперативной памятью.
   2) программа управления устройством, воспринимающая и выполняющая запросы от программ верхних уровней. Например, драйверы диска при чтении информации из файла должны выяснить, где находится запрошенный блок данных, проверить, работает ли привод диска, спозиционировать головки диска на нужную дорожку нужного сектора, считать информацию и занести ее в регистр контроллера.

3. **Не зависимые от аппаратуры программные модули ОС**
   Это системные вызовы, которые реализуют следующие функции:
   - Единообразный интерфейс
   - Буферизация
   - Обеспечение аппаратно независимого размера блока
   - Захват и освобождение монопольных УВВ
   - Обработка ошибок
   - Обработка прерывания
4. **Пользовательские программы**

1.4.4 Типы прерываний

(Для каждой вычислительной системы вариативны)

1. По вызову супервизора. Инициатор — сам процесс
2. От УВВ (инициируются аппаратурой ввода/вывода)
3. Внешние. Инициатор — пользователь ПК / истечение кванта времени, заданного на таймере прерываний / прием прерывания от другого процессора.
4. Программные (по ошибке программы)
5. По ошибке компьютера (инциируются аппаратными прерываниями)
6. Прерывание по рестарту (пользователь или от другого процесса)

Для обработки каждого из типов прерываний в составе ОС предусмотрены программы, которые называются обработчиками прерываний. Когда происходит прерывание, ОС запоминает состояние прерванного процесса, запрещает все другие прерывания, приступает к обработке прерывания. После нее процесс возвращается в то состояние, в котором он был прерван, и все типы прерываний снова разрешаются.

2 UNIX-подобные системы

2.1 Технические характеристики UNIX-подобных систем

Все работающие программы в Unix-подобных системах представлены множеством конкурирующих процессов/потоков. Unix-системы реализованы с разделением времени (несколько потоков могут идти одновременно). В unix-подобных

системах реализуется единый интерфейс для обработки информации. Все ресурсы ассоциированы с понятием файла (внешнее устройство,процесс, память — это тоже файлы).

К **достоинствам** ОС UNIX относятся: открытость исходного кода, инструментальность (насыщенность программными продуктами), мобильность (возможность перенести на другие вычислительные средства с минимальными затратами), сетевая направленность (текущие используемые сетевые протоколы изначально разрабатывались для unix-систем), унифицированность (все ресурсы системы унифицированы под понятие файла).

К **недостаткам** следует отнести неподдержку режима реального времени "из коробки", низкую эффективность при решении однотипных задач, слабую устройчивость к аппаратным сбоям.

Структура ОС UNIX:
— Ядро
— Системные программы

2.2 Ядро ОС

2.2.1 Функции ядра ОС:

1. Диспетчеризация процессов, распределение ресурсов между ними
2. Обработка прерываний
3. Управление УВВ
4. Распределение внешней и оперативной памяти
5. Выполнение системных вызовов

К секции управляющих структур относятся:

1. Таблица процессов
2. Таблица файлов
3. Таблица связей драйверов
4. Таблица очередей сообщений
   etc.

2.2.2 Программная секция ядра

Программная секция ядра состоит из 2-х частей, являющихся целыми наборами программ:

1. **Диспетчер процессов**

К нему относится набор программ, который определяет последовательность выполнения процессов, распределяет ресурсы системы (в том числе память), обрабатывает системные вызовы и сигналы.

1. **Диспетчер УВВ**

Контролирует и обеспечивает передачу информации между оперативной памятью и УВВ.

В него входят драйверы символьных устройств, драйверы сетевых устройств, драйверы дисковых устройств, дисциплину линии, буферный, страничный кэши, маршрутизаторы, сетевые протоколы, etc.

2.3 Состояния процессора

В каждый момент времени процессор может находиться в одном из трех состояний:

- Система(выполняются команды ядра ОС т.е СВ)
- Процесс (выполняются команды пользовательского процесса)
- Ожидание (наступает, когда в системе нет процессов/потоков, готовых к выполнению)

Переход "процесс" $\rightarrow$ "система" может быть вызван прерыванием по таймеру или выполнением системного вызова.

Переход "система" $\rightarrow$ "процесс" может быть вызван окончанием обработки прерываний или окончание выполнения системного вызова.

Переход "ожидание" $\rightarrow$ "процесс" невозможен.

Переход "процесс" $\rightarrow$ "ожидание" возможен, когда в системе нет потоков вообще или потоков, готовых к выполнению.

2.4 Процесс в UNIX

2.4.1 Структура

Структура процесса в момент его присутствия в оперативной памяти представляется следующими сегментами:

1. Процедурный сегмент: машинные команды, константы
2. Сегмент данных: проинициированные данные, которые могут изменяться в процессе работы

3. Динамический сегмент: данные, не инициированные при компиляции
4. Контекст процесса — таблица, в которой содержатся некоторые данные о процессе

Контекст процесса — структура данных, таблица, содержащая информацию о процессе:

1. Машинные регистры (для рестарта)
2. Таблицу пользовательских дескрипторов файлов процесса
3. Информацию о состоянии текущего системного вызова (параметры и результаты)
4. Учетную информацию (указатель на таблицу, учитывающую использованное процессом процессорное время)
5. Стек ядра (фиксированный стек для работы процесса в режиме ядра)

Доступ к нему имеет только ядро ОС (можно посмотреть через системные вызовы). Он не относится к адресному пространству процесса, однако подвергается своппингу совместно с процедурным сегментом и сегментом данных, которые образуют core-файл. Динамический сегмент своппингу не подвергается.

Контекст процесса не является структурой ядра, он относится к процессу.

2.4.2 Дескриптор процесса

Кроме контекста процесса ядро поддерживает таблицу процессов, которая является резидентной (не выгружается из памяти). Каждая запись этой таблицы называется дескриптором процесса и содержит информацию об одном из процессов:

1. PID — идентификатор процесса
2. Целое число — текущее состояние процесса
3. PPID — идентификатор родительского процесса
4. UID — идентификатор владельца
5. GID — идентификатор группы
6. Время до истечения интервала будильника
7. События, которые ожидает процесс

   6 и 7 нужно,чтобы выводить процесс из состояния блокировки.

8. Сигнальная маска — массив сигнальных флагов. Если флаг 1, то ОС пропускает этот сигнал процессу
9. Приоритет CPU
10. Приоритет NICE
11. Приоритет PRI. Если ОС поддерживает потоки, то приоритеты уходят к потокам, а в дескипторе процесса этого пункта не будет.
12. Счетчик использования процессорного времени
13. Указатель на процедурный сегмент. Если используется страничная организация памяти, то указатель на таблицу страниц процедурного сегмента
14. Указатель на сегмент данных. Если используется страничная организация памяти, то указатель на таблицу страниц сегмента данных
15. Указатель на динамический сегмент. Если страничная организация памяти, то указатель на таблицу страниц динамического сегмента
16. Указатели внешней памяти

2.4.3 Создание процесса

Все процессы в ОС UNIX, за исключением процесса с PID=0 (swapper), создаются с помощью *fork*. Например, `./a.exe` — запуск *fork*. Когда выполняется *fork*, он ищет свободную запись в таблице процессов, в которую можно записать данные о новом сыновьем процессе. Туда копируется информация с родительского процесса. Затем *fork* выделяет память для сегментов данных и динамического сегмента процесса-сына, куда копируются соответствующие сегменты родительского процесса. Контекст процесса также создается для процесса-сына.

fork присваивает процессу-сыну идентификатор. Затем настраивается карта памяти сына и регистры. Также сыну предоставляется доступ ко всем файлам отца посредством копирования таблицы пользовательских дескрипторов файлов из контекста отца.

Карта памяти сына настраивается так: сыну выделяются новые таблицы страниц, но эти таблицы указывает на страницы отца, в правах доступа у которых стоит "доступно только для чтения". Когда сын пытается записать что-то в страницу, происходит прерывание. При этом ядро выделяет сыну новую копию страницы, к которой у сына есть досутп на запись. Таким образом, копируются только те данные, в которые процесс-сын запишет новые данные. Страницы с процедурным сегментом не копируются.

2.4.4 Планирование процессов

Определение последовательности выполнения процессов предусматривает 2 алгоритма. Оба реализованы в процессах с PID = 0 (swapper).

1. **Низкоуровневый алгоритм**

Выбирает следующий процесс из набора готовых к работе процессов в памяти, которые следует запустить.

Используются несколько очередей. С каждой очередью связан диапазон непересекающихся значений приоритетов. Самый высокий приоритет у самых больших по модулю отрицательных значений.

Когда запущен низкоуровневый алгоритм, он ищет очередь с наивысшим приоритетом, пока не находит очередь, в которой есть хотя бы один процесс, и после этого из очереди выбирает и запускает первый процесс в ней (все процессы в очередях готовые). После отработки своего кванта времени или в случае прерывания он помещается в конец своей очереди. Но если он был заблокирован, то в очередь не возвращается.

Раз в секунду приоритет каждого процесса пересчитывается по формуле:

$$\mathrm{PRI} = \mu\mathrm{CPU} + \eta\mathrm{NICE} + k\mathrm{BASE}$$

И на основе нового приоритета прикрепляется к соответствующей очереди.

CPU — время выполнения процесса в тиках таймера начиная с создания (при каждом тике таймера счетчик использования процесса увеличивается на 1). Значение этого счетчика добавляется к приоритету процесса, тем самым увеличивая его.

NICE — пользовательская составляющая приоритета процесса, является числом в пределах от -20 до 20. Может быть изменена с помощью системных вызовов от 0 до 20. Изначально NICE=0.

BASE — отрицательное число, определяющееся событием, которое ожидал процесс. Отрицательные значения для дискового/терминального ввода-вывода, для срабатывания таймера, выполнения системного вызова жестко прошиты в ОС и могут быть изменены только перекомпиляцией всей системы. Добавляется к приоритету поцесса после выхода из состояния блокировки. Блокировка может произойти: из-за соответствующего ввода-вывода, прерывания, системного вызова или включения ожидания.

2. **Высокоуровневый**

Перемещает процессы из памяти на диск и обратно (что называется Свопинг / swapping).

2.5 Управление внешней памятью

То есть распределение внешней памяти на файлы, управление файловой системой. С понятием файла связаны все ресурсы в UNIX-системах.

2.5.1 Типы файлов

Тип файла можно посмотреть командой `ls -l` (первый символ в выводе).

1. Обычные
2. Каталоги (первый символ в выводе для ls — d)
3. FIFO (каналы) (p)
4. Специальные (b — блокоориентированные, c — байтоориентированные)
5. Гнезда (s)(т.е сокеты)

2.5.2 Характеристики файлов

По типам файлов:

1. **Обычные**

Содержат информацию, определенную пользователем или формирующуюся в результате работы прикладных/системных программ.

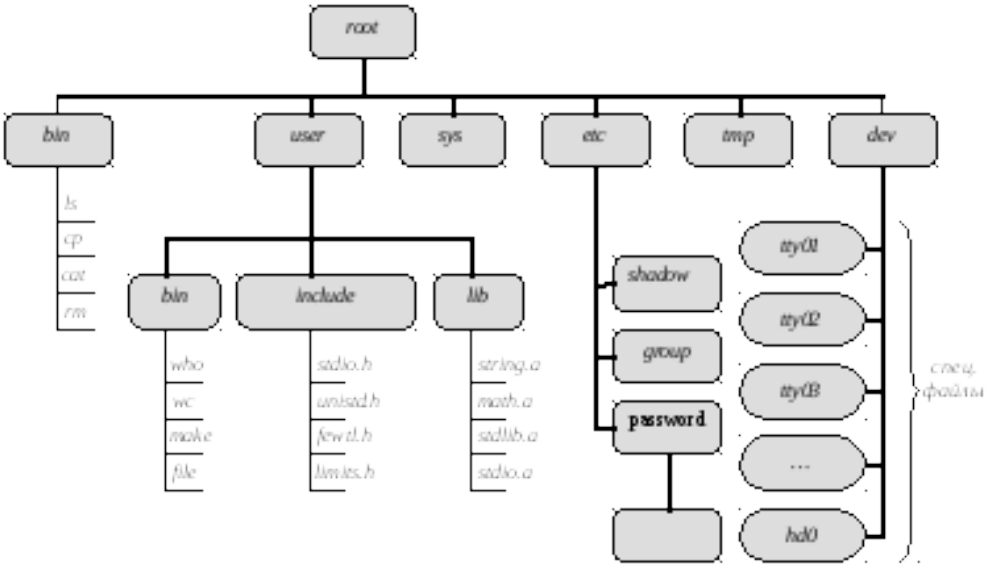
Нет никаких ограничений на хранимую информацию.

Различаются на *текстовые* и *бинарные*.

Могут быть удалены командой `rm` (удаляет только жесткую ссылку, но не файл) или `unlink`

2. **Каталоги**

Файл, который содержит информацию о других файлах в виде несортированного набора записей длиной в 256 б (4 б под номер I-узла, 252 б под имя).



Каталог считается пустым, если содержит только . (точка — ссылка на текущий каталог) и .. (две точки — ссылка на родительский каталог).

Символьная ссылка (мягкая) позволяет ссылаться на на каталог или файл в другом каталоге. Создает дополнительный идексный дескриптор, в котором в поле "тип файла" записано "символьная ссылка" и который имеет дополнительное поле, в котором указывается, на какой именно файл ссылка. Во всем остальном индексный дескриптор символьной ссылки идентичен дескриптору оригинального файла.

Имена файлов хранятся не в дескрипторе, а в каталоге.

/usr/bin/wc - полное имя файла(путь или жесткая ссылка на файл (файл может содержать несколько жестких ссылок))

/ — корневой каталог

Если имя файла неполное,то ОС будет искать в текущем каталоге.

3. **FIFO**

Предназначены для создания временного буфера (канала), обеспечивающего взаимодействие нескольких процессов посредством чтения или записи в буфер. Буфер, связанный с FIFO-файлом создается, когда один из процессов открывает этот файл для чтения или записи. Буфер удаляется, когда все процессы, связанные с FIFO-файлом, закрывают все свои ссылки на него.

mkfifo() — функция создания именованного (временного) канала.

pipe() — функция создания неименованного канала.

4. **Специальные**

Предназначены для организации взаимодействия с УВВ. С каждым внешним устройством связан свой специальный файл.

Бывают байториентированными и блокориентированными. С ними можно работать как с обычными файлами.

Обычный файл создается (пример Истратова):

fd=open("/dev/tty",1);

write(fd,...);

Создаются с помощью команды mknod c(байт), b(блок)

Пример создания специального файла:

`mknod /dev/crk c 5 12`, где

`mknod` — команда создания специального файла,

c — ориентация файла,

5 (старший номер)— номер типа устройства, а также номер строки в таблице драйверов, которую ведет ОС,

12 (младший номер) — номер устройства, который передается в драйвер устройства как 1-й внешний аргумент.

2.5.3 Индексный дескриптор файла

У файла может быть несколько дескрипторов. Мягкая ссылка — несколько дескрипторов, жесткая — один.

Индексный дескриптор содержит следующие атрибуты файла:

1. Права доступа
2. Количество жестких ссылок
3. UID — идентификатор пользователя
4. GID — идентификатор группы
5. Размер файла (в байтах)
6. Номер индексного дескриптора (их может быть несколько)
7. IFS — идентификатор файловой системы, в которой находится файл
8. Время последнего доступа (все время указывается в секундах с 1970г.)
9. Время последней модификации
10. Время последнего изменения прав доступа
11. Тип файла
12. Диспозиция файла (временный / постоянный / рабочий)

13. Тип организации файла
14. Физический адрес (адрес первого блока)

Каждый дескриптор имеет размер 64 б.

Для каждой файловой системы (UNIX может поддерживать неограниченное их количество) формируется свой идентификатор и создает таблица индексных дескрипторов, в которой хранится информация обо всех файлах. Каждая запись в таблице идексных дескрипторов содержит все перечисленные атрибуты файла и определяется по номеру индексного дескриптора и идентификатора файловой системы (идентификатор пприсваивается с помощью команды *mount*). СВ *stat()* позволяет получить все атрибуты.

Всякий раз, когда создается новый файл, ядро ОС UNIX создает новую запись в таблице индексных дескрипторов для сохранения информации о нем. Кроме того, ядро добавляет его имя и номер дескриптора в соответствующий каталог. Таблицы индексных дескрипторов содержатся в соответствующих файловых системах на диске, но ядро ОС также ведет их копии в оперативной памяти.

2.5.4 Поддержка файлов ядром ОС UNIX

Ядро ведет таблицу пользовательских дескрипторов открытых файлов процесса. Ведется для каждого процесса в его контексте.

	Имя файла	№ ТФ
0		
1		
2		
3	a.txt	4

Ядро также ведет таблицу файлов (ТФ), в которой отслеживаются все открытые в системе файлы. Ведется для всех процессов.

	ИД	Указ. тек. поз. в файле	Режим откр. файла	Счетчик ссылок
0				
1				
2				
3				
4	241	0	0	1

Каждый вновь созданный процесс содержит таблицу пользовательских дескрипторов открытых файлов, в которой автоматически система открывает файлы стандартных ввода, вывода, протокола (т.е. файлы с дескрипторами №№0, 1, 2), и в которой автоматически регистрируются файлы, открытые процессом.

Когда процесс вызывает СВ *open()* или *creat()*, чтобы открыть или создать файл для чтения или записи, ядро преобразует имя файла в индексный дескриптор файла. Если индексный дескриптор файла доступен процессу, ядро ищет в таблице пользовательских дескрипторов открытых файлов процесса первую незадействованную позицию (обычно это 3, т.к. 0, 1, 2 заняты стандартными файлами). Номер этой позиции будет возвращен процессу=== как пользовательский дескриптор открытого файла.

Далее ядро просматривает таблицу файлов и ищет в ней первую незадействованную позицию. В таблице указывается ссылка на эту запись ТИДФ. Заносится в ТФ информация об индексном дескрипторе, указателе текущей позиции указателя в файле, режиме открытия, счетчике ссылок (указывает, сколько дескрипторов файла из процесса обращается к данной записи ТФ).

Ядро будет использовать пользовательский дескриптор открытого файла для поиска записи в таблице файлов, соответствующей данному открытому файлу. Ядро проверяет эту запись, чтобы убедиться, что файл открыт в соответствии с режимом доступа. Далее ядро использует указатель из записи ТФ для доступа к индексному дескриптору файла. Кроме того, по записи в ТФ ядро определяет, с какого байта должна осуществляться операция чтения/записи. Ядро также проверяет тип файла по индексному дескриптору и вызывает соответствующий драйвер, чтобы начать обмен данными.

2.5.5 Логическая и физическая организация файловой системы

Логическая и физическая структуры файлов в UNIX не совпадают. Логически файл представляется непрерывной цепочкой блоков, а физически — списком блоков, которые могут быть разбросаны по всему дисковому пространству.

Достоинство: размер файла не ограничен.

Недостаток: нужно где-то хранить адреса блоков и их последовательность.

В 0 блок (*блок начальной загрузки*) помещается адрес программы, которая запускает операционную систему.

1 блок (*суперблок*) содержит заголовок файловой системы. В заголовке находится информация о размере ФС (fsize — сколько блоков диска занимает файловая система), количество индексных дескрипторов (isize, синоним — количество i-узлов), указатель на связный список свободных блоков. *fsize* > *isize*.

Любые индексные дескрипторы занимают 64 байта

Количество блоков $= 2 + \frac{isize}{(8)(16,24,32)}$

Блоки от 2 до 2 + isize содержат индексные дескрипторы файлов.

Последующие блоки — это блоки с данными, свободные блоки (на которые есть указатель в суперблоке), косвенные блоки.

Косвенные блоки — ссылки на блоки на диске, которые занимает файл и которые не поместились в индексный дескриптор.

0 блок	суперблок	индексные дескрипторы	блоки с данными,косвенные данные,свободные блоки	супер блок			блоки с данными,косвенные данные,свободные блоки

Для эффективной работы с файлами диск разбивается на группы блоков(цилиндров)(по сути совокупность файловых систем,индексных дескрипторов и блоков с данными), у каждой из которых собственые суперблок, индексные дескрипторы и блоки с данными. Это сделано, чтобы хранить индексные дескрипторы и блоки с данными ближе друг к другу (снижает время, затрачиваемое жестким диском на перемещение блока головок).

Для обеспечения быстрого доступа к файлам некоторые части файловой системы должны постоянно находиться в оперативной памяти. Прежде всего, это суперблоки и таблицы индексных дескрипторов + таблица файлов

2.6 Управление оперативной памятью

2.6.1 Управление памятью на основе свопинга

Это подход, при котором программа многократно перекачивается на диск и обратно во время выполнения. Большинство UNIX-систем использует именно этот подход.

Перемещением данных между оперативной памятью и диском осуществляет верхний уровень планировщика — свопер.

Выгрузка данных из памяти на диск инициируется, когда у ядра ОС кончается свободная память из-за одного из событий:

- Вызов *fork()*, появление нового процесса. Это требует дополнительной памяти, а ее может не хватать.
- Вызов *brk()* — требует память для расширения сегмента данных
- Вызов *alloc()* — динамически пытается расширить память
- У ядра наступает потребность возобновить процесс, долго находящийся на диске. Swapper удаляет из оперативной памяти (перемещает на диск) один из процессов. Выбирая жертву среди таких процессов, swapper сначала рассматривает процессы, находящиеся в состоянии блокировки. Если такие процессы нашлись, выбирается процесс с наивысшим значением суммы приоритета и временем пребывания в памяти (PRI+TIME). Если блокированных процессов нет, то по тому же критерию выбирается процесс в состоянии готовноти

Каждые несколько секунд свопер исследует список процессов, выгруженных на диск с целью определить, не готов ли какой-то из них к работе. Из списка процессов в состоянии "готов" выбирается процесс, дольше всех находящийся на диске. Затем свопер проверяет, будет ли это *легкий* свопинг или *тяжелый*. При легком свопинге не требуется дополнительного высвобождения памяти. Тяжелый свопинг — свопинг, в котором для загрузки в память выгруженного на диск процесса из памяти требуется удалить один или несколько других процессов.

Эта процедура выполняется до тех пор, пока не выполнится одно из условий:

- на диске нет процессов, готовых к работе
- в памяти нет места для новых процессов (т.е. из памяти ничего нельзя удалять)

Чтобы не терять большую часть производительности системы на свопинг, ни один процесс не выгружается на диск, если он находится в [оперативной] памяти меньше двух секунд.

Свободное место в памяти и на диске (в оперативной и внешней памяти) учитывается при помощи связного списка свободных блоков.

2.6.2 Управление памятью на основе свопинга со страничной подкачкой

Другое название — **виртуальная память**.

Для предоставления возможности работы с программами больших размеров (даже превосходящими по объему размер оперативной памяти) в UNIX-системах к свопингу добавлена страничная подкачка.

При таком подходе процессу не нужно целиком находиться в оперативной памяти. Требуется только контекст процесса и таблица страниц. Если они загружены, процесс считается находящимся в памяти и может быть запущен планировщиком. Страницы с процедурным сегментом, сегментом данных и динамическим сегментом загружаются в память динамически по мере обращения к ним.

Страничная подкачка реализуется частично свопером (процесс с PID = 0), частично "страничным демоном" (процесс с PID = 2).

"Страничный демон" в основном спит, периодически запускается и смотрит, есть ли для него работа. Если он обнаруживает, что количество страниц в списке свободных страниц мало, страничный демон инициирует освобождение страниц, т.е. выгрузку страниц из памяти на диск.

Память условно делится на три части:

1. Ядро ОС (имеются в виду в основном программы)
2. Карта памяти (в том числе страницы ядра (данные))
3. Страницы (как пустые, так и занятые)

Карта памяти — таблица, содержащая информацию о содержимом каждой страницы памяти. Каждая ее запись имеет один и тот же размер.

Запись карты памяти содержит:

1. Номер следующей записи
2. Номер предыдущей записи. Если это и предыдущее поля свободны, то все остальные заняты. Если заняты, то остальные поля свободны. Они используются только тогда, когда страницы находятся в списке свободных страниц. В этом случае свободные страницы сшиваются в двусвязный список, и по этому списку ОС определяет, сколько свободно места в оперативной памяти.

   Следующие поля используются только тогда, когда страница содержит какую-то информацию:

3. Номер блока диска
4. Номер блока устройства
5. Хэш-код блока
6. PID процесса
7. Вид сегмента (сегмент данных, контекст процесса, etc.)
8. Смещение сегмента
9. Бит использования. Флаг, который используется алгоритмом замещения (страничной подкачки).

У каждой страницы, находящейся в оперативной памяти есть фиксированное место хранения на диске, в которое помещается, когда выгружается из памяти. Это место характеризуют пункты 3—5. Поля 6—8 характеризуют информацию, которая хранится в странице.

При запуске процесс может вызвать страничное прерывание (это прерывание по вызову супервизора), если одной или нескольких его страниц не окажется в памяти. При страничном прерывании ОС берет первую страницу из списка свободных страниц, удаляет ее из списка и считывает в нее требуемую страницу. Запись в карте памяти меняется. Если список свободных страниц пуст, выполнение процесса приостанавливается до тех пор, пока "страничный демон" не освободит необходимое количество страниц.

Алгоритм замещения страниц

Задача алгоритма — следить, чтобы количество свободных страниц в оперативной памяти было достаточным для того, чтобы система работала.

Реализован в процессе с идентификатором PID = 2. Раз в 250 мс этот "страничный демон" просыпается, чтобы сравнить количество свободных страниц с системным параметром lostfree. Если количество свободных страниц меньше этого параметра или минимума, "страничный демон" начинает переносить страницы из памяти на диск, пока количество свободных страниц не станет равным lostfree или максимуму. После этого снова засыпает на 250 мс.

Используется модифицированный алгоритм "часов с двумя стрелками". Поддерживает два указателя на карту памяти. Одним указателем сбрасывает бит использования страницы, другим указателем через некоторое время проверяет, обращался ли кто-то за это время к блоку. Если нет, он удаляется из памяти.

Если ОС (точнее, процесс с PID = 0, свопер) обнаруживает, что частота подкачки страниц слишком высока, а количество свободных страниц все время

ниже параметра lostfree или максимума, то свопер начинает удалять из памяти один или несколько процессов. Сначала свопер проверяет, есть ли процессы, которые бездействовали в течение 20 и более секунд. Если такие процессы нашлись, из них выбирается процесс, бездействовавший в течение максимального срока, и выгружается на диск. Если таких процессов нет, свопер анализирует четыре самых больших процесса (у которых больше всего страниц памяти), из которых выбирает тот, что находится в памяти дольше всех, и выгружает его на диск. Так до тех пор, пока не освободится достаточное количество памяти.

Каждые несколько секунд (по умолчанию две) свопер проверяет, есть ли на диске процессы в состоянии "готов" и выбирает из них процесс с наивысшим приоритетом. Загрузка выбранного процесса производится только при условии наличия достаточного количества свободных страниц, чтобы для него нашлись свободные страничные блоки, когда случится неизбежное страничное прерывание.

Свопер загружает в память только контекст процесса и таблицы страниц. Сами страницы процедурного, динамического сегментов и сегмента данных подгружаются с помощью обычной страничной подкачки. Для каждого сегмента каждого активного процесса есть место на диске, где он располагается. Сегмент данных и динамический сохраняются в области свопинга (это буфер на диске) в виде временной копии, а процедурный сегмент в самом двоичном файле.

2.7 Система управления вводом-выводом в ОС UNIX

2.7.1 Логическая и физическая организация СУВВ

С точки зрения логической организации, все УВВ представляются в виде файлов, все файлы рассматриваются как последовательный набор байтов, к которым возможно как последовательное, так и прямое обращение.

Чтобы работать с таким файлом, его нужно открыть с помощью СВ *open()*, с помощью *lseek()* возможен прямой доступ к файлу и *close()* позволяет закрыть этот файл и освободиться от устройства.

Физическая организация опирается на наличие программных средств, работающих на уровне ядра ОС — драйверов. Драйвер УВВ — программа обмена данными между конкретным УВВ и оперативной памятью компьютера, учитывающая все особенности данного УВВ. Ядро ОС устанавливает однозначную связь между драйверами и соответствующими специальными файлами.

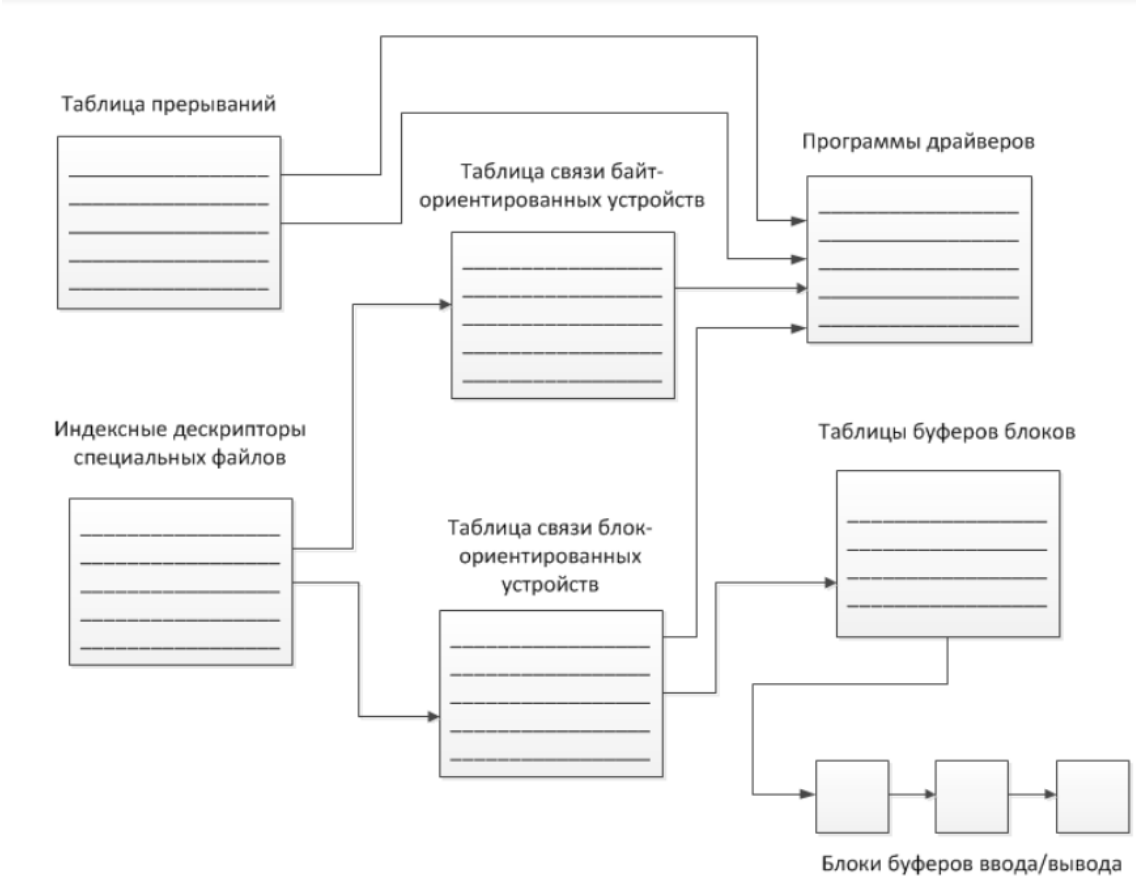
Драйвер УВВ и система буферизации ядра ОС поддерживают два вида интерфейсов — байтоориентированный и блокоориентированный. **Блокоориентированный** интерфейс позволяет осуществлять обмен блоками между устройством и оперативной памятью, а также значительно повысить эффективность СУВВ за счет организации кэш-памяти. Кэш-память — системные буфера ввода-вывода размером с блок, где оседают те блоки данных, обращение к которым производится наиболее часто. Когда возникает запрос на считывание какого-то блока данных с УВВ, то сначала просматривается кэш-память, и только если необходимый блок там отсутствует, производится обращение через драйверы к внешнему устройству.

Байтоориентированный интерфейс предназначен в основном для медленных периферийных устройств. Обмен данными осуществляется посимвольно через буфер символов в ядре системы. При обмене данными без буферизации байтоориентированный интерфейс называется прозрачным блокоориентированным.

Для большей части внешних устройств допускается использование обоих видов интерфейсов.

СУВВ — часть ядра, выполняющая функции диспетчера УВВ. Программная часть диспетчера включает управляющую компоненту, систему буферизации и набор драйверов. В качестве структурных устройств (табличных компонент) используются:

- индексные дескрипторы специальных файлов
- таблица связи между драйверами *байтоориентированных* устройств и ядром системы
- таблица связи между драйверами *блокоориентированых* устройств и ядром системы
- таблица буферов блоков
- таблица векторов прерываний



Каждый индексный дескриптор связан с таблицами связи

Индексный дескриптор специальных файлов

Включает информацию:

1. Имя специального файла (4 байта, также не может быть жестких ссылок и альтернативных имен)
2. Тип устройства (блоко-/байтоориентированное)
3. Номер типа устройства (старший номер)
4. Номер устройства (младший номер)

Имя файла включено в индексный дескриптор, у него нет альтернативных имен => можно убрать счетчик жестких ссылок, можно убрать диспозицию файла, идентификатор файловой системы, остальное оставляем.

Каждый индексный дескриптор специального файла ссылается на соответствующую запись в таблицах связи. Каждая запись в этих таблицах относится к одному УВВ. Каждая запись адресуется к определенному драйверу. Таблица связи для блокоориентированных устройств также связана с таблицей буферов блоков, указывающей на буфера ввода-вывода и образующей кэш-память.

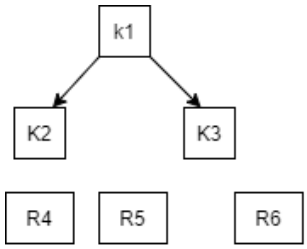
Программы драйверов предназначены для открытия и закрытия специальных файлов, чтения и записи данных, а также для управления режимами работы устройств.

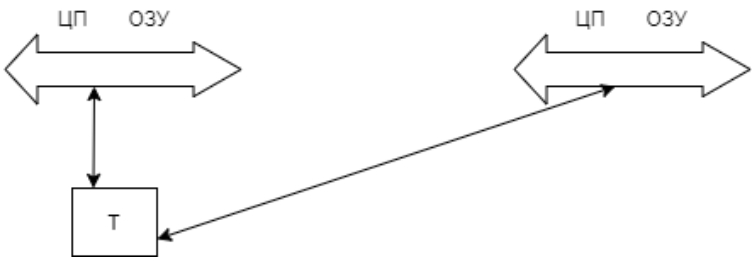
Таблица прерываний располагается в начальной области памяти по фиксированным адресам. При возникновении прерываний от УВВ управление передается программе-обработке прерываний драйвера, адрес которой выбирается из таблицы векторов прерываний.

Рассмотрим примеры операций чтения с блокоориентированного устройства.

1. Открыть файл, с которого собираемся читать:
   ```
   fd = open ("dev/bhk 0" , 0);
   ```
2. Считать файл:
   ```
   read (fd, buf, n);
   ```
3. *read()* инициирует обращение к спец. файлу и через индексный дескриптор специального файла определяет тип и номер устройства, с которого необходимо прочитать информацию. По адресу из таблицы связи управление передается необходимому драйверу. Программа драйвера проверяет, не располагается ли требуемый блок информации в кэш-памяти,если этот блок информации располагается в кэш памяти, драйвером эмулирует сигнал прерывания об окончании обмена. В противном случае драйвер освобождает участок кэш-памяти (т.е. выталкивает из нее один блок) и запускает операцию обмена (считывает этот блок данных с УВВ). По окончании операции обмена моделируется прерывание и запускается программа обработки прерываний драйвера, адрес которой извлекается из записи в таблице прерываний. Далее драйвер переписывает информацию из кэш-памяти в область основной памяти, указанную программистом для ввода (buf).
4. По окончании операции ввода управление передается (обычно) диспетчеру процессов ядра ОС.

2.8 Начальная загрузка и выход на интерактивный режим работы



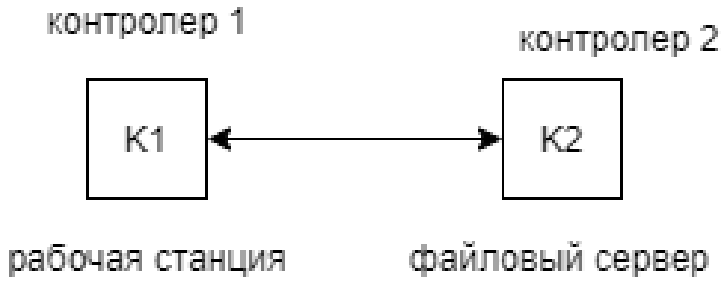


2.8.1 Протокол обмена

Или *методика передачи*, или *дисциплина линии*.

Чтобы обмениваться информацией по линии связи, между двумя компьютерами должно быть принято соглашение по следующим вопросам:

1. Метод синхронизации должен быть одинаков. То есть необходимо соглашение о том, где начинается и заканчивается байт во избежание получения ложной информации. Бывает *синхронный*, то есть как по рации, и *асинхронный*, который заключается в обрамлении парой бит, стартовым и стоповым, каждого отдельного байта
2. Направленность передачи. Однонаправленная передача — симпликс, в двух направлениях, но не одновременно — полудуплексное соединение, в двух направлениях и одновременно — дуплексное.
3. Скорость передачи. (бот = бит в секунду)
4. Система кодирования (картинка о СИП — сетевой интерфейсной плате)
5. Способ контроля. Кроме стартового и стопового битов к каждому байту может быть добавлен также бит паритета. Допустим, четный паритет сигнализирует о том, что количество единиц четное.



|картинка: рабочая станция + контроллер, сервер файловой системы + контроллер, СИП| + МНОГО СХЕМОК

СИП — сетевая интерфейсная плата (контроллер), позволяющая реализовывать ввод/вывод.

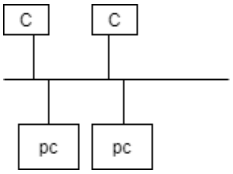
В качестве кабеля используются:

- витая пара
- коаксиальный кабель
- оптоволокно
- радиосигнал

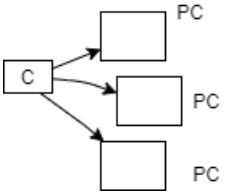
Типы СИП:

С — сервер, РС — рабочая станция

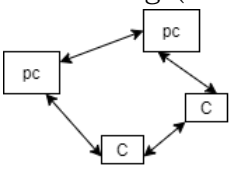
- Ethernet (сети с шиной топологий)



- Arcnet (топологическая звезда)



- Token-Ring (топологическое кольцо)



Топологии :

- шинная (Ethernet)
- звезда (arcnet/Ethernet)
- кольцо (token-ring)

Узел сети — это либо сервер, либо рабочая станция

Объединенная сеть представляет собой группу узлов одной сети, которая может быть связана с группой узлов другой сети с помощью моста, имеющей другую систему соединения или другую топологию.

Мост — узел с двумя или более сетевыми интерфейсными платами, каждая из которых связана с отдельной системой соединений. Может быть внутренним или внешним. Внутренний мост находится внутри сервера, внешний — на рабочей станции.

/картинка/

Шлюз — компьютер, с которого доступен интернет.

Каждая сеть имеет уникальный сетевой адрес. Каждая СИП, соединенная в общую систему, должна иметь один и тот же сетевой адрес. Сетевой адрес — 16-ричное число с длиной в 8 символов

Предметный указатель

В Указателе приведены *не все* термины, только те, которые могут находиться не в самых очевидных разделах или просто часто используются.

Навигация по билетам

Некоторые вопросы взяты из домашних рефератов. Информация для них (в сокращенном виде) взята с сайта hse.migalin.ru из этого документа.

Билет №1

Понятие операционной системы

ОС — набор программ и управляющих структур (таблиц), который обеспечит возможность использования аппаратуры компьютера и предоставит интерфейс для управления информацией и реализации ПО.

По видимому, включает в себя начало раздела Введение в ОС, в который входят функции ОС, какими ресурсами ОС управляет и немного о ядре ОС.

Управление памятью в ОС Unix на основе своппинга

В разделе Управление памятью на основе свопинга описано подробно. Если вкратце: это высокоуровневый алгоритм планирования процессов, подход, при котором программа многократно перекачивается на диск и обратно во время выполнения. Занимается этим свопер — верхний уровень планировщика. Бывают легкий и тяжелый свопинги: легкий — когда проблем с памятью нет и спокойно можно подгрузить нужный процесс, а тяжелый — когда для этого потребуется удалить из памяти некоторое количество других процессов.

На всякий случай про свопинг есть в разделе пораньше, до конкретизации про Unix.

Билет №2

Расслоение памяти. Регистр перемещений. Прерывания и опрос состояний

Это вопрос из ДЗ_1.

Расслоение памяти = интерливинг. Применяется для ускорения доступа к оперативной памяти. Соседние по адресам ячейки размещаются в различных модулях памяти, так что появляется возможность обращаться сразу к нескольким ячейкам одновременно.

Регистр перемещения. Обеспечивает возможность динамического перемещения программ в памяти.

В регистр перемещения заносится базовый адрес программы, хранящейся в основной памяти. Содержимое регистра перемещения прибавляется к каждому указанному в выполняемой программе адресу. Благодаря этому пользователь может писать программу так, как если бы она начиналась с нулевой ячейки памяти. Во время выполнения программы все исполнительные адреса обращений формируются с использованием регистра перемещения, и благодаря этому программа может реально размещаться в памяти совсем не в тех местах, которые она должна была бы занимать согласно адресам, указанным при трансляции.

Прерывания и опрос состояний

Опрос: одно устройства запрашивает у другого, находится ли то в определенном состоянии.

Прерывания: устройство может привлечь внимание другого устройства, чтобы сообщить об изменении своего состояния.

Алгоритм замещения страниц в ОС Unix

В разделе Управление памятью на основе свопинга со страничной подкачкой находится подраздел Алгоритм замещения страниц.

Вкратце: "страничный демон" (процесс с PID = 2) раз в 250 мс просыпается, проверяет, не слишком ли мало свободных страниц, если мало — некоторые страницы перемещает на диск. Если ОС (точнее, процесс с PID = 0, свопер) обнаруживает, что частота подкачки страниц слишком высока, а количество свободных страниц все время ниже параметра lostfree или максимума, то свопер начинает удалять из памяти процессы. Каждые несколько секунд (по умолчанию 2) свопер проверяет, есть ли на диске процессы в состоянии "готов" и выбирает из них процесс с наивысшим приоритетом и загружает его, только если свободных страниц достаточно.

Билет №3

Буферизация. Периферийные устройства. Защита памяти

Вопрос из ДЗ_1.

Буферизация. Буфер — это область основной памяти, предназначенная для промежуточного хранения данных при выполнении операций ввода-вывода.

При вводе, например, данные помещаются в буфер средствами канала ввода-вывода; после занесения данных в буфер процессор получает возможность доступа к этим данным. При вводе с *простой* буферизацией канал помещает дан-

ные в буфер, процессор обрабатывает эти данные, канал помещает следующие данные и т. д. В то время, когда канал заносит данные, изменять и обрабатывать их нельзя. Метод *двойной* буферизации позволяет совмещать выполнение операции вводавывода с обработкой данных.

Периферийные устройства — это аппаратура, которая позволяет вводить информацию в компьютер или выводить её из него. Неперифериные — процессор и память. Остальные периферийные бывают трех типов: ввода, вывода, хранения.

Защита памяти. Ограничивает диапазон адресов, к которым разрешается обращаться программе. Два варианта: граничные регистры, где указаны старший и младший адреса этого блока памяти, то есть все ячейки расположены рядом. И ключи защиты памяти – ячейки раскиданы по адресному пространству, и программе разрешается обращение к ячейкам памяти только тех областей, ключи которых совпадают с ключом данной программы.

Таблица процессов. Контекст процесса

Две первые подсекции раздела Процесс в UNIX содержат всю информацию.

Таблица процессов никогда не выгружается из памяти и содержит список дескрипторов процесса. Контекст процесса — структура данных, которая содержит информацию о процессе и доступ к которой есть только у ядра.

Билет №4

Таймер. Каналы ввода/вывода. Захват цикла памяти

Вопрос из ДЗ_1.

Таймер. Интервальный таймер — эффективный способ предотвращения /монополизации процессора одним из пользователей в многоабонентских системах. По истечении заданного интервала времени таймер генерирует сигнал прерывания для привлечения внимания процессора; по этому сигналу процессор может переключиться на обслуживание другого пользователя.
Часы истинного времени дают возможность компьютеру следить за реальным календарным временем с точностью до миллионных долей секунды, а при необходимости даже точнее.

Каналы ввода/вывода — это специализированные процессоры для управления В/В независимо от основного процессора. Нужны для параллелизации

работы аппаратуры и освобождения основного процессора от подавляющей части нагрузки, связанной с В/В.

Захват цикла памяти. К одному модулю может быть только одно обращение в каждый момент, и если каналу и процессору одновременно эта память нужна, то приоритет у канала. Он ее *захватывает*.

Планирование процессов в ОС Unix

Подробно в разделе Планирование процессов. Вкратце: есть два алгоритма: низкоуровневый и высокоуровневый. Высоко- — это свопинг. Низкий связан с подсчетом приоритета по формуле $\mathrm{PRI} = \mu\mathrm{CPU} + \eta\mathrm{NICE} + \mathrm{kBase}$, и выборе процесса для запуска в зависимости от этого приоритета.

Билет №5

Относительная адресация. Режимы работы ЭВМ

Вопрос из ДЗ_1.

Относительная адресация. Нужна, чтобы эффективно работать с большими адресными пространствами и перемещать программу в памяти. Адресация типа база+смещение, когда все адреса программы суммируются с содержимым базового регистра.

Режимы работы ЭВМ. *Режим задачи* ограничивает доступ программе к командам. Например, не позволяет непосредственно производить операции В/В, чтобы не выдать список всех паролей. *Режим супервизора* дает доступ ко всем командам, обычно выдается операционной системе.

Структурные компоненты системы управления вводом-выводом в ОС Unix

В разделе Система управления вводом-выводом в ОС UNIX подробно об управлении В/В в Unix. Именно структурные компоненты СУВВ впервые упоминаются здесь. Это всяческие таблицы связи, буферов, векторов прерываний и индексные дескрипторы специальных файлов.

Билет №6

Виртуальная память. Мультипроцессорная обработка. Прямой доступ к памяти

Виртуальные адреса. Позволяют работать с адресными пространствами большего, чем пространство основной памяти, размера. Во время выполнения программы преобразуются в реальные адреса. Вряд ли понадобится, но больше есть в разделе Управление памятью на основе свопинга со страничной подкачкой.

Мультипроцессорная обработка. Несколько процессоров работают с общей основной памятью и единой ОС. Обеспечивается координация их работы, чтобы два процессора не могли одновременно изменять содержимое одной ячейки памяти.

Прямой доступ к памяти. Когда требуется только одно прерывание на блок символов, передаваемых УВВ. Обычно так: Символы передаются в основную память по принципу захвата цикла: канал захватывает шину связи, передает один символ, отдает шину процессору, ждет, пока ему будут готовы передать еще символ, снова передает. А при прямом доступе к памяти прерывание одно, смысла запоминать состояние процессора нет.

Интерпретатор команд shell

Было на семинарах. Упоминались несколько модификаций: B-Shell (bash, by Stephen Bourne), C-Shell (си-подобный синтаксис, by Bill Joy), K-Shell (своеобразная комбинация C- и B-Shell, by David Korn), P-Shell (собственно стандарт POSIX).

Shell:

1. Интерпретирует команды
2. Обрабатывает имена файлов, определенные через метасимволы
3. Осуществляет переадресацию ввода-вывода
4. Создает среду пользователя
5. Поддерживает командный язык

Существуют *внутренние* (определенные в самом интерпретаторе и не требующие запуска нового процесса) и *внешние* (порождающие дополнительный процесс) команды.

Билет №7

Программирование на машинном языке. Ассемблеры и макропроцессоры. Компиляторы

Из ДЗ_1.

Машинный язык — ЯП, непосредственно воспринимаемый компьютером. Он машинно-зависимый чаще всего.

Ассемблеры и макропроцессоры. *Языки ассемблерного типа* используют мнемонические сокращения и слова естественного языка вместо чисел. Однако компьютеры не могут непосредственно воспринять программу на языке ассемблера, поэтому ее необходимо вначале перевести на машинный язык. Такой перевод осуществляется при помощи программы-транслятора, называемой *ассемблером*. Языки ассемблерного типа также являются машинно-зависимыми. *Макропроцессор* нужен для ускорения работы: когда видит макрокоманду, преобразует ее в соответствующие команды ассемеблера. Частый вопрос: чем отличаются компилятор и ассемблер. Ассемблер переводит на машинный язык за один этап, а компилятор — за несколько.

Компиляторы. Транслятор, который переводит высокоуровневый ЯП в машинный и генерирует объектную программу. Частый вопрос: отличия компилятора и интерпретатора. Компилятор быстрее, чем интерпретатор, потому что если уже видел команду и переводил ее, второй раз делать это не станет.

Управление памятью в ОС UNIX на основе свопинга со страничной подкачкой

Есть раздел Управление памятью на основе свопинга со страничной подкачкой. Это про виртуальную память. Реализуется страничная подкачка частично свопером, частично страничным демоном, который в основном спит, но раз в 250мс просыпается и, если свободных страниц мало, инициирует выгрузку страниц на диск. Используется алгоритм замещения страниц.

Билет №8

Конвейеризация. Иерархия памяти

Вопрос из ДЗ_1.

Конвейеризация. Бывает программной и аппаратной. Программная: результат работы программы передается на вход другой программе. Аппаратная: ко-

гда в процессоре как на конвейере одновременно находятся несколько команд в разных стадиях. Требует больше аппаратуры, но эффективность выполнения команд повышается. Неформально: когда начинается процесс, то он приходит «на все готовое» и как будто с конвейера берет все необходимое, ему не нужно где-то искать команды.

Иерархия памяти. Первичная память (оперативная, основная), вторичная, кэш. Частый дополнительный вопрос: что такое массовая память? Вторичная и массовая – это одно и то же. Еще частый вопрос: зачем нужен кэш? Кэш используется для ускорения работы процессора. Сначала процессор смотрит в кэш, и только потом идет дальше по иерархии памяти.

Таблица процессов. Контекст процесса

Две первые подсекции раздела Процесс в UNIX содержат всю информацию.

Таблица процессов никогда не выгружается из памяти и содержит список дескрипторов процесса. Контекст процесса — структура данных, которая содержит информацию о процессе и доступ к которой есть только у ядра.

Билет №9

Построение локальных вычислительных сетей

Нужно будет сказать, какие программные и аппаратные средства при этом используются. Аппаратные: модем, роутер, кабель Ethernet; программные: интерфейс сети и протокол.

На всякий случай есть раздел Начальная загрузка и выход на интерактивный режим раб

Состояние системы и виды взаимодействия в ОС UNIX. Таблица процессов. Создание процессов

Про взаимодействие тут: Состояния процессора.

Таблица процессов никогда не выгружается из памяти и содержит список дескрипторов процесса. В разделе Процесс в UNIX есть про дескрипторы процессов, которые являются строчками таблицы процессов.

Про создание процессов в разделе Операции над процессами.

Билет №10

Система управления вводом-выводом. Спулинг

Спулинг — это ввод-вывод с буферизацией. Между работающей программой и низкоскоростным устройством посредник — обычно буфер на диске. Вместо непосредственного вывода сохраняются данные на диск, чтобы распечатать по освобождению принтера. Благодаря этому текущая программа может быстрее завершиться, с тем чтобы другие программы могли быстрее начать работать.

СУВВ — часть ядра, выполняющая функции диспетчера УВВ. Про СУВВ в разделе Система управления вводом-выводом в ОС UNIX.

Физическая и логическая организация файловой системы в ОС Unix

Логически файл представляется непрерывной цепочкой блоков, а физически — списком блоков, которые могут быть разбросаны по всему дисковому пространству.
Достоинство: размер файла не ограничен.
Недостаток: нужно где-то хранить адреса блоков и их последовательность.

Подробно в разделе Логическая и физическая организация файловой системы.

Билет №11

Процедурно-ориентированные и проблемно-ориентированные языки. Интерпретаторы

Процедурно-ориентированные языки поддерживают несколько парадигм программирования, т.е. позволяют решать разные задачи. Проблемно-ориентированные, типа SQL, нацелены на решение определенных задач.

Интерпретатор — программа, выполняющая пошаговую обработку исходного кода. Частый вопрос: отличия компилятора и интерпретатора. Компилятор быстрее, чем интерпретатор, потому что если уже видел команду и переводил ее, второй раз делать это не станет.

Создание процессов в ОС Unix

Про создание процессов в разделе Операции над процессами.

Билет №12

Абсолютные и перемещающие загрузчики. Связывающие загрузчики и редакторы связей

Вопрос из ДЗ_1.

Загрузчик — программа, которая размещает команды и данные программы в ячейках основной памяти. *Абсолютный* размещает в те ячейки, адреса которых указаны в программе на машинном языке. *Перемещающийся* может загружать в различные места в зависимости, например, от наличия свободного участка памяти.

Связывающий загрузчик во время загрузки объединяет программы и загружает в основную память. Редактор связей делает то же, но еще создает загрузочный модуль (для будущего использования, записывается во внешнюю память). Он работает быстрее, потому что один раз сохранил, и пересобирать не надо. Связывающий при очищении памяти собирает заново.

Индексный дескриптор файла ОС UNIX

Короткий раздел Индексный дескриптор файла.

Билет №13

Микропрограммирование. Эмуляция. Горизонтальный и вертикальный микрокод

Вопрос из ДЗ_1.

Микрокод — язык более низкого уровня, чем ассемблер и машинный язык. *Вертикальный* похож на обычный машинный. Задает, например, пересылку элементов данных между регистрами (пересылку только одного регистра). *Горизонтальный* позволяет задавать параллельную операцию пересылки данных для многих регистров.

Эмуляция — метод, позволяющий сделать одну вычислительную машину функциональным эквивалентом другой. Набор команд машинного языка *эмулируемого* компьютера микропрограммируется на *эмулирующем*.

Индексный дескриптор файла. Поддержка файлов ядром ОС UNIX

Про дескриптор вообще здесь: Дескриптор файла.
Конкретно в Unix: Индексный дескриптор файла.

Поддержка файлов здесь: Поддержка файлов ядром ОС UNIX.

Билет №14

Понятие процесса. Операции на процессом

Процесс — программа в стадии выполнения вместе с текущими значениями счетчиков команд, регистров, переменных.

Про операции в разделе Операции над процессами.

Интерпретатор команд Shell в Unix

Было на семинарах. Упоминались несколько модификаций: B-Shell (bash, by Stephen Bourne), C-Shell (си-подобный синтаксис, by Bill Joy), K-Shell (своеобразная комбинация C- и B-Shell, by David Korn), P-Shell (собственно стандарт POSIX).

Shell:

1. Интерпретирует команды
2. Обрабатывает имена файлов, определенные через метасимволы
3. Осуществляет переадресацию ввода-вывода
4. Создает среду пользователя
5. Поддерживает командный язык

Существуют *внутренние* (определенные в самом интерпретаторе и не требующие запуска нового процесса) и *внешние* (порождающие дополнительный процесс) команды.

Билет №15

Блок управления процессом. Операции над процессом

Блок управления процессом = дескриптор. Про него здесь: Дескриптор процесса.

Про операции в разделе Операции над процессами.

Приоритет процесса в ОС Unix

Приоритет процесса описывается в разделе Планирование процессов.

Раз в секунду приоритет каждого процесса пересчитывается по формуле:

$$\text{PRI} = \mu\text{CPU} + \eta\text{NICE} + k\text{Base}$$

И на основе нового приоритета прикрепляется к соответствующей очереди.

CPU — время выполнения процесса в тиках таймера начиная с создания (при каждом тике таймера счетчик использования процесса увеличивается на 1). Значение этого счетчика добавляется к приоритету процесса, тем самым увеличивая его.

NICE — пользовательская составляющая приоритета процесса, является числом в пределах от -20 до 20. Может быть изменена с помощью системных вызовов от 0 до 20. Изначально NICE=0.

BASE — отрицательное число, определяющееся событием, которое ожидал процесс. Отрицательные значения для дискового/терминального ввода-вывода, для срабатывания таймера, выполнения системного вызова жетско прошиты в ОС и могут быть изменены только перекомпиляцией всей системы. Добавляется к приоритету поцесса после выхода из состояния блокировки. Блокировка может произойти: из-за соответствующего ввода-вывода, прерывания, системного вызова или включения ожидания.

Билет №16

Понятие прерывания. Схемы прерываний

Прерывания: устройство может привлечь внимание другого устройства, чтобы сообщить об изменении своего состояния. То есть прерывания — сигналы, сообщающие о наступлении какого-либо события.

Что такое *схемы* прерываний непонятно, но было про *типы*: Типы прерываний.

Вся оперативная память делится на три части: одна отводится под ОС, вторая под карту памяти, третья под процессы пользователя. КП — это таблица, каждая запись этой таблицы содержит 9 полей. Первые 2 поля записи либо пустые (если страница памяти занята каким-то процессом, иначе — указатели на последующий и пред блоки)

Страница памяти — блоки одинаковой длины. Карта памяти характеризует каждую страницу памяти.

Физическая и логическая организация файловой системы ОС UNIX

Логически файл представляется непрерывной цепочкой блоков, а физически — списком блоков, которые могут быть разбросаны по всему дисковому пространству.

Достоинство: размер файла не ограничен.
Недостаток: нужно где-то хранить адреса блоков и их последовательность.

Подробно в разделе Логическая и физическая организация файловой системы.

Билет №17

Понятия параллельного процесса, асинхронного процесса, взаимоисключения, потока, семафора, мьютекса, монитора, тупика

Все в разделе Операции над процессами.

Процессы называются параллельными, если существуют одновременно. Параллельные процессы называются асинхронными, если работают совершенно независимо друг от друга.

Взаимоисключение потоков - обеспечение использования каждого разделяемого ресурса только одним потоком от момента выделения ресурса этому потоку до момента освобождения ресурса. (Настя Зайцева)

У любого процесса есть хотя бы один поток управления (последовательность команд для выполнения конкретного действия) и один счетчик команд.

В некоторых современных ОС реализована возможность создания нескольких потоков управления в одном процессе. Такие потоки получили названия нитей (threads). У всех нитей процесса одно общее адресное пространство, но у каждой личные счетчик команд, регистры и состояния. Информация о нитях записывается в таблицу в ОС. Каждая ее запись описывает один поток.

Мониторы представляют собой программные модули (объекты), которые реализуют (инкапсулируют) все необходимые действия с разделяемым ресурсом. (Настя Зайцева)

Семафор — системная переменная, которая является объектом ядра ОС и используется в основном для синхронизации процессов. Может принимать неотрицательные целочисленные значения.

Мьютекс / mutex — упрощенная версия семафора, которая может принимать только значения 0 и 1.

Процесс находится в состоянии тупика (deadlock, клинч), если он ожидает событие, которое никогда не произойдет:

Управление памятью в ОС UNIX на основе страничной подкачки. Карта памяти

Все тут: Управление памятью на основе свопинга со страничной подкачкой.

Про настройку карты памяти для процесса-сына здесь.

Билет №18

Стратегия управления памятью. Концепции распределения памяти. Алгоритмы замещения

Довольно короткий подраздел Стратегии.

Типы файлов в файловой системе ОС UNIX. Топология файловой системы. Последние изменения в файловой системе ОС UNIX

Про типы файлов: Типы файлов.
Бывают обычные, каталоги, FIFO-файлы, специальные и гнезда.

Топология — это рассказать про древовидную структуру файловой системы. Во главе находится корневая директория (корневой каталог), дальше, на нижних уровнях, идут как каталоги, так и обычные файлы.

Последние изменения в файловой системе ОС UNIX

1. Кэширование файлов

2. Увеличение размера имени файла. В начальных версиях под имя файла отводилось 16 байт (2 байта под № индексного дескриптора, 14 под само имя). Сейчас под имя отводится 256 байт (4 под № дескриптора, 252 под само имя).

3. Форматирование диска на блоки двух размеров. При форматировании диск разбивается на некоторое количество блоков (раньше одного, а теперь двух размеров), в которые потом записываются файлы.

Билет №19

Организация памяти при связном распределении. Оверлейные перекрытия

Оверлейные перекрытия. Если очередная часть программы не помещается, то необходимо разбить ее на еще более мелкие блоки. Это делает компоновщик. Программист же указывает компоновщику, в какой последовательности нужно запускать части программы.

Организация памяти при связном распределении: Подходы. Это организация памяти с разделами фиксированного, переменного размеров, свопинг и виртуальная память.

Понятие стандартного ввода/вывода и переназначения стандартного ввода/вывода в ОС UNIX

Все библиотечные функции, все программы (утилиты) обращаются при вводе/выводе к одним и тем же файлам — файлам стандартного в/в.

При работе процесса в его таблице файловых дескрипторов автоматически создаются три первые записи: дескриптор стандартного ввода, вывода и стандартного вывода ошибок.

При необходимости стандартный ввод-вывод можно переназначить путем удаления одной из этих записей и "подстановки" на ее место другого дескриптора, например, дескриптор файла, из которого нужно считать или в который нужно записать информацию.

Делается это с помощью СВ *dup* и *dup2*.

Билет №20

Организация памяти на основе мультипрограммирования с переменными разделами

Мультипрограммирование = разделить память для многих программ. Имеется в виду раздел про организацию памяти с разделами переменного размера. При таком подходе каждая программа занимает столько памяти, сколько ей нужно.

Системные вызовы ввода/вывода ОС UNIX

Было на семинарах.

int read(int fd, char* buf, int n)

fd – пользовательский дескриптор открытого для чтения файла,

buf – указатель на буфер в программе пользователя, куда помещаются данные из файла,

n – количество байтов, которое требуется прочитать.

Возвращает количество байт, успешно прочитанных и сохраненных в buf. Обычно это число равно n. Но если в файле меньше n байт, то и выдаст меньше n. Если встретился признак конца файла, то возвращается 0. Если прочитать не удалось, вернет -1.

int write(int fd, char* buf, int n)

fd – пользовательский дескриптор открытого для записи файла,

buf – указатель на буфер в программе пользователя, откуда надо считать данные,

n – количество байтов, которое требуется считать из buf.

Возвращает количество байт, успешно записанных в файл. Или -1 в случае неудачи.

`write(1, "aaaa", 2)` ⇒ 2

`write(1, "aaaa", 10)` ⇒ 10.

Билет №21

Организация памяти на основе своппинга

В разделе Управление памятью на основе свопинга описано подробно. Если вкратце: это высокоуровневый алгоритм планирования процессов, подход, при котором программа многократно перекачивается на диск и обратно во время выполнения. Занимается этим свопер — верхний уровень планировщика. Бывают легкий и тяжелый свопинги: легкий — когда проблем с памятью нет и спокойно можно подгрузить нужный процесс, а тяжелый — когда для этого потребуется удалить из памяти некоторое количество других процессов.

На всякий случай про свопинг есть в разделе пораньше, до конкретизации про Unix.

Системные вызовы работы с файлами open, creat, close, dup, dup2

int open(char* name, int flag)

name — указатель на строку символов, содержащую имя открываемого/создаваемого файла,

flag — режим открытия файла (права доступа)

Используется для открытия/создания обычных файлов на диске.

Возвращает пользовательский дескриптор открытого файла (№ первой свободной записи в таблице пользовательских дескрипторов открытых файлов процесса).

```
close(1);
open("a.txt", 1) ⇒ 1
```

int creat(char* name, int mode)

name — указатель на строку символов, содержащую имя файла,

mode — режим открытия файла (права доступа).

Создает и открывает на запись файл. Если такой файл же существовал, "обнуляет" его.

Возвращает пользовательский дескриптор открытого файла (№ первой свободной записи в таблице пользовательских дескрипторов открытых файлов процесса).

int close(int fd)

fd — пользовательский дескриптор открытого файла.

Возвращает 0, если файл закрылся успешно, и -1 в случае неудачи.

int dup(int old_fd)

Копирует пользовательский дескриптор открытого файла old_fd в первую свободную запись таблицы пользовательских дескрипторов открытых файлов процессаю

В случае успеха возвращает номер этой записи, иначе — -1.

int dup2(int old_fd, int n_fd)

Делает то же, что и *dup*, но здесь явно указывается номер записи, куда нужно скопировать дескриптор. Как следствие, с помощью *dup2* можно закрыть файл, а с помощью *dup* нет. (Татьяна Козликина)

Билет №22

Концепция виртуальной памяти

В разделе Управление памятью на основе свопинга со страничной подкачкой говорится про свопинг, виртуальную память. Про виртуальные адреса здесь.

Системные вызовы fork, wait, pipe

int fork()

Позволяет получить дубликат текущего процесса. После его применения процесс разделяется на две идентичные копии, которые продолжают выполняться как два независимых процесса. Возвращаемое значение: процесс-сын получает от *fork* код ответа 0, в то время как процесс-отец — идентификатор, под которым запущен процесс-сын.

int wait(int* s)

В аргумент s записывается причина гибели сына. В биты с 0 по 6 помещаются нули, если сын завершен через *exit()*. 7-й бит нулевой, только если процесс завершился нормально, то есть его образ сохранен, в противном случае он равен 1. Биты с 8 по 11 содержат аргумент функции *exit()*.

int pipe(int fd[2])

fd[2] — массив, в который записываются два пользовательских дескриптора: f[0] — на чтение, f[1] — на запись.

Создает канал связи между двумя взаимодействующими процессами. Фацл, который создается pipe-ом, называется межпроцессным.

Синхронизация обена через межпроцессный канал устроена таким образом, что если процесс читает пустой канал, то он будет ждать появления данных

Билет №23

Понятия файла, символьного набора, организации файлов

Файл — именованный набор данных, состоящий из записей с заданной структурой, наложенной пользователем.

Символьный набор — это когда каждому символу противопоставляется численный код. Символьный набор кем-то предлагается и утверждается сообществом. Иногда это называется способом кодировки. Известные символьные наборы - ASCII, UNICOD, EBCDIC и др. (на сегодня их больше 50 и перечислять все не имеет смысла).

Больше в разделе Управление файловой системой.

Понятие системного вызова в ОС UNIX. Системные вызовы system и execl

Когда во время выполнения программы необходимо выполнить другую программу или предоставить системные функции, используется аппарат системных вызовов. Синтаксически применение системного вызова похоже на вызов подпрограммы, однако исполняемый код системного вызова находится в ядре операционной системы, а не в загрузочном модуле.

int system(char* cmd)

cmd — текст, команда командной строки.

Позволяет в рамках выполняющейся программы (процесса) выщвать новую программу (процесс).

В случае успеха возвращает 0, иначе — -1.

int execl(char* cmd, char* arg0, char* arg1, . . . , NULL)

cmd — имя вызываемой программы (полное),

Прочие argN — внешние аргументы для нее.

Вызывает программу, убивая уже выполняющуюся.

Возвращаемое значение: -1 в случае неудачи, в случае успеха управление не возвращает.

Билет №24

Реализация файловой системы. Реализация файлов и каталогов

Одноименный раздел Реализация файловой системы.

Логическая и физическая организация системы управления вводом-выводом в ОС UNIX

Логическая и физическая организация файловой системы.

Билет №25

Блок управления файлом

Видимо, по аналогии с блоком управления процессом, блок управления файлом — это синоним для дескриптора. В общем виде здесь: Дескриптор файла, конкретно про Unix здесь: Индексный дескриптор файла.

Структура процесса ОС UNIX. Создание процессов в ОС UNIX

Здесь: Процесс в UNIX.

Билет №26

Средства файловой системы. Топология файловой системы

Файлами управляет ОС. Их структура, именование, защита, использование, доступ к ним относятся к той части ОС, которая называется файловой системой.

ОС содержит (предоставляет), как правило, следующие средства:

1. Средства доступа к файлам

2. Средства управления файлами (удаление, переименование, создания, копирование и т.д.)
3. Средства размещения файлов
4. Средства сохранения целостности файлов

Топология — это рассказать про древовидную структуру файловой системы. Во главе находится корневая директория (корневой каталог), дальше, на нижних уровнях, идут как каталоги, так и обычные файлы.

Структура ОС UNIX. Ядро и процессы

Структура ОС UNIX:
— Ядро
— Системные программы
Раздел Ядро ОС.

Билет №27

Блок управления файлом. Средства файловой системы

Видимо, по аналогии с блоком управления процессом, блок управления файлом — это синоним для дескриптора. В общем виде здесь: Дескриптор файла, конкретно про Unix здесь: Индексный дескриптор файла.

Файлами управляет ОС. Их структура, именование, защита, использование, доступ к ним относятся к той части ОС, которая называется файловой системой.

ОС содержит (предоставляет), как правило, следующие средства:

1. Средства доступа к файлам
2. Средства управления файлами (удаление, переименование, создания, копирование и т.д.)
3. Средства размещения файлов
4. Средства сохранения целостности файлов

Основные характеристики ОС UNIX

Туть: UNIX-подобные системы.

Билет №28

Поддержка файлов ядром ОС UNIX. Индексный дескриптор файла

Поддержка файлов ядром ОС UNIX.
Индексный дескриптор файла.

Планирование процессов в ОС UNIX

Планирование процессов.

Билет №29

Поддержка процессов ядром ОС UNIX

Программная секция ядра состоит из 2-х частей, являющихся целыми наборами программ. Нас интересует первая секция:

1. *Диспетчер процессов*

К нему относится набор программ, который определяет последовательность выполнения процессов, распределяет ресурсы системы (в том числе память), обрабатывает системные вызовы и сигналы.

Непонятно, что ожидается в ответе. Но вот целый раздел: Процесс в UNIX.

Управление памятью в ОС UNIX на основе свопинга

В разделе Управление памятью на основе свопинга описано подробно. Если вкратце: это высокоуровневый алгоритм планирования процессов, подход, при котором программа многократно перекачивается на диск и обратно во время выполнения. Занимается этим свопер — верхний уровень планировщика. Бывают легкий и тяжелый свопинги: легкий — когда проблем с памятью нет и спокойно можно подгрузить нужный процесс, а тяжелый — когда для этого потребуется удалить из памяти некоторое количество других процессов.

Билет №30

Система управления вводом-выводом. Программное обеспечение

В целом про ввод-вывод здесь: Управление устройствами ввода-вывода. Конкретно в Unix: Система управления вводом-выводом в ОС UNIX.

Про ПО есть только в общем разделе вне Unix: ПО системы управления УВВ.

Поддержка файлов ядром ОС UNIX

Поддержка файлов ядром ОС UNIX.