Я НЕНАВИЖУ ОСИ

2024-2025 гг.

1. Этапы развития вычислительной техники и программного обеспечения.

1 поколение: Компьютеры этого поколения строились на электронно вакуумных лампах, решали задачи энергетики и баллистики.
Середина 40-х — начало 50-х годов XX века.

2 поколение: Компьютеры основываются на использовании полупроводниковых приборов — диодов и транзисторов, которые по функционалу, размеру и энергопотреблению в десятки раз превосходили возможности компьютеров 1-го поколения.
Уже у них появилась пакетная обработка данных, аппаратная поддержка мультипрограммирования и файловая система.
Конец 50-х годов — вторая половина 60-х годов XX века.

3 поколение: Элементная база — интегральные схемы, что привело к увеличению производительности компьютеров, снижению их размеров и веса. Появились драйверы устройств и виртуальные устройства.
Конец 60-х — начало 70-х годов XX века.

4 и последующие поколения: Элементная база основана на больших интегральных схемах, компьютеры стали иметь более понятные интерфейсы для обычных пользователей. Наибольшее развитие получила сеть Internet.
Начало 70-х годов XX века — наше время.

2. Структура вычислительной системы. Ресурсы ВС — физические ресурсы, виртуальные ресурсы. Уровень операционной системы.

Уровни ВС:
1. Аппаратный уровень
2. Уровень управления физическими устройствами
3. Уровень управления виртуальными устройствами
4. Уровень систем программирования
5. Уровень прикладных программ

Физическим ресурсом называется некий аппаратный компонент компьютера, обладающий правилами использования.

Физический ресурс описывается тремя **характеристиками**:
1. Объемные свойства
2. Степень используемости
3. Интерфейс программного взаимодействия

Виртуальным ресурсом называется ресурс некоторые характеристики которого (возможно все), реализованы программно.

Ресурсами ВС называется совокупность физических и виртуальных устройств.

Уровень ОС представляет собой совокупность двух уровней — управления физическими устройствами и управления виртуальными устройствами.

Драйвер физического устройства - программа основанная на использовании команд и средств управления физического устройства, предназначенная для организации работы с ним.

Драйвер виртуального устройства — программа, обеспечивающая существование и использование соответствующего устройства.

Уровень ОС на уровне управления физическими устройствами представляется совокупностью драйверов физических устройств, а на уровне управления виртуальными устройствами - совокупностью драйверов виртуальных устройств, обобщающих характеристики и свойства различных устройств одного типа.

3. Структура вычислительной системы. Ресурсы ВС — физические ресурсы, виртуальные ресурсы. Уровень систем программирования.

Уровни ВС:
1. Аппаратный уровень
2. Уровень управления физическими устройствами
3. Уровень управления виртуальными устройствами
4. Уровень систем программирования
5. Уровень прикладных программ

Физическим ресурсом называется некий аппаратный компонент компьютера, обладающий правилами использования.

Физический ресурс описывается тремя **характеристиками**:
1. Объемные свойства
2. Степень используемости
3. Интерфейс программного взаимодействия

Виртуальным ресурсом называется ресурс некоторые характеристики которого (возможно все), реализованы программно.

Ресурсами ВС называется совокупность физических и виртуальных устройств.

Система программирования — комплекс программ, предназначенных для поддержания и реализации жизненного цикла создаваемой программы.

Жизненный цикл программы — состоит из 4х шагов:
1. Проектирование
2. Кодирование
3. Тестирование и отладка
4. Запуск программы и ее сопровождение

Уровень систем программирования представляет инструментальные средства для разработки программных систем, каждая из которых предназначена для решения своей задачи.

4. Структура вычислительной системы. Ресурсы ВС — физические ресурсы, виртуальные ресурсы. Уровень прикладных систем.

Уровни ВС:
1. Аппаратный уровень
2. Уровень управления физическими устройствами
3. Уровень управления виртуальными устройствами
4. Уровень систем программирования
5. Уровень прикладных программ

Физическим ресурсом называется некий аппаратный компонент компьютера, обладающий правилами использования.

Физический ресурс описывается тремя **характеристиками**:
1. Объемные свойства
2. Степень используемости
3. Интерфейс программного взаимодействия

Виртуальным ресурсом называется ресурс некоторые характеристики которого (возможно все), реализованы программно.

Ресурсами ВС называется совокупность физических и виртуальных устройств.

Прикладная система - программная система, ориентированная на решение или автоматизацию задач из конкретной предметной области.

На **уровне прикладных систем** реализуются средства формализации взаимодействия с программными системами, используемыми для решения конкретных прикладных задач.

5. Основы архитектуры компьютера. Основные компоненты и характеристики, структура ЦП.

Структура компьютера Фон Неймана:
1. **Центральный процессор**
2. **Оперативное запоминающее устройство** — устройство памяти компьютера, в котором располагается выполняемая программа
3. **Внешние устройства**

Структура ЦП:
1. **Устройство управления (УУ) -** отвечает за координацию выполнения команд.
2. **Арифметико-логическое устройство (АЛУ)** — отвечает за выполнение команд, арифметической или логической обработки операндов.
3. **Регистровая память -** память реализованная в процессоре.
4. **КЭШ-память (L1)** — специализированная быстрая, по сравнению с ОЗУ, память, размещенная в процессоре.

6. Основы архитектуры компьютера. Основные компоненты и характеристики, ОЗУ, расслоение памяти.

Структура компьютера Фон Неймана:
1. **Центральный процессор**
2. **Оперативное запоминающее устройство**
3. **Внешние устройства**

ОЗУ — устройство памяти компьютера, в котором хранится исполняемая программа. Состоит из совокупности ячеек, каждая из которых имеет структуру - <Тег><Машинное слово>.
Тег — поле служебной информации. Может использоваться для создания различных механизмов контроля системы, например использовать тег для контроля целостности информации или разделения команд и данных.
Машинное слово - поле программно изменяемой информации в ОЗУ.

Время доступа — время между запросом на чтение слова из ОЗУ и получением содержимого этого слова.
Длительность цикла памяти — минимальное время между началом текущего и последующего обращения к ОЗУ.

Расслоение памяти — оптимизационное решение организации ОЗУ, позволяющее снизить время работы с данными.
Адресное пространство ОЗУ делится на $K=2^L$ независимых подустройств — **банок**.
Ячейки с последовательными адресами хранятся в соседних банках.
Младшие L битов адреса — номер банка памяти.

Модели расслоения памяти:
1. **С общим контроллером доступа к памяти**

Все банки координирует один контроллер доступа к памяти. Это позволяет сравнять длительность цикла памяти с временем доступа.

2. **С K контроллерами доступа к памяти**

Каждый банк координируется отдельным контроллером. Это позволяет совершать последовательные запросы длиной не более K слов практически параллельно.

7. Основы архитектуры компьютера. Основные компоненты и характеристики, Кэширование ОЗУ.

Структура компьютера Фон Неймана:
1. **Центральный процессор**
2. **Оперативное запоминающее устройство**
3. **Внешние устройства**

КЭШ-память - высокоскоростное устройство хранения данных, используемое для буферизации работы ЦП с ОЗУ.
Скорость доступа к информации в КЭШэ соизмерима со скоростью обработки информации в ЦП.
Организован в виде последовательности блоков фиксированного размера (2^K). Каждый блок имеет **тэг**, в котором хранится служебная информация, описывающая данный блок.
При обращении ЦП к ОЗУ, он сначала обращается к КЭШу, по тэгу однозначно аппаратно определяет нахождение этих данных в блоке.
В случае нахождения данных (**попадания**), данные берутся из КЭШа.
В случае отсутствия данных по определенной стратегии происходит **вытеснение** — обновление содержимого КЭШа.

Стратегии вытеснения:

- **Сквозное кэширование** — при вытеснении блока из КЭШа, происходит только загрузка содержимого нового блока, а при изменении данных обновление происходит как в КЭШе так и в ОЗУ.
- **LRU** — вытесняется наименее «популярный» блок.
- **Кэширование с обратной связью** — при записи по адресу, содержимое которого кэшируется, происходит обновление соответствующей по этому адресу информации только в блоке КЭШа, а также установка специального **тега модификации**. При вытеснении, если тег модификации установлен, то содержимое блока перед вытеснением «сбрасывается в память», тем самым уменьшается частота выполнения записи в ОЗУ.

8. Основы архитектуры компьютера. Аппарат прерываний, Последовательность действий в вычислительной системе при обработке прерываний.

Прерывание — событие в компьютере, при возникновении которого в процессоре происходит предопределенная последовательность действий.
Виды прерываний:
1. **Внутренние —** возникают в схемах контроля работы процессора (Например: деление на 0).
2. **Внешние** — возникают в результате взаимодействия процессора с внешними устройствами (Например: ввод символа с клавиатуры).

Этапы аппаратной обработки прерываний:
1. Аппаратный этап - реакция аппаратуры компьютера на возникновение прерывания и автоматическое выполнение предопределенной последовательности действий.
Этапы:
- Завершение текущей команды
- Переход ЦП в режим блокировки прерываний
- Сохранение актуального состояния процессора
- Переход на программный этап обработки прерывания.

2. Программный этап — при необходимости, передача управления на некоторую предопределенную точку ОЗУ, в предположении что с этой точки находится программа-обработчик прерывания.
- Анализ типа прерывания:
  - Если оно **короткое**, то есть его обработка не требует дополнительных ресурсов, то прерывание обрабатывается, выключается режим блокировки прерывания, восстанавливается состояние процессора и управление передается в прерванную точку. (Например: прерывание по таймеру)
  - Если оно **фатальное**, то есть после него невозможно продолжить выполнение программы, то выключается режим блокировки прерываний и управление передается в часть ОС, прекращающую выполнение прерванной программы. (Например: деление на 0)
- Полное сохранение контекста (все регистры ЦП).
- Снятие блокировки прерываний.
- Передача управления программе-обработчику.
- Завершение обработки прерываний.

9. Основы архитектуры компьютера. Внешние устройства, Организация управления и потоков данных при обмене с внешними устройствами.

Обмен данными с ВУ осуществляется:

- записями фиксированного размера — **блоками**
- записями произвольного размера

Доступ к данным ВЗУ:

- **Последовательный** - для просмотра i-ой записи, требуется просмотреть все предыдущие i-1 записей. (Например: магнитная лента)
- **Прямой** — для просмотра i-ой записи, не требуется просматривать предыдущие. (Например: диск)

Модели организации обмена с ВУ:

- **Синхронная** — при такой организации, процесс приостанавливается на время ожидания завершения обмена с ВЗУ.
- **Асинхронная** — при такой организации появляется возможность «параллельной» обработки взаимодействия с ВЗУ, с последующим информированием ВС о результатах запросов с помощью прерываний.

Два потока информации:

- Поток данных
- Поток управляющей информации (команд)

Модели организации управления ВУ:

- **Непосредственное управление процессором** — в ЦП интегрированы управляющие схемы, обеспечивающие отправку команд для управления ВУ, синхронизацию обмена данными и обработку информации, перемещая ее из регистров в ОЗУ.
- **Синхронный** - метод с использованием **контроллеров**, обеспечивающих разгрузку процессора при обмене данными, передавая информацию блоками.
- **Асинхронный** — в этой модели также используются контроллеры, но уже не требуется ожидания процессором окончания обработки команд. Необходим аппарат прерываний.
- Модель с использованием **DMA** — контроллеров прямого доступа, позволяющих исключить участие ЦП в обработке потока данных.
- Модель с использованием **канала ввода — вывода**, специального компьютера между ЦП и устройством, обрабатывающего высокоуровневые запросы на обмен.

10. Основы архитектуры компьютера. Иерархия памяти.

Иерархия устройств хранения памяти в порядке:

- Увеличения емкости
- Увеличения времени доступа к памяти
- Уменьшению скоростью обмена с памятью
- Увеличению времени хранения данных

1. РОН
2. КЭШ L1
3. КЭШ L2
4. ОЗУ
5. ВЗУ прямого доступа с внутренней КЭШ буферизацией
6. ВЗУ прямого доступа без КЭШ буферизации
7. ВЗУ долговременного хранения данных.

11. Аппаратная поддержка ОС. Мультипрограммный режим.

Мультипрограммный режим ОС — режим при котором возможна организация переключения выполнения с одной программы на другую.

Состояния выполнения программ:

- **Выполнение** — команды программы исполняются ЦП
- **Блокировка** — программа ожидает завершения обмена данными
- **Готовность к выполнению** — программа завершила обмен данными, но в ЦП исполняются команды другой программы.

Корректное функционирования мультипрограммного режима - результат работы конкретной программы не зависит от деятельности других программ.

Необходимые аппаратные средства для корректного функционирования мультипрограммного режима:

- **Аппарат защиты памяти**
- **Привилегированный режим работы ЦП**
- **Аппарат прерываний** (минимум — прерывание по таймеру).

12. Аппаратная поддержка ОС. Организация регистровой памяти ЦП.

Регистровая память — высокоскоростная память реализованная в ЦП. Представляет собой совокупность регистров, предназначенных для хранения:

- Управляющей информации
- Операндов
- Результатов выполняемых команд

Регистры делятся на:

- **Регистры общего назначения (РОН):**
  - Состоят из доступных для программ регистров, предназначены для хранения операндов, адресов операндов и результатов выполнения команд.
  - Позволяют оптимизировать программы, при помещении наиболее часто используемых операндов на регистры
  - Могут иметь машинную типизацию (целочисленные, с плавающей точкой)
  - Могут быть скалярными или векторными.
- **Специальные регистры:**
  - Состоят из служебных регистров, определяемых архитектурой ЦП
  - Примеры: Счетчик команд, указатель инструкции, указатель стека, слово состояние процессора.

13. Аппаратная поддержка ОС. Виртуальная оперативная память.

Аппарат виртуальной памяти — аппаратные средства компьютера, обеспечивающие преобразование виртуальных адресов, используемых в программе, к адресам физической памяти, в которой размещена программа.

Есть 2 проблемы организации ОЗУ:

- **Проблема перемещаемости программы по ОЗУ** — проблема состоит в том, что программа может быть настроена на определенный диапазон физической памяти, который на данный момент занят, но при этом существует область свободной памяти, в которую эта программа бы поместилась.
- **Проблема фрагментации ОЗУ** - проблема заключается в том, что при использовании ЭВМ, возникают ситуации, когда суммарный объем всех свободных областей достаточно большой, чтобы вместить туда программу, но каждая из этих областей по отдельность слишком мала для этой программы.

Решение проблемы перемещаемости программы по ОЗУ - **базирование адресов**.
Вводится специальный **регистр базы**, в который помещается адрес размещения программы в ОЗУ и каждый исполнительный физический адрес представляется в виде $\mathbf{A_{прог} = A_{прог} + <R_{базы}>}$,
где $\mathbf{A_{прог}}$ — относительный адрес, а $\mathbf{A_{физ}}$ — абсолютный адрес.

Решение проблем перемещаемости программы по ОЗУ и фрагментации ОЗУ — **страничная виртуальная память**.
Все адресное пространство представляется в виде последовательности **страниц** фиксированного размера (2^K).
Каждый виртуальный адрес имеет структуру:
<Номер виртуальной страницы><Номер в странице>, где последние K бит соответствуют номеру страницы.
Таблица страниц — аппаратная регистровая таблица в ЦП, предназначенная для организации соответствия между виртуальными и физическими страницами конкретной программы.

- Каждой виртуальной странице программы ставится в соответствии строка в таблице страниц
- В каждой строке таблицы находится номер физической страницы, в которой размещается соответствующая виртуальная страница
- **Аппарат виртуальной страничной памяти** аппаратно преобразовывает номер виртуальной страницы, в номер физической.

14. Аппаратная поддержка ОС. Пример организации страничной виртуальной памяти.

Рассмотрим пример организации страничной виртуальной памяти:

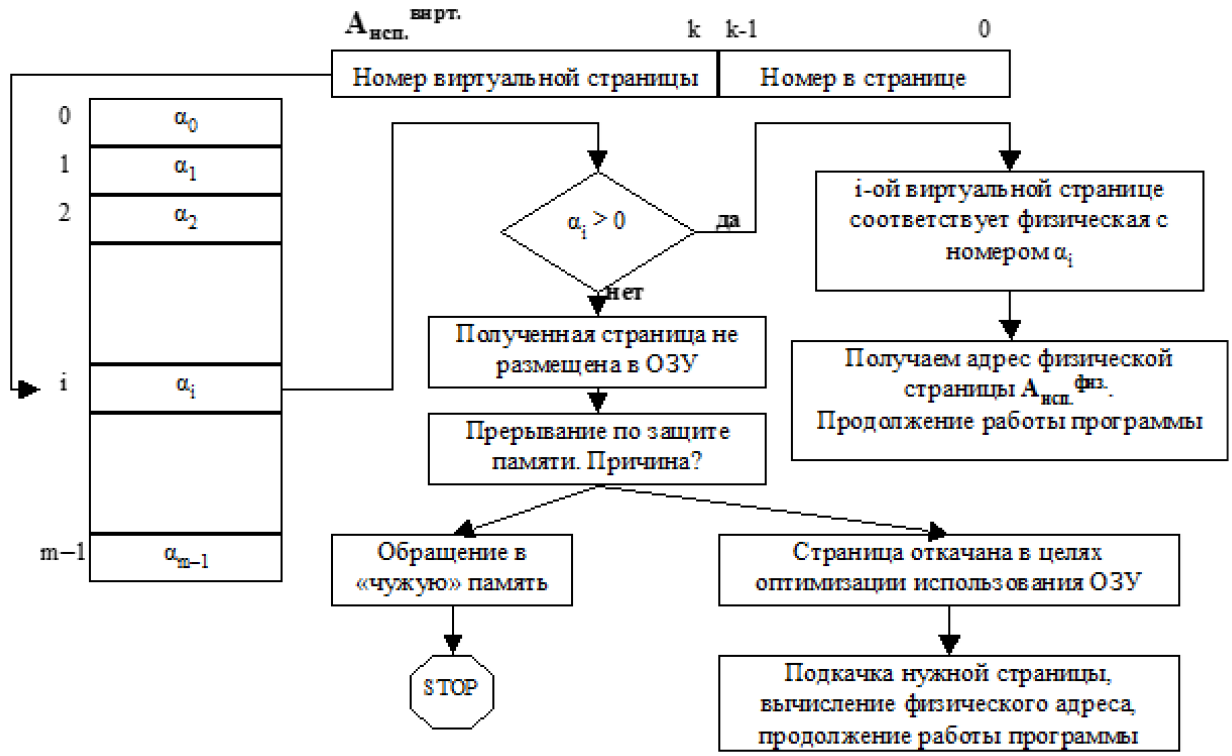
Пусть в таблице страниц N строк.
Пусть i-ой виртуальной странице соответствует α_i физическая страница.

- Если у программы есть i-ая виртуальная страница, то $\alpha_i > 0$
- Если у программы нет i-ой виртуальной страницы, то $\alpha_i = 0$ (Т.к нулевая физическая страница считается страницей ОС)

Если $\alpha_i = 0$, то возникает прерывание по защите памяти по одной из двух причин:

- Либо у программы нет i-ой страницы и она попытается обратиться за область своего адресного пространства
- Либо в ОЗУ еще не загружена i-ая страница

Иначе, номер виртуальной страницы преобразуется в номер физической страницы.



15. Многомашинные, многопроцессорные ассоциации. Классификация. Примеры.

Архитектуры классифицируются по:

- Потоку команд (инструкций)
- Потоку данных

Классификация архитектур многопроцессорных ассоциаций (Флинна):

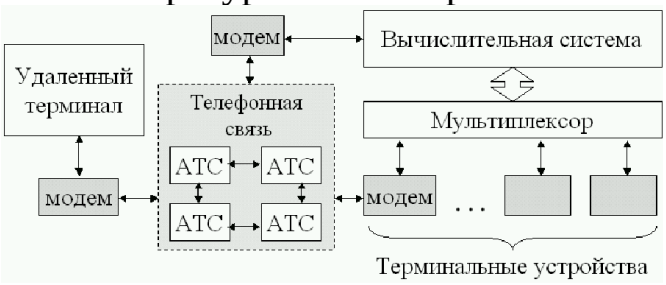
- **SISD** (single instruction, single data stream) — компьютеры с единственным ЦП, близкие к компьютеру Фон Неймана.
- **SIMD** (single instruction, multiple data stream) — компьютеры основанные на векторной обработке данных (Например: **Cray-1**)
- **MISD** (multiple instruction, single data stream) — несколько ЦП обрабатывают одни и те же данные, вырожденный случай.
- **MIMD** (multiple instruction, multiple data stream) — множество процессоров одновременно выполняют разные последовательности команд над своими данными.

MIMD системы делятся на:

- **Системы с общей ОП** — ЦП имеет неепосредственный доступ к любой ячейке ОП
  - **UMA** (uniform memory access) — системы с однородным доступом к ОП (по различным характеристикам)
  - **NUMA** (non uniform memory access) — системы с неоднородным доступом к ОП
  - **SMP** (symmetric multiprocessor)
- **Системы с распределенной ОП** — объединение нескольких компьютеров, каждый из которых имеет свои ЦП и ОЗУ и которые не имеют доступа к ОЗУ других компьютеров
  - **MPP** (Massively Parallel Processors) — ЦП с массовым параллелизмом, дорогостоящие узкоспециализированные ВС, высокоэффективные при решении определенных задач.
  - **COW** (Cluster of workstations) — кластеры рабочих станций, выход из строя одной компоненты кластера не приводит к отказу системы, может привести только к понижению производительности.

16. Многомашинные, многопроцессорные ассоциации. Терминальные комплексы. Компьютерные сети.

Терминальные комплексы — многомашинная ассоциация, предназначенная для организации массового доступа удаленных и локальных пользователей к ресурсам некоторой ВС.



Компоненты ТК:

- Основная ВС
- Локальные и удаленные **мультиплексоры** — устройства подключения нескольких каналов к одному
- Локальные и удаленные **терминалы** — внешние устройство, позволяющее осуществлять обмен информации между самими терминалами и компьютерами, к которому они подключены.
  - Локальные — подключается к ВС напрямую или через мультиплексор.
  - Удаленные — подключаются к ВС через коммуникационную среду напрямую или через мультиплексор.

Каналы и линии связи

Два типа каналов:

- **Выделенный** — линия связи, соединяющая терминал и компьютер, выделяемая на постоянной основе (дорого и неэффективно)
- **Коммутируемый** - линия связи выделяется между двумя взаимодействующими устройствами, только на промежуток времени, когда осуществляется взаимодействие

Организация канала:

- **Канал точка-точка** — взаимодействие двух устройств напрямую
- **Многоточечный канал** — взаимодействие многих каналов

Направление движениия информации:

- **Симплексный** — канал, позволяющий осуществлять передачу информации только в одном направлении (Например: громкоговоритель)
- **Дуплексный** — канал, позволяющий осуществлять передачу инфомрации в двух направлениях (Например: телефон)
- **Полудуплексный** — передача информации в двух направлениях, но в один момент времени только в одном (Например: рация)

Компьютерная сеть — объединение нескольких компьютеров, взаимодействующих через коммуникационную среду.
Коммуникационная среда — объединнение каналов и/или линий связи и специализированного оборудования, предназначенного для организации передачи данных.
Свойства компьютерных сетей:

- **Связность компьютеров** — два компьютера связны, если они могут обмениваться информацией.
- **Расширяемость сети** — сеть может быть расширеня территориально и функционально.
- **Распределенность** обработки информации — результат работы может быть совокупностью результатов работы разных компьютеров.
- **Симметричность** интерфейсов обменаа внутри сети — возможность произвольного распределения функций внутри сети.

Компьютеры в сети:

- **Абонентские —** основные компьютеры, хосты
- **Коммуникационные компьютеры —** вспомогательные, выполняющие функции по обеспечению функционирования сети.

Сообщение — логически целостностная порция данных произвольного размера.
Сеанс связи — время, в течение которого сетевые устройства обмениваются сообщениями.
Три модели организации каналов:

- Сеть коммутации каналов — выделение каналов происходит на весь сеанс связи
  - Плюсы: постоянная готовность, минимальные требования к коммуникационным компьютерам, уменьшение накладных расходов по передаче данных, фиксированная определенная пропускная способность сети
  - Минусы: дорого, избыточность сети, неэффективность использования канала, при сбоях — повторение всего сеанса связи.
- **Сети коммутации сообщений** — взаимодействие происходит в виде последовательности обменов сообщениями, для каждого сообщения происходит выделение канала.
  - Плюсы: эффективность использования каналов
  - Минусы: Требования к коммуникационным компьютерам (буферизация сообщений), неопределенная пропускная способность сети, при сбое — повторить отправку сообщения.

- **Сети коммутационных пакетов** — сообщения разделяются на блоки фиксированного размера - **пакеты**, которые в свою очередь передаются
  - Передача - цепочками пакетов в произвольном порядке доходящих до устройства-получателя, который в свою очередь восстанавливает по ним сообщение.
  - Пакет содержит информацию: о целостности пакета, об отправителе и получателе, информацию для сборки назад в сообщение.
  - Плюсы: фиксированная определенная пропускная способность, необходимость передачи только одного пакета при сбое
  - Минусы: увеличение количества передаваемой информации до 10 раз, необходимость сохранения пакетов для сборки, необходимость сборки пакетов.

17. Операционные системы. Основные компоненты и логические функции. Базовые понятия: ядро, процесс, ресурс, системные вызовы. Структурная организация ОС.

ОС — комплекс программ, обеспечивающий контроль за существованием, распределение и использованием ресурсов ВС.

- *Контроль за существованием ресурсов* — обеспечение реализации виртуальных ресурсов и средств доступа к физическим ресурсам.
- *Контроль за распределением ресурсов* — выбор стратегии распределения ресурсов ВС
- *Контроль за использованием ресурсов* — предоставление программам средств для доступа к ресурсам ВС.

Процесс — совокупность машинных команд и данных, обрабатывающаяся в рамках ВС и обладающая правами на владение некоторыми ресурсами ВС.

Модели владения ресурсами процессом:

- **Эксклюзивные ресурсы** — пока данный процесс владеет этим ресурсом, другие не имеют к нему доступа.
- **Разделяемые ресурсы** — могут одновременно принадлежать нескольким процессам.

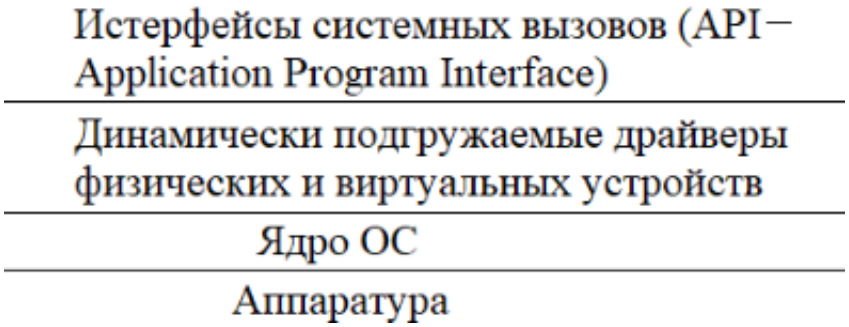
Логические функции ОС:

- *Управление процессами* -создание и организация его обработки и выполнения
- *Управление оперативной памятью*
- *Планирование* - преобразование потока запросов к ресурсам в структурированную очередь
- *Управление устройствами и наличие файловой системы*
- *Сетевое взаимодействие*
- *Безопасность*

Ядро — резидентная (постоянно размещаемая в ОП) часть ОС, реализующая базовую функциональность ОС, работающая в режиме супервизора.

Системный вызов — обращение к ОС за предоставление той или иной функции.

Структурная организация ОС:



- **Аппаратура —** основа всего
- **Ядро** — реализует некий базовый набор функций, который релизуется посредством включенных в ядро драйверов (зависит от реализации ОС)
- **Драйвера физических и виртуальных ресурсов** — драйверы, состав которых при установке и загрузке системы может меняться:
  - **Резидентные драйверы** — подгружаются ОС в процессе загрузки и находятся в ней до завершения работы
  - **Нерезидентные** — подгружаются ОС на время работы с соответствующим устройством
- **Интерфейсы системных вызовов** - обеспечение возможности обращения из процесссов пользователей за функциями ОС.

18. Операционные системы. Пакетная ОС, ОС разделения времени, ОС реального времени, распределенные и сетевые ОС.

Пакетная ОС - ОС, оперирующая **пакетами программ** — совокупностью программ, которые должны быть обработаны ВС.

Пакетный режим — способ организации мультизадачности, при котором переключение процессов происходит только по причинам:

- Завершения процесса
- Прерывания по вводу-выводу
- Зацикливания

ОС разделения времени — ОС, процессы которой могут непрерывно использовать ЦП только **квантами времени** — фиксированными ОС промежутками времени работы ЦП, выделяемыми для непрерывной работы процесса

Критерий эффективность — максимальная загрузка ЦП

Режим разделения времени — планирование времени ЦП с помощью квантов времени:

- Процессы переключаются по необходимости или по истечению кванта времени
- Процессы разделяются на группы и для каждой группы определяется свой квант времени
- Группы приоритезируются и процессы переключаются, если появляется более приоритетный

Критерий эффективности — минимизация отклика системы на запросы пользователя

ОС реального времени

Режим реального времени — любой процесс в ВС гарантированно завершается за определенное время

Распределенные ОС — ОС, обеспечивающие функционирование ВС, как комплекса компьютеров

- На каждом компьютере функционирует отдельное ядро
- ВС предоставляет распределенные функции ОС — управление набором процессов, распределенная ФС и тд.

Сетевая ОС — ОС, обеспечивающие функционирование ВС в пределах сети.

- Устанавливается на каждом компьютере сети
- Обеспечивает функционирование распределенных приложений (реализация функций которых, распределена по всем компьютерам)

19. Организация сетевого взаимодействия. Эталонная модель ISO/OSI. Протокол, интерфейс. Стек протоколов. Логическое взаимодействие сетевых устройств.

Модель ISO/OSI организации сетевого взаимодействия:

7. Прикладной уровень
   - Стандартизуются средства взаимодействия прикладных программ.
6. Уровень представления
   - Унификация представления данных
5. Сеансовый уровень
   - Определяются функции начала/конца сеанса, подтверждение полномочий.
   - Организуются средства для дигностирования и обработки ошибок.
4. Транспортный уровень
   - Обеспечение взаимодействия между взаимодействующими сетевыми программами, а также выбор способа транспортировки.
3. Сетевой уровень
   - Обеспечение взаимодействия между сетевыми устройствами
2. Канальный уровень
   - Решается задача передачи по линии связя **кадров** — порций данных, содержащих информацию для определения и устранения ошибок.
   - Решаются задачи доступности и синхронизации канала, а также задача локальной адресации сетевых устройств в рамках локальной сети.
1. Физический уровень
   - Передача неструктурированного потока двоичной информации

Протокол — описание сообщений и правил, по которым ВС осуществляет обмен данными (обеспечивает взаимодействие в сети на одном уровне).

Интерфейс — правила взаимодействия вышестоящего уровня с нижестоящим

Стек протоколов — перечень протоколов (от первого уровня до максимально реализованного), реализованных в ВС.

Взаимодействие сетевых устройств:

- Передача данных с максимально реализованного уровня на первый
- Передача информации по коммуникационной среде
- Получение информации из коммуникационной среды другим устройством
- Передача информации с первого уровня, до максимально реализованного уровня на другом устройстве.

20. Организация сетевого взаимодействия. Семейство протоколов TCP/IP, соответствие модели ISO/OSI. Взаимодействие между уровнями протоколов семейства TCP/IP. IP адресация.

Уровни протоколов TCP/IP

4. **Уровень прикладных программ** — состостоит из прикладных программ и процессов, использующих сеть и доступных пользователю. Стандартизирется представление данных
(Уровень прикладных программ и представления ISO/OSI)

3. **Транспортный уровень** — обеспечивает доставку данных, средства для поддержки логических соединений между прикладными программами. (Не всегда осуществляется контроль за ошибками и их коррекция)
(Сеансовый и транспортный уровни)

2. **Межсетевой уровень** —обеспечивает взаимодействие между сетевыми устройствами (без установления соединения).
(Сетевой уровень)

1. **Уровень доступа к сети** — специфицирует доступ к физической сети
(Физический и канальный уровни ISO/OSI)

Взаимодействие между уровнями протоколов TCP/IP

- Все протоколы пакетные
- Перемещая информацию от нижнего к верхнему уровню, сообщение разбивается на пакеты соответствующего уровня, затем передается на следующий уровень.

Для именования на межсетевом уровне используются **IP- адреса:**
Рассмотрим 32-х битные
Класс A: <0><Сеть><Хост>, под хост 24 бит, под сеть 7 бит
Класс B: <10><Сеть><Хост>, под хост 16 бит, под сеть 14 бит
Класс C: <110><Сеть><Хост>, под хост 8 бит, под сеть 21 бит
Класс D: <1110><Группа>
Класс E: <1111><Группа>

21. Управление процессами. Определение процесса, типы. Жизненный цикл, состояния процесса. Свопинг. Модели жизненного цикла процесса. Контекст процесса.

Процесс — совокупность машинных команд и данных, обрабатываемых в рамках ВС и обладающих правами на владения некоторыми ресурсами ВС.

Типы процессов:

- **Полновесные** — процессы, выполняющиеся внутри защищенных участков оперативной памяти.
- **Легковесные** — процессы, работающие в мультипрограммном режиме одновременно с активировавшим их полновесным процессом и использующие его виртуальное адресное пространство

Жизненный цикл процесса — связь состояний, в которых может находится обрабатываемый в системе процесс

- Формирование
- Выполнение
- Ожидание постановки на исполнение
- Завершение процесса.

Свопинг — откачка процесса во внешню память.

Обобщенная модель жизненного цикла процесса:

- После обращение к СВ fork() процесс формируется и практически сразу попадает в очередь процессов, готовых к выполнению.
- Планировщик определяет может ли процесс переходить в состояние выполнение или назад, в состояние готовности.
  - Чем дольше процесс выполняется, тем ниже приоритет, а чем дольше процесс находится в состоянии готовности — тем выше приоритет.
- В состоянии выполнения процесс может работать в режиме супервизора или в пользовательском режиме
- Из режима выполнения процесс может как перейти в состояние готовности, так и заблокироваться (ожидая завершение некоторого действия с внешним окружением)
- После обращение к системному вызову _exit() -процесс переходит в состояние «зомби»
- После получение отцом информации о завершении данного процесса он завершает свой жизненный цикл.

Контекст процесса — совокупность данных, характеризующих актуальное состояние процесса

- **Пользовательская (программная) составляющая** — текущее состояние программы
  - Совокупность машинных команд и данных, размещенных в ОЗУ и характеризующих выполнение процесса
- **Аппаратная составляющая** — текущее состояние ЦП в момент выполнения процесса
  - Совокупность регистров, настроек ЦП и др.
- **Системная составляющая** — структуры данных ОС, содержащие характеристики процесса
  - Идентификаторы (PID — Process Identifier)
  - Содержимое регистров ЦП
  - Информация для управления процессом (состояние процесса, приоритет и др.)
  - Копия аппаратной составляющей для остановленного процесса

Для исполняемого процесса актуальна аппаратная составляющая, а для остановленного — системная

22. Реализация процессов в ОС UNIX. Определение процесса. Контекст, тело процесса. Состояния процесса. Аппарат системных вызовов в ОС UNIX.

Процесс в UNIX — объект зарегистрированный в таблице процессов.
Таблица процессов — программная таблица, предназначенная для регистрации процессов.

- Номер записи в таблице — PID.
- Запись таблице — ссылка на контекст процесса

Процесс в UNIX — объект порожденный системным вызовом fork(), обеспечивающим создание копии текущего процесса
(исключение - процессы с PID = 0,1)
Состояния процесса — формирование, выполнение, ожидание, завершение.
Контекст процесса — совокупность данных, характеризующих актуальное состояние процесса:

- **Системная составляющая** — информация ОС о процессе:
  - PID, PPID, PGID,SID,UID, EUID, GID, EGID
  - Параметры для планировщика времени ЦП — текущее состояние процесса, приоритет процесса
  - Аргументы командной строки и переменные окружения
  - Текущий и корневой каталоги
  - Таблица ФД процесса
  - Диспозиция сигналов — обработчики процесса длля сигналов
  - Информация о сигналах, ожидающих доставки в процесс
  - Счетчики потребленных ресурсов (времени ЦП и тд)
  - Сохраненные значение аппаратной составляющей
- **Пользовательская составляющая** - тело процесса (состоящее из 2х частей):
  - **Сегмент кода** — неизменяемая часть тела, содержит машинные команды и константы, может быть изменена посредством СВ.
  - **Сегмент данных** включающий область данных, область разделяемой памяти, область стека
- **Аппаратная составляющая** — состояние ЦП, характеризующее процесс в момент исполнения (некоторые регистры, таблицы ЦП..)

Системным вызовом в ОС UNIX — средство ОС, предоставляемое процессом, посредством которого они могут обращаться к ядру за выполнением тех или иных функций, выполняемых в привилегированном режиме ОС.

23. Реализация процессов в ОС UNIX. Базовые средства управления процессами в ОС UNIX. Загрузка
ОС UNIX, формирование нулевого и первого процессов.

Системный вызов **fork**() создает новый процесс, являющийся копией вызывающего процесса

- В таблицу процессов заносится новая запись, порожденный процесс получает новый PID
- В системе создается контекст нового процесса, большая часть которого **наследуется** от старого:
  - Сегменты кода и данных, аргументы командной строки и переменные окружения, таблица открытых ФД, диспозиция сигналов, разделяемые ресурсы и т.д.
- При этом **не наследуются**:
  - PID, PPID, информация о сигналах, ожидающих доставки в процесс, счетчики потребленных ресурсов и др.
- Родительский и порожденный процессы продолжают выполнение одной и той же программы с точки возврата из fork().
- СВ возвращает порожденному процессу — 0, родительскому PID порожденного процесса в случае успеха, в случае ошибки -1.

Системные вызовы **getpid(), getppid(), getpgid(), getsid()** возвращают соответствующий идентификатор для текущего процесса.

Системные вызовы семейства **exec()** осуществляют подмену тела процесса, при успешном выполнении. Системные вызовы принимают путь к исполнительному файлу и в том или ином виде аргументы командной строки и переменные окружения. В случае неуспеха возвращает управление в точку вызова.

Системный вызов **_exit(int status)** - завершает текущий процесс, через параметр status процессу передается информация о статусе своего завершения.

Системный вызов **pid_t wait(int *status) —** при вызове блокирует родительский процесс до завершения одного из сыновьих процессов, возвращая через параметр статус завершения, а сама функция вернет PID завершившегося процесса. В случае если детей нет, то он вернет -1.

Загрузка ОС UNIX

- При включении компьютера, управление передается на предопределеный физический адресс ОЗУ, начиная с которого находится ПЗУ, в нем располагается аппаратный загрузчик.
- Аппаратный загрузчик информируется о системных устройствах компьютера, с предположением что на этих устройствах — ОС.
- Строится приоритетный перечень системных устройств, в каждом из которых он ищет программный загрузчик в нулевом блоке.
- После его нахождения аппаратный загрузчик считывает программный загрузчик и передает управление на предопределенную точку входа программного загрузчика, после чего последний начинает работать.
- Программный загрузчик загружает исполняемый файл, в корне ФС, в физическую память и передает управление на точку входа.

После входа в ядро ОС, оно:

- Инициализирует все аппаратные устройства (Пр: часы)
- Инициализирует все структуры данных (Пр: таблица процессов)
- Формирует процесс с номером 0, записывая некоторую информацию в 0ую запись таблицы процессов.

Процесс с PID = 0:

- не имеет корректной информации
- не имеет кодового сегмента
- существует в течении всего времени работы системы

Процесс с PID = 1:

- Корректен с точки зрения структуры таблицы процессов
- Имеет контекст
- Имеет сегмент кода
- Имеет выделенную память, в которую ОС загружает реализацию СВ **exec()**, после чего идет обращение по стандартной схеме fork — exec, на замену тела процесса на процесс **init**
- Существует в течении всего времени работы системы.

Сам процесс **init** смотрит системные данные и запускает работу ОС либо в однопользовательском, либо в многопользовательском режиме.

24. Реализация нитей в ОС UNIX. Пример программирования нитей.

Основная программа:
создаёт нить, дожидается её окончания, печатает на экран системный идентификатор нити и время в секундах, которое заняло выполнялнение нити.
Порождённая нить:
в точке входа засыпает на несколько секунд (от 0 до 10), после чего завершается.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <pthread.h>
#include <unistd.h>
/*функция - точка входа в нить */
void* thr_fn(void *arg)
{
	/*засыпаем на 0..10 секунд */
	sleep(time(0) % 10);
	pthread_exit(NULL);
}
int main()
{
	pthread_t th_id;
	void *pret;
	/*создаём нить и запоминаем время создания */
	pthread_create(&th_id, NULL, thr_fn, NULL);
	time_t tm_start = time(0);
	/*ожидаем завершения ранее созданной нити */
	pthread_join(th_id, &pret);
	printf("Thread %#x execution time = %u\n", th_id, time(0) -tm_start);
	return 0;
}
```

25. Взаимодействие процессов. Разделяемые ресурсы. Критические секции. Взаимное исключение.Тупики.

Процессы называются **параллельными**, если их выполнение хотя бы частично перекрывается по времени.

- **Независимые** — если они используют независящие друг от друга множества ресурсов.
- **Взаимодействующие** — если они совместно используют ресурсы, и выполнение одного процесса может оказать влияние на результат другого.

Разделение ресурса — совместное использование ресурса ВС двумя или более параллельными процессами, когда каждый из процессов некоторое время владеет этим ресурсом.

Если разделяемый ресурс доступен только одному процессу в каждый момент времени, то он называется **критическим ресурсом**.

Часть программы, в которой осуществляется работа с критическим ресурсом, называется **критической секцией**.

Чтобы избежать гонок за разделяемые ресурсы, организовывается **взаимное исключение** — способ организации мультипроцессного взаимодействия, при котором при работе одного из процессов с разделяемым ресурсом, другие не могут иметь доступ к этому ресурсу.

При этом могут возникнуть проблемы:

- **Блокировки** — ситуации, когда доступ к разделяемому ресурсу одного из процессов не обеспечивается из-за активности других, более приоритетных процессов.
- **Тупика** — ситуации, когда конкурирующие за критический ресурс процессы взаимоблокируются.

26. Взаимодействие процессов. Некоторые способы реализации взаимного исключения: семафоры Дейкстры, мониторы, обмен сообщениями.

Семафоры Дейкстры

Имеется тип данных — **семафор**, переменные этого типа могут принимать целочисленные значения.
Над этими переменными определены следующие **атомарные** операции:
1) Опустить семафор **down(S)**:

- Проверяется значение семафора S:
  - Если S > 0, то операция уменьшает его на 1.
  - Иначе — процесс блокируется, причем связанная с ним операция down еще не завершена.

2) Поднять семафор **up(S)**:

- Увеличивает значение семафора на 1, причем если в системе присутствуют процессы, блокированные ранее при выполнении down на этом семафоре, то один из них разблокируется и завершает выполнение операции down.

Двоичный семафор — семафор, максимальное значение которого равно 1

Вывод: Семафоры Дейкстры — низкоуровневые средства синхронизации, для корректной работы которых необходимо наличие специальных атомарных машинных команд.

Мониторы Хоара

Монитор — специализированный модуль, включающий в себя совокупность процедур и функций, а также структур данных, с которыми работают эти процедуры и функции.
Его свойства:

- Структуры данных монитора доступны толоько через обращения к процедурам и функциям этого монитора
- Считается, что процесс входит в монитор, если он вызывает одну из процедур или функций монитора
- В каждый момент времени внутри монитора может находится не более одного процессса

Если процессс пытается войти в монитор, в котором уже находится другой процесс, то в зависимости от стратегии он:

- Либо получает отказ
- Либо блокируется, становясь в очередь

Вывод: чтобы защитить разделяемые структуры данных, их достаточно поместить внутрь монитора, вместе с процедурами, которые представляют собой критические секции для обработки этих структур данных

Механизм передачи сообщений

Основан на двух функциональных примитивах:

- **send** — отправить сообщение
- **receive** — принять сообщение

Операции посылки/приема сообщений могут быть:

- **блокирующими —** процессы с блокирующими операциями блокируются до момента завершения (получения, приема) операции
- **неблокирующими —** операции send/receive происходят без блокировок

Адресация

- **Прямая** — при такой адресации в операциях указывается конкретный адрес получателя/отправителя
- **Косвенная** — при такой адресации конкретный адрес не указывается, а отправляется в некоторый пул, из которого по определенной стратегии доступа, извлекается сообщение

Вывод: Механизм передачи сообщений совмещает средства передачи информации и средства синхронизации. Является средством организации межпроцессного взаимодействия в системах с разделенной памятью.

27. Взаимодействие процессов. Классические задачи синхронизации процессов. “Обедающие философы”.

Обедающие философы

Есть круглый стол, за которым сидит группа философов.
Они едят из общей миски, стоящей в центре стола. Для приема пищи они используют две вилки: в левой и правой руках.
Вилки располагаются по одной между каждыми двумя философами.
У философа есть два состояния:

- Размышлять
- Взять две вилки и начать принимать пищу

Поведение философов независимо.
Задача состоит в том, чтобы организовать доступ к вилкам.
Описание алгоритма решения:

- Создается массив семафоров, символизирующий философов
- Создается массив состояний для философов, его элементы принимают значения для состания бездействия, ожидания ресурса и использования ресурса.
- Создается один двоичный семафор.
- Цикл работы философа:
  - Философ думает
  - Пытается взять вилки (меняя свое состоянииe на ожидание ресурса)
  - После этого проверяется состояние соседей.
  - Если оба они не едят, то в состояние использования переключается текущий философ (процесс)
  - Иначе — философ (процесс) блокируется.
  - После того как философ поел, производится последовательность действий по освобождению вилок. Состояние использование переключается на состояние бездействия, а соседние процессы, которые ожидали освобождения получают доступ к вилкам.
  - Сам двоичный семафор использовался в функциях освобождения и взятия двух вилок, дабы исключить одновременное разрешение на получение для двух процессов.

```
#define N 5                     /* количество философов */
#define LEFT (i-1)%N            /* номер левого соседа для i-го философа */
#define RIGHT (i+1)%N           /*номер правого соседа для i-го философа*/
#define THINKING 0              /* состояния философов: «думает»,«голоден»,«кушает»*/
#define HUNGRY 1
#define EATING 2
/*массив состояний каждого из философов, инициализированный нулями*/
int state[N];
/* семафор для доступа в критическую секцию */
semaphore mutex = 1;
/*массив семафоров по одному на каждого из философов,инициализированный нулями*/
semaphore s[N];
```

```
/* Процесс-философ (i = 0..N-1) */
void Philosopher(int i)
{
      while(TRUE)
      {
       Think();
/* философ берёт вилки или блокируется */
       TakeForks(i);
       Eat();
       PutForks(i);
       }
}
```

```
 /* получение вилок */
void TakeForks(int i)
{
       /* вход в критическую секцию */
       down(&mutex);
       state[i] = HUNGRY;
       Test(i);
       /* выход из критической секции */
      up(&mutex);
      down(&s[i]);
}
```

```
/* освобождение вилок */
void PutForks(int i)
{
      /* вход в критическую секцию */
     down(&mutex);
     state[i] = THINKING;
     Test(LEFT);
     Test(RIGHT);
     /* выход из критической секции */
    up(&mutex);
}
```

```
/* функция проверки возможности получения вилок – проверяется состояние соседей данного философа */
void Test(int i)
{
      if(state[i] == HUNGRY &&
      state[LEFT] != EATING &&
      state[RIGHT] != EATING)
      {
            state[i] = EATING;
            up(&s[i]);
      }
}
```

28. Взаимодействие процессов. Классические задачи синхронизации процессов. “Читатели и писатели”.

Задача **Читателей и писателей** заключается в следующем:
Пусть у нас есть некоторая информационная система и два типа процессов: читатели и писатели.
В произвольный момент времени читать информацию может любое число процессов, однако если писатель начнет свою работу, то остальные процессы(и читатели и писатели) должны быть заблокированы. Как организовать такую систему?

Алгоритм решения: Сделаем читателей — более приоритетными так, что писатели должны будут ожидать, пока читатели не выйдут из системы.

- Потребуется два двоичных семафора и глобальная переменная-счетчик читателей.
- Первый семафор используется для корректного подсчета читателей
- Второй используется для блокировки доступа к системе в случае, когда первый читатель заходит в нее.
- Цикл работы читателя:
  - Опускается семафор на переменную-счетчик
  - счетчик увеличивается на 1, если это первый читатель и доступ к системе блокируется
  - Далее читатель выполняет свою работу, уменьшает счетчик на 1
  - В случае если это последний читатель открывает доступ к базе данных
- Цикл жизни писателя
  - Опускает семафор
  - Работает
  - Выходит из системы, поднимая семафор

```
/* процесс-читатель */
void Reader(void)
{
    down(&mutex);
    rc = rc + 1;
    if(rc == 1)
        down(&db);
    up(&mutex);
    ReadDataBase();
    down(&mutex);
    rc = rc – 1;

    if(rc == 0)
        up(&db);
    up(&mutex);
    UseDataRead();
}
```

```
/*семафор для корректного подсчета*/
semaphore mutex = 1;
/* семафор для доступа к базе данных */
semaphore db = 1;
/* количество читателей внутри хранилища */
int rc = 0;

/* процесс-писатель */
void Writer(void)
{
      ThinkUpData();
      down(&db);
      WriteDataBase(); /
       up(&db);
}
```

29. Взаимодействие процессов. Классические задачи синхронизации процессов. «Спящий парикмахер».

Пусть есть парикмахерская, в которой работает один барбер.
В парикмахерской есть N мест ожидания.
Если парикмахер не обслуживает клиента, то он спит и ожидающему клиенту необходимо его разбудить.
Если в зале ожидания не хватает места для нового клиента, то клиенту поступает отказ.
Необходимо организовать работу такой системы

Алгоритм решения:

- Понадобится семафор с количеством мест ожидания
- Понадобится семафор для синхронизации доступа к последующей переменной
- Понадобится переменная с количеством клиентов
- Понадобится семафор для состояния парикмахера
- Цикл парикмахера выглядит так:
  - Если клиентов нет, то блокируемся
  - Иначе уменьшаем число клиентов на 1 и выставляем парикмахеру состояние готовности
  - Один из клиентов займет парикмахера и будет обслужен, затем в начало цикла.
- Клиент входит в критическую секцию, проверяет есть ли свободные места
  - Если нет, то уходит
  - Если есть то увеличиваем число клиентов в очереди
  - Если клиент является единственным клиентом барбера, то он его разбудит

```
#define CHAIRS 5
semaphore customers = 0;
semaphore barbers = 0;
semaphore mutex = 1;
int waiting = 0;

/*Барбер*/
void Barber(void)
{
   while(TRUE)
   {
      down(&customers);
      down(&mutex);
      waiting = waiting – 1;
      up(&barbers);
      up(&mutex);
      CutHair();
    }
}
```

```
/* Посетитель */
void Customer(void)
{
    down(&mutex);
    if(waiting < CHAIRS)
    {
        waiting = waiting + 1;
        up(&customers);
        up(&mutex);
        down(&barbers);
        GetHaircut();
     }
     else
     {
        up(&mutex);
      }
}
```

30. Базовые средства взаимодействия процессов в ОС UNIX. Сигналы. Примеры программирования.

В ОС UNIX реализован **аппарат сигналов**, позволяющий влиять на работу процесса.
Сигналы отправляются из некоторого предопределенного множества.
Инициатором отправки сигналов процессу может быть:

- **ОС** — сигналы ОС посылаются в случае наступления некоторых строго предопределенных ситуаций, каждой из которых сопоставлен свой сигнал (Например: при делении на 0, случается прерывание и ОС отправляет сигнал процесу, породившему данную ошибку)
- **Другой процесс** (Например: остановка процесса сигналом, после нажатия комбинации клавиш CTRL+C в консоли)

Каждому сигналу сопоставляется какое-то заранее определенное в ОС целочисленное значение.

Три типа реакции процесса на сигнал:

- Обработка сигнала по умолчанию
- Перехватывание обработки пришедшего сигнала — при получении сигнала вызывается функия **обработчик сигнала**
- Игнорирование процессом сигнала

При единовременном получении различных сигналов порядок их обработки —**не определен.**
Пример:
При получении сигнала SIGINT (что соответствует нажатиюCTRL+C) четыре раза вызывается специальный обработчик. На пятый же раз происходит
обработка сигнала обработчиком по умолчанию, в результате чего процесс завершается.

```
#include <sys/types.h>
#include <signal.h>
#include <stdio.h>
int count = 0;

/* функция – пользовательский обработчик сигнала */
void SigHndlr (int s)
{
        printf("\n I got SIGINT %d time(s) \n", ++count);
        if (count == 5)
                signal (SIGINT, SIG_DFL);  /* ставим обработчик сигнала по умолчанию */
        else
                signal (SIGINT, SigHndlr);   /* восстанавливаем обработчик сигнала */
}

int main()
{
        /* установка реакции на сигнал */
        signal (SIGINT, SigHndlr);
        while (1);                              /*”тело программы” */
        return 0;
}
```

31. Базовые средства взаимодействия процессов в ОС UNIX. Неименованные каналы. Примеры программирования.

Неименованный канал — область памяти ВЗУ, управляемая ОС, которая осуществляет выделение взаимодействующим процессам части этой области, для совместной работы. Система ассоциирует с неименованным каналом 2 файловых дескриптора:

- Один предназначен для чтения из канала
- Другой предназначен для записи в канал
- Для таких ФД неприменимы системные вызовы перемещения указателя файла.
- Предельный размер канала декларируется параметрами ОС

Данные из канала можно читать по стратегии **FIFO** - только в той последовательности, в которой они были записаны.
Чтобы создать неименованный канал нужно использовать системный вызов **pipe()**, аргументом которого будет указатель массив из 2х элементов первый из которых содержит дескриптор чтения из канала, второй дескриптор записи в канал. СВ возвращает 0 при успехе, и заполняет массив.

Пример:
В приведенном ниже примере
производится копирование текстовой строки с использованием канала.

```
int main()
{
        char *s = “channel”;
        char buf[80];
        int pipes[2];
        pipe(pipes);                        /*Создается неименованный канал*/
        write(pipes[1], s, strlen(s) + 1);  /*В него записывается строка*/
        read(pipes[0], buf, strlen(s) + 1); /*Из канала информация записывается в buf*/
        close(pipes[0]);                    /*Закрытие ФД*/
        close(pipes[1]);
        printf(“%s\n”, buf);                /*Печать buf*/
        return 0;
}
```

32. Базовые средства взаимодействия процессов в ОС UNIX. Взаимодействие процессов по схеме ”подчиненный-главный”. Общая схема трассировки процессов.

Цель: получить возможность выделять среди процессов «Главный» процесс, который будет осуществлять управление над «подчиненными» процессами.
В ОС UNIX трассировка возможна только между родственными процессами.
Для организация модели «Главный-подчиненный» используется системный вызов **ptrace()**\
#includde <sys/ptrace.h>
int ptrace (int cmd, int pid,. Int addr, int data)

- **cmd** — код команды
- **pid** — пид процесса потомка
- **addr** — некоторый адрес в адресном пространстве потомка
- **data** — слово информации

Он позволяет читать данные из сегментов кода и данных «подчиненного» процесса, некоторые данные из его контекста, модифицировать эти данные, исполнять процесс в **пошаговом режиме** (аппаратно обеспеченный режим, вызывающий прерывание после выполнение каждой машинной команды «подчиненного» процесса).

Пример:
Рассмотрим типовую схему организации трассировки. Будем рассматривать взаимодействие родительского процесса-отладчика с подчиненным сыновним процессом.

```
...
if ((pid = fork()) == 0)
{
        ptrace(PTRACE_TRACEME, 0, 0, 0);        //Сын разрешил себя трассировать
        exec(“трассируемый процесс”, 0);        //Замена тела процесса на трассируемый
}
else                                            //Процесс отладчик
{
        wait((int ) 0);                         //Останавливается до приостановки
        for(;;)
        {
                ptrace(PTRACE_SINGLESTEP, pid, 0, 0);  //Даем сделать сыну 1 шаг
                wait((int ) 0);                        //Ждем пока сделает 1 шаг
                …                                      //Отлаживаем с помощью  ptrace
                ptrace(cmd, pid, addr, data);
                …
        }
}
```

33. Система межпроцессного взаимодействия ОС UNIX. Именование разделяемых объектов. Очереди сообщений. Пример.

IPC система обеспечивает взаимодействие произвольных процессов в пределах локальной машины. Она предоставляет взаимодействующим процессам возможность использования **общих** и **разделяемых** ресурсов.

Очередь сообщений — разделяемый ресурс, позволяющий организовывать сообщения:

- Один процесс кладет в эту очередь сообщение, а другой — читает его.
- Механизм имеет возможность блокировок, следовательно его можно использовать:
  - Как средство передачи информации между процессами
  - Как средство синхронизации процессов

Система ключей: при создании разделяемого ресурса с ним ассоциируется **ключ** — какое-то целое значение

Для генерации уникальных ключей используется функция **ftok()**

В ресурсе **очередь сообщений** каждое сообщение, кроме содержательной части имеет **тип сообщения**.

Функция **msgget()** позволяет создать очередь сообщении используя ключ и флаг, возвращает идентификатор очереди сообщений.

Функция **msgsnd()** позволяет «поставить в очередь» сообщение (определенного типа и размера)

Функция **msgrcv()** позволяет «вытащить» сообщение из очереди (определенного типа, или любое в зависимости от поданных аргументов)

Функция **msgctl** позволяет получать или изменять управляющие параметры, связанные с очередью, а также уничтожать ее с помощью IPC_RMID, принимает идентификатор очереди, команду и определенную структуру

Пример: Сначала отправляет сообщения, потом получает их, печатая на экран «bad».

Подключение заголовочных файлов опущено.

```
int main()
{
	key_t key = ftok(«file»,100);
	int msgId = msgget(key, 0666 | IPC_CREAT);
	struct
	{
		long type;
		char data[1];
	} msg;
	//Отправка сообщений
	msg.type = 1; msg.data[0] = 'a'; msgsnd(msgId, &msg, 1, 0);
	msg.type = 2; msg.data[0] = 'b'; msgsnd(msgId, &msg, 1, 0);
	msg.type = 2; msg.data[0] = 'c'; msgsnd(msgId, &msg, 1, 0);
	msg.type = 1; msg.data[0] = 'd'; msgsnd(msgId, &msg, 1, 0);
	//Получение сообщений и вывод на экран
	msgrcv(msgId, &msg, 1, 2, 0); putchar(msg.data[0]);
	msgrcv(msgId, &msg, 1, 0, 0); putchar(msg.data[0]);
	msgrcv(msgId, &msg, 1, 1, 0); putchar(msg.data[0]);
	//Удаление очереди сообщений
	msgctl(msgId, IPC_RMID, NULL);

	return 0
}
```

34. Система межпроцессного взаимодействия ОС UNIX . Именование разделяемых объектов. Разделяемая память. Пример.

IPC система обеспечивает взаимодействие произвольных процессов в пределах локальной машины. Она предоставляет взаимодействующим процессам возможность использования **общих** и **разделяемых** ресурсов.

Разделяемая память — указатель на область памяти, которая является общей для двух и более процессов. С ней можно работать как с массивом. Разделяемая память IPC почти не обладает средствами синхронизации.
Система ключей: при создании разделяемого ресурса с ним ассоциируется **ключ** — какое-то целое значение
Для генерации уникальных ключей используется функция **ftok()**

Для создания/подключчения к ресурсу разделяемой памяти IPC используются функция **shmget()** (Принимающая ключ и размер памяти, возвращаюшая идентификатор области памяти)
Разделяемую память можно присоединить к адресному пространству процесса с помощью функции **shmat()**, которая принимает идентификатор РП, адрес и флаг.
Разделяемую память можно также открепить от адресного пространства процесса, с помощью функции **shmdt()**, принимающей только адрес
Для управления разделяемой памятью используют функцию **shmctl()**, которая позволяет процссу получчать или изменять управляющие параметры связанные с РП, а также уничтожать ее с помощью команды IPC_RMID.

Пример:
В данном примере процесс:

- создает ресурс разделяемая память, размером в 100 байт с флагами,
- присоединяет ее к своему адресному пространству; при этомуказатель на начало данной области сохраняется в переменной shmaddr.
- Дальше процесспроизводит различные манипуляции
- А перед своим завершением он удаляет данную область разделяемой памяти.
- В данном примере считается, что putm() iwaitprocess() – некие пользовательские функции, определенные в другом месте.

Подключение заголовочных файлов опущено.

```
int main(int argc, char **argv)
{
        key_t key;
        char *shmaddr;
        key = ftok(“/tmp/ter”, ’S’);
        shmid = shmget(key, 100, 0666 | IPC_CREAT | IPC_EXCL);
        shmaddr = shmat(shmid, NULL, 0);
        /* работаем с разделяемой памятью, как с обычной */
        putm(shmaddr);
        waitprocess();
        shmctl(shmid, IPC_RMID, NULL);
        return 0;
}
```

35. Система межпроцессного взаимодействия ОС UNIX . Именование разделяемых объектов. Массив семафоров. Пример.

IPC система обеспечивает взаимодействие произвольных процессов в пределах локальной машины. Она предоставляет взаимодействующим процессам возможность использования **общих** и **разделяемых** ресурсов.

Массив семафоров — ресурс, представляющий собой массив из N элементов, где N задается при создании ресурса и каждый из элементов является семафором IPC.
Семафоры IPC предназначены для использованя в качестве средств синхранизации.

Система ключей: при создании разделяемого ресурса с ним ассоциируется **ключ** — какое-то целое значение
Для генерации уникальных ключей используется функция **ftok()**.

Для создания массива семафоров используется функция **semget()**, которая по ключу, количеству семафоров и флагу генерирует дескриптор созданного массива семафоров. Каждый из его элементов можно проинициализировать функцией **semctl()**. Над элементами массива семафоров определены операции (которые можно настроить самостоятельно), которые производятся с помощью функции **semop()**, собственно, совершаюшие над массивом семафоров данные операции.

Пример: Создаем семафор, инициализируем его 1, захватываем семафор с помощью функции P, выполняем защитное действие, освобождаем семафор с помощью функции V, уничтожаем семафор.

```
void P(int semid) {
   struct sembuf sb; sb.sem_num = 0;
   sb.sem_op = -1; sb.sem_flg = 0;
   semop(semid, &sb, 1);
}

void V(int semid) {
   struct sembuf sb; sb.sem_num = 0;
   sb.sem_op = 1;  sb.sem_flg = 0;
   semop(semid, &sb, 1)
}

int main() {
   key_t key = ftok("semaphore_file", „A“);
   int semid = semget(key, 1, IPC_CREAT |
0666);
   // Инициализация семафора значением 1
   semctl(semid, 0, SETVAL, 1)
   printf("Попытка захватить семафор...\n");
   P(semid); // Захват семафора
   printf("Семафор захвачен!\n");
    // Здесь можно выполнять защищенные
действия
   printf("Выполняем защищенное
действие...\n");
   sleep(2); // Имитация длительной операции
   V(semid); // Освобождаем семафор
   printf("Семафор освобожден!\n");
   // Уничтожение семафора
   semctl(semid, 0, IPC_RMID)
   printf("Семафор уничтожен.\n");
   return 0;
}
```

36. Сокеты. Коммуникационный домен. Схема работы с сокетами с установлением соединения. Пример программирования с установлением соединения.

Сокеты представляют собой обобщение и расширение механизма каналов, но с учетом особенностей, возникающих при работе с сети.
Общая схема работы с сокетами: каждый из взаимодействующих процессов на своей стороне должен создать и отконфигурировать сокет, после чего процессы должны осуществить соединение с использованием этой пары сокетов. В конце сокеты уничтожаются
Типы сокетов:

- **Соединение с использованием виртуального канала —** последовательный поток байтов, гарантирующий надежную доставку сообщений с сохранением их порядка следования.
- **Дейтаграммное соединение** — передача отдельный порций данных — **дейтаграмм**, не гарантирующая доставку, сохранение порядка следования, но работающая быстрее.

При создании сокета указывается **коммуникационный домен**, которому сокет будет принадлежать. Он определяет конкретную модель именования (форматы адресов, правила их интерпритации) и область взаимодействия процессов.
Сокет можно создать с помощью функции **socket()** принимающей домен, тип соединения и протокол. Она возвращает дескриптор сокета.
После создания необходимо связать сокет с некоторым именем(адресом).Для этого используется функция **bind().**
Сокеты с предварительным установлением соединения — организация взаимодействия, при которой адреса сокетов устанавливаются до начала передачи данных.
Сокеты типа виртуальный канал **ОБЯЗАНЫ** устанавливать соединения, дейтаграммы как правило не устанавливают.
Для установления соединения процесс-клиент использует функцию **connect()**, а процесс-сервер вызов **listen()**, сообщающую системе о готовности к обработке запросов на соединение, поступающих на данный сокет.
Конкретное соединение устанавливается системным вызовом **accept()**, извлекающим первый запрос на соединение из очереди запросов и устанавливающим соединение с ним. **Accept()** возращает новый дескриптор сокета, который будет использоваться для обмена данными, а старый продолжает обработку поступающих запросов на соединение.
Для передачи сообщений через сокеты используется функция **send()**, для приема сообщений используется функция **recv()**, для завершения работы с сокетом используется функция **shutdown()**.

37. Сокеты. Коммуникационный домен Схема работы с сокетами без установления соединения.
Пример программирования без установления соединения.

Сокеты представляют собой обобщение и расширение механизма каналов, но с учетом особенностей, возникающих при работе с сети.
Общая схема работы с сокетами: каждый из взаимодействующих процессов на своей стороне должен создать и отконфигурировать сокет, после чего процессы должны осуществить соединение с использованием этой пары сокетов. В конце сокеты уничтожаются
Типы сокетов:

- **Соединение с использованием виртуального канала —** последовательный поток байтов, гарантирующий надежную доставку сообщений с сохранением их порядка следования.
- **Дейтаграммное соединение** — передача отдельный порций данных — **дейтаграмм**, не гарантирующая доставку, сохранение порядка следования, но работающая быстрее.

При создании сокета указывается **коммуникационный домен**, которому сокет будет принадлежать. Он определяет конкретную модель именования (форматы адресов, правила их интерпритации) и область взаимодействия процессов.
Сокет можно создать с помощью функции **socket()** принимающей домен, тип соединения и протокол. Она возвращает дескриптор сокета.
После создания необходимо связать сокет с некоторым именем(адресом).Для этого используется функция bind().
Сокеты без установления соединения — организация, при которой соединение до начала передачи данных не устанавливаются, а адреса сокетов передаются каждым сообщением.
Для такой организации используются сокеты дейтаграммного типа.
Для передачи сообщений используется функция **sendto()**, а для получения **recvfrom()**, которые вместе с сообщением получают информацию об адресе отправителя/получателя, что позволяет передавать сообщения без установления соединения.
Для завершения работы с сокетом используется функция shutdown().

38. Взаимодействие нитей в ОС UNIX. Мьютексы. Пример синхронизации нитей.

Для организации межнитевого взаимодействия в ОС UNIX предоставлен механизм **Мьютексов**, организующий взаимное исключение.
Мьютекс по своему функционалу напоминает двоичный семофор, он может находится в открытом и закрытом состоянии.
Для инициализации мьютекса используется функция **pthread_mutex_init()**, инициалзирующая переменную, которая является одним из параметров функции. В таком случае для уничтожения мьютекса используется функция **pthread_mutex_destroy()**.
Мьютекс, объявленный как автоматическая переменная, можно инициализировать сразу при объявлении: pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER. В этом случае мьютекс сам уничтожится при завершении работы программы.

Операции над мьютексами:
Чтобы закрыть мьютекс используется функция **pthread_mutex_lock()**, которая:

- Если мьютекс был закрыт другой нитью, то функция будет заблокирована, пока мьютекс не откроется
- Иначе, если он был открыт, — закрывает мьютекс

Функция **pthread_mutex_trylock()**

- Закрывает мьютекс, если он был открыт
- Если он был закрыт другой нитью, то ничего не произойдет, но вернется спнциальный код ответа

Для перехода закрытого мьютекса в открытое состояние используется функция **pthread_mutex_unlock()**, которая:

- Если к ней обращется нить, закрывшая данный мьютекс — открывает ее
- Иначе возникает ошибка, поведение непредопределено.

Пример: Порождаются 10 параллельно работающих нитей, каждая из которых модифицирует общий ресурс — строку. Каждая из нитей определяет текущую длину строки и добавляет символ (с последовательно возрастающим значением, начиная с «A») в конец строки. Мьютексы реализуют взаимное исключение, при модификации нитями разделяемого ресурса.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#define N_SIZE 10
char str[N_SIZE + 1];
pthread_mutex_t mutex =
PTHREAD_MUTEX_INITIALIZER;

void* fThread(void * arg)
{
    pthread_mutex_lock(&mutex);
    int iPos = strlen(str);
    str[iPos] = 'A' + iPos;
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

```
int main()
{
    pthread_t th_id;
    int i;
    memset(str, 0, N_SIZE+1);
    for (i = 0; i<N_SIZE; i++)
    {
        pthread_create(&th_id, NULL,fThread,
NULL);
    }
    sleep(20);
    printf("%s\n", str);
    return 0;
}
```

39. Общая классификация средств взаимодействия процессов в ОС UNIX.

Рассмотрим взаимодействие процессов в рамках локальной сети

- Взаимодействие **родственных** процессов (процессов с некоторой иерархией)
  - Неименованные каналы — некоторый ресурс, наследуемый сыновьими процессами, используемый для организации между ними.
    - Симметричная модель взаимодействия
  - **Модель «Главный-подчиненный» -** среди взаимодействующих процессов можно выделить процессы, имеющие больше полномочий, контролирующие «подчиненные» процессы
- Взаимодействие **произвольных** процессов
  - **Именованные каналы** — ресурс, принадлежащий взаимодействующим процессам, посредством которого осуществляется взаимодейтсвие, с использованием имен процессов.
  - Передача **сигналов —** средство оказания воздействия одним процессом на другой.
  - Система **IPC (Массив семафоров, Разделяемая память, Очередь сообщений)** предоставляет взаимодействующим процессам общие разделяемые ресурсы
  - Аппарат **сокетов** — унифицированное средство организации взаимодейсствия. Именование осуществляется посредством связывание процесса с конкретным сокетом, через который происходит взаимодействия

Взаимодействия в пределах сети

- Аппарат **Сокетов** — базовое средство организации сетевого взаимодействия
  - Используются сокеты и взаимодействие осуществляется между ними.
- Система **MPI** — интерфейс передачи сообщений.
  - Система иллюстрирует механизм передачи сообщений.

40. Файловые системы. Структурная организация файлов. Атрибуты файлов. Основные правила работы с файлами. Типовые программные интерфейсы работы с файлами.

Файл — именованный набор данных на ВЗУ.
Файловая система(ФС) — компонент ОС, представляющий собой:

- Суктуры данных, описывающие хранящиеся на ВЗУ файлы
- Программные средства, обеспечивающие **корректный именованный доступ** к файлам

Под **корректным доступом** к данным понимается:

- Корректное управление свободным и занятым пространством ВЗУ (физическим и виртуальным)
- Защита данных от несанкционированного доступа
- Организация распределенного доступа к одному и тому же файлу нескольких пользователей

Модели структурной организации файлов:

- **В виде последовательности байт**
  - Содержимое файлов — неинтерпритируемая информация
  - Позволяет считывать и записывать произвольные порции данных
  - **Недостатки:** неэффективность
- **В виде последовательности записей (блоков) переменной или постоянной длины**
  Недостатки: внутренняя фрагментация, сложность редактирования файла в середине
- **Иерархическая организация файла**
  - Содержимое файлов — структура для динамической работы с данными файла
  - Возможность редактировать файл в середине
  - **Недостатки**: сложная реализация

Режимы доступа к файлам:

- Чтение
- Запись
- Чтение/Запись

Атрибуты файла — фиксированный набор параметров, характеризующих свойства файла и его долговременное и оперативное состояния. Состоит из:

- **Имени** — последовательности символов, для именованного доступа к файлу
- **Прав доступа** — характеристики доступа к содержимому файла различных пользователей
- **Информации о владельце** — информация о принадлежности файла пользователю
- **Тип** — информация о способе организации фалйа и интерпритации его содержимого (Регулярные файлы, файлы устройств)
- **Размер блока** (в байтах)
- **Указатель чтения и записи**
- **Время последней модификации**
- **Время последнего обращения**
- **Предельный размер файла**
- **Размещение содержимого файла в ФС** - где и как хранится файл в ФС и **т. д.**

Правила работы с файлами:

- **Открытие файла**
  - Процесс передает ФС запрос на работу с файлом
  - ФС проверяет возможность процесса работы с файлом
  - В случае успеха ФС создает **Файловы дескриптор** — системную структуру данных, содержащую информацию об актуальном состоянии открытого файла
  - При открытии файла процесс получает либо номер ФД, либо указатель на начало структуры данных
  - Последующие операции с файлом происходят при помощи ФД
- **Работа с файлом**
  - **Операции с содержимым файла** — чтение или запись
  - **Операции с атрибутами файла**—режимами доступа, указателями чтения/записи..
- **Закрытие файла**
  - ОС уведомляется о закрытии процессом ФД
  - ОС выполняет необходимые действия по завершению работы с ФД
  - При закрытии последнего открытого ФД конкретного файла, ОС осуществляет корректное завершение работы с файлом.

Типовой программный интерфейс системных вызовов для работы с файлами

- **open** — открытие
- **close** — закрытие
- **read/write** — чтение/запись относительно указателя чтения/записи
- **seek** — позиционирование указателя чтения/записи
- **read_attributes/write_attributes** — чтение/модификация аттрибутов
- **delete** — удаления файла из ФС

41.Файловые системы. Модели реализации файловых систем. Понятие индексного дескриптора.

Модели реализации файлов:

Модель непрерывных файлов — размещение каждого файла в непрерывной области внешнего устройства.
Для доступа к файлу его **атрибуты должны содержать**: имя файла, блок начала файла, длину файла.
Плюсы:

- Простота
- Минимальные накладные расходы

Минусы:

- Внутренняя фрагментация по ВЗУ
- Внешняя фрагментация ВЗУ
  - Система деградирует, приходится осуществлять процесс дефрагментации (компрессии), сдвигающий файлы друг к другу, отднако это трудоемко и времязатратно.
- Сложность увеличения размера файла
  - Решением этого служит добавление к реальному объему файла некоторое количество свободного пространство, однако это увеличивает фрагментацию

Модель связного списка - файл состоит из блоков, каждый из которых включает в себя хранимые в файле **данные** и **ссылку** на следующий блок файла.
Для доступа к файлу его атрибуты **должны** содержать: имя файла и блок начала файла
Плюсы:

- Простота
- Небольшие накладные расходы при последовательном доступе к файлу
- Отсутствие внешней фрагментации ВЗУ

Минусы:

- Неэффективность прямого доступа к файлу
- Внутренняя фрагментация по ВЗУ — накладные расходы на доступ к файлу
- Потребность в служебной информации (из-за чего размер занимаемого места увеличиться)

Модель с использованием таблицы размещения файлов (FAT — таблицы) — в оперативной памяти хранится специальная FAT- таблица:

- Количество строк таблицы совпадает с количеством блоков ФС
- В i-ой строке таблицы хранится:
  - Информация о состоянии i-го блока ФС
  - Номер строки таблицы, соответствующей следующему блока файла (которому соответсвует i-ая строка)
- Нулевая строка таблицы не соответствует никакому БЛОКУ ФС
- В специальном каталоге для каждого имени файла есть запись, содержащая номер блока ФС, соответствующего начальному блоку файла
- Для получения списка блоков ФС, хранящих файл, необходимо:
  - Определить номер начального блока
  - Последовательно обращаясь к таблице извлекать из строк номера следующих блоков
  - Закончить извлечение, наткнувшись на нулевую строку

Для доступа к файлу, его **атрибуты должны содержать**: имя файла.
А в специальном каталоге должна присутствовать запись о номере начального блока для каждого файла.

Плюсы:

- Небольшие накладные расходы при последовательном доступе к файлу
- Отсутствие служебной информации в блоках
- Возможность прямого доступа к файлу (Можно заранее получить список номеров блоков ФС, в которых хранится файл)
- Отсутствие внешней фрагментации ВЗУ

Минусы:

- Высокое потребление памяти при размещении таблицы в ОЗУ
- Внутренняя фрагментация ВЗУ
- Ограничение на размер файла (длина списков блока ФС ограничена)

Модель с использованием индексных дескрипторов .

- **Индексный дескриптор** — системная структура данных, содержащая информацию о размещении блоков конкретного файла в ФС.
- Для каждого файла ОС формирует индексный дескриптор.
- Эти структуры храняться в выделенном разделе, предназначенном для хранения параметров настройки системной информации
- При открытии файла ФС находит его ИД, и при соблюдении прав доступа, открывается сеанс работы с файлом, а его ИД перемещается в ОП.
- После этого каждый обмен происходит через информацию, содержащуюся в ИД.

Плюсы:

- Нет необходимости размещения в ОЗУ информации о всех файлах системы, достаточно хранить только информацию об открытых

Минусы:

- При увеличении размера файла совокупность размеров ИД может быть сравнима с размером FAT таблицы.
    - Решение :
        - Ограничение размеров файла
        - Иерархическая организация ИД

42. Файловые системы. Координация использования пространства внешней памяти. Квотирование пространства ФС. Надежность ФС. Проверка целостности ФС.

Координация использования пространства ВЗУ:

Выбор размера блока ФС:

При увеличении размера блока:

- Уменьшение фрагментации файлов по ВЗУ
- Увеличение внутренней фрагментации
- Эффективность обмена

При умеьшении размера блока:

- эффективное использование пространства ВЗУ
- Выссокая фрагментация данных по диску

Учет свободных блоков ФС:

Модель связных списков:

Совокупность номеров свободных блоков помещается в связный список:

- Список располагается в блоках ФС
- Первый блок со списком распологается в ОП для оперативной работы со свободными блоками
- Размер списка не является проблемой:
  - Если список большой, то много свободных блоков, значит проблем со свободными блоками из-за использования списка не возникает
  - При уменьшении числа свободных блоков, уменьшается и список, и блоки, занятые списком освобождаются

Модель битовых массивов:

Заводится битовый массив, каждый бит которого соответствует блоку ФС:

- Если блок свободен то бит равен 1, если занят то - 0
- Размер массива в битах соответствует числу блоков ФС
- Решает проблему фрагментации блоков по устройству.

Квотирование пространства ФС

Проблема: в многопользовательских системах, пользователи могут занимать весь свободный ресурс системы. Необходима регламентация использования ресурсов

Модель квотирования имеет два типа лимитов:

- **Жесткий —** количество имен в каталогах или количество блоков ФС, которое пользователь превзойти не может.
  - Если происходит превышение жесткого лимита, работа пользователя в системе блокируется
- **Гибкий —** значение, которое устанавливается в виде лимита. С ним ассоциируется **счетчик предупреждений**.
  - При входе пользователя в систему происходит подсчет соответствующего ресурса
  - Если вычисленное значение не превышает гибкий лимит, то счетчик предупреждений сбрасывается на начальное значени и пользователь работает.
  - Если превышает лимит, то значение счетчика уменьшается на единицу.
  - При равенстве счетчика 0, вход пользователя блокируется
  - Иначе пользователь получает предупреждение, после чего продолжает работу.

Надежность ФС

Резервное копирование — способ минимизации потерь данных при сбоях ФС

- Резервные копии следует хранить распределенно
- Регулярное полное копирование ФС неэффективно и ресурсоемко
  - Решение: **минимизация копий**
  - Решение: **Копирование на ходу**, однако во время копирования пользователь может изменить файл

        - Решение: копирование при отсутствии пользователя в системе
        - Решение: расписание копирования

Стратеггии копирования:

- **Физическое копирование** — поблочное копирование данных
    - Проблема: копируются и занятые и свободные блоки
    - Решение: **интеллектуальное физическое копирование** — копируются только занятые блоки
- **Логическое копирование** — копирование пофайлово.

Стратегии минимизации копий:

Избирательное копирование — файлы, которые могут быть восстановлены — не копируются

- Исполняемые файлы при условиях:
    - При наличии готовых копий
    - При наличии исходного кода
- Определенные файлы пользователей определенных категорий

Инкрементное архивирование — создание цепочки архивов

- Первое архивирование — создание полной копии всех файлов - **мастер копии**
- Последующие архивирования — копирования только тех файлов, которые созданы или изменены с момента предыдущего архивирование.

Компрессия — сжатие данных при архивировании

Плюсы:

- Уменьшается объем резервной копии

Минусы:

- Чувствительность к потере информации

Проверка целостности ФС

Для выявления сбоев целостности ФС используется избыточная информация.

Аспект целостности ФС — непротиворечивость свободных и занятых блоков.

- Формируетсся две таблицы: занятых и свободных блоков размера в количестве блоков ФС.
- Проводится анализ блоков ФС: для каждого свободного блока увеличивается на 1 соответствующая запись в таблице свободных блоков
- Проводится анализ ИД: для каждого встреченного в ИД блока увеличивается на 1 соответствующая запись в таблице занятых блоков
- Производится сопоставление содержимого таблиц и коррекция ошибок: S, Z
    - Если S+Z = 1, то ошибки нет
    - Если S = Z = 0, то блок потерян из списка свободных
    - Если S > 1, Z = 0 , то нарушено хранение свободных блоков
    - Если S = 0, Z > 1 — блок принадлежит нескольким файлам, для исправления ошибки определяются файлы с ИД, указывающими на блок ФС с этим номером, создаются копии конфликтующих файлов, исходные файлы удаляются, копии файлов переименовываются в исходные, пересоздается таблица занятых блоков

Аспект целостности ФС — соответствие количества жесктих ссылок на файл с соответствующим количеством жестких ссылок в атрибутах этого файла

При несоответствии происходит коррекция аттрибутов

43. Примеры реализаций файловых систем. Организация файловой системы ОС UNIX. Виды файлов. Права доступа. Логическая структура каталогов.

Индексный дескриптор файла в ОС UNIX — хранящаяся на ВЗУ структура данных, содержащая информацию о файле за исключением его имени

Файл в ОС UNIX — совокупность ИД и блоков данных ФС, в которых расположен файл

- Пустой файл имеет только ИД
- Имя файла не является частью файла в UNIX

Жесткая ссылка — имя файла, ссылающаяся на некоторый ИД

- При создании файла создается оригинальная жесткая ссылка
- Несколько имен файла могут быть связаны с одним ИД
- ИД содержит счетчик жестких ссылок

Типы файлов:

- **Регулярный файл**
- **Каталог** — файл содержит имена и ссылки на аттрибуты, файлов, содержащихся в нем
- **Файл устройств** — специальный файл, которому поставлено в соответствие определенное устройство
- **Именнованный канал** (FIFO-файл) — файл процессорного взаимодействия
- **Символическая ссылка** — специальный файл, содержащий имя другого файла
- **Сокет** -файл предназначенный для распределенного взаимодействия

Пользователь — объект, имеющий **учетную запись в системе** с ним связан UID.

Группа — объект объединяющий несколько пользователей, с ним связан GID.

- У каждого пользователя 1 **основная группа** и несколько **доп** групп.

Суперпользователь — пользователь с UID = 0, на которого не действуют разграничения полномочий.

Разграничения полномочий — функция ОС, заключающаяся в проверке прав процесса на системный вызов

- Процессы имеют права на те или иные системные вызовы
- Разные процессы имеют разные права

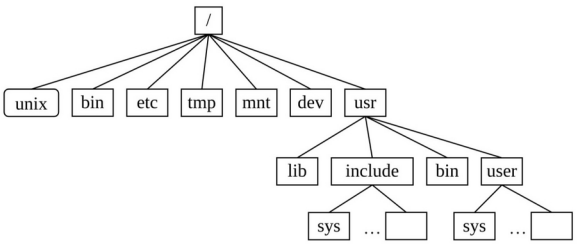
Права доступа к файлу — характеристики файла, регламентирующие какие операции с файлами могут совершать процессы пользователей:

- По категориям пользователей: Владелец файла, члены основной группы владельца файла, все пользователи системы
- В каждой группе три вида прав доступа: чтение, запись, исполнение

Структура каталогов в UNIX

В UNIX — корневая ФС

- Корневой каталог - /
- Имена файлов образуют дерево



Рекомендованная иерархия каталогов:

/bin — исполняемые файлы команд
/etc — системные таблицы и команды работы с ними
/tmp — каталог временных файлов
/mnt — каталог монтировки
/dev — каталог специальных файлов устройств
/usr — каталог пользовательской информации
/usr/lib — инструменты работы пользователей
/usr/include — каталог заголовчных файлов для СИ-компилятора
/usr/bin — каталог команд пользователя
/usr/user — домашние каталоги пользователей системы

44.Примеры реализаций файловых систем Внутренняя организация ФС. Модель версии UNIX System V.

Структура файловой системы System V:
ФС занимает часть **системного раздела**.
Эта часть состоит из трех подпространств:

- **Суперблок** — содержит тип ФС, размер ФС в блоках, размер области ИД, число свободных блоков, число свободных ИД, флаги, размер блока, список номеров свободных ИД, список адресов свободных ИД.
- **Область индексных дескрипторов**
- **Блоки файлов**

Массив номеров свободных блоков — первый элемент списка свободных блоков, располагающийся в суперблоке.
Каждый элемент списка хранит номер свободного блока, а также номер следующего элемента списка. Все последующие элементы списка хранятся в области блоков файлов.

- При запросе свободного блока происходит поиск номеров свободных блоков с ячейки ненулевой информации
  - Если такая есть, то ячейка обнуляется и выдается номер свободного блока
  - Если такая не найдено, то содержимое следующего элемента списка загружается в массив номеров свободных блоков и номер блока в котором размещался освободившийся элемент возвращается ФС как ответ на запрос.
- При освобождении блока происходят противоположные действия

Массив номеров свободных ИД — заполнен номерами свободных ИД

- При освобождении ИД:
  - Если в массиве есть свободное место, то в массив записывается номер освободившегося ИД в первое свободное место
  - Иначе номер ИД забывается
- При создании файла:
  - Если массив не пуст, то из него изымается первый !=0 номер свободного ИД
  - Иначе, если в суперблоке есть информация о наличии свободных блоков, то система обновляет массив, и выдает номер свободного ИД

Индексный дескриптор — систеемная структура данных, содержащая аттрибуты файла, а также всю оперативную информацию об организации и размещении данных.
Построено **взаимооднозначное соответствие** между содержимым файла и его ИД.
Структурная организация блоков файла:

- Массив номеров блоков файла состоит из 13 элементов
- Первые 10 — номера первых десяти блоков файла
- 11-ый элемент — ссылка на массив из N номеров блоков файла
- 12-ый элемент — ссыка на массив из N ссылок на массив из N номеров блоков файла
- 13-ый элемент — трехуровневая косвенная адресация N^3 блоков

Каталог ФС System V — файл специального типа, содержимое которого находится в области блоков файла. Его структурная организация:

- Каждая запись в нем — фиксированного размера
- Первые несколько байт хранят номер ИД файла
- Оставшиеся байты хранят имя файла
- При создании каталога он получает 2 предопределенные записи, которые невозможно удалить или изменить: «.» - ссылка на текущий каталог, «..» -ссылка на родительский каталог

Плюсы ФС System V:	**Минусы ФС System V:**
-Иерархичность системы -Удачное решение организации блоков файло -Оптимизированная работа с массивом СБ и ИД	-В суперблоке — ключевая информация о ФС -Ограничение на длину имени файла -Фрагментация блоков по диску, приводит к изнашиванию головки -Частые обращения к суперблоку — поломка.

45. Примеры реализаций файловых систем. Внутренняя организация ФС. Принципы организации файловой системы FFS UNIX BSD.

Раздел представлен как последовательность дисковых цилиндров, разбитую на порции фиксированного размера. В каждом из полученных кластеров размещается :

- Копия суперблока
- Блоки файлов
- Информация об ИД, ассоциированных с данным кластером
- Информация о свободных ресурсах данного кластера

Оптимизация размещения каталога

При создании каталога систем осуществляет поиск наименее загруженного ИД кластера. Среди таких кластеров выбирается кластер с наименьшим числом каталогов.

Равномерность использования блоков данных

При создании файла он делится на несколько частей. Часть файла, ассоциированная с прямой адресацией из ИД, по возможности размещается в том же кластере, что и ИД. Оставшиеся части делятся на равные порции и ФС размещает их в отдельных кластерах.
(Это решение проблемы **фрагментации файла по разделу**)

Размещение последовательности блоков файлов.

Размещение последовательности блоков с небольшим отступом.
(Решение проблемы возникновения **лишних оборотов диска**, при чтении из блока)

Внутрення организация блоков

В ФС FFS увеличены размеры блока, за счет чего: уменьшается фрагментация файла по диску, уменьшаются накладные расходы при чтении подряд идущих данных файла.
Однако увеличивается внутренняя фрагментация.
Решение: Деление каждого блока на фиксированное число **фрагментов.**
Размещение файла по блокам ФС:

- Считаем, что все блоки файла (кроме последнего) целиком заполнены содержимым данного файла и хранятся в аттрибутах файла
- Номер последнего блока, а также информация о занятых в нем фрагментах хранятся в аттрибутах данного файла.
- При необходимости выделения пространства другому файлу, он может быть расположен в том же блоке что и данные, но в других, ранее свободных фрагментах этого блока.

Структура каталога FFS

- Каталог FFS позволяет описывать имена файлов длины не более 255 символов.
- Состоит из записей переменной длины.
- Первая запись — номер ИД, размер записи и длину имени файла.
- При удалении файла из каталога освободившееся пространство, занимаемое записью присоединяется к предыдущей записи.
- Удаление первой записи выражается в обнулении номера ИД в этой записи

Предопределенные номера ИД

0 — несуществующий ИД	3 — квоты пользователей
1 — список поврежденных блоков	4 — групповые квоты
2 — корневой каталог	8 — журнал

Алгоритм доступа к содержимому файла (на примере имени «/dir/filename»)

1) Считываем ИД №2 из области ИД
2) Считываем содержимое этого файла каталога согласно секции адресации содержимого файла в данном ИД
3) Перебираем элементы до момента пока не обнаружится запись с именем dir(ИД_А)
4-6) Считываем ИД_А, содержимое этого файла каталога и перебираем записи пока не находим filename (ИД_Б)
7-8)Считываем ИД_Б и содержимое целевого файла согласно секции адр. Сод. Файла в ИД

46. Управление внешними устройствами. Архитектура организации управления внешними устройствами, основные подходы, характеристики.

Организация управления ВУ:

- **Непосредственное управление** ЦП ВУ
  - Компьютер на уровне макрокоманд полностью обеспечивает все действия по управлению ВУ. Поток управления и данных идет через ЦП.
  - Невозможность ЦП вычислять при обмене.
- **Синхронное управление** ВУ
  - **Специализированные контроллеры** — устройства, располагающиеся между ЦП и ВУ.
  - Они позволяли ЦП работать с более высокоуровневыми операциями, при управлении ВУ
  - Невозможность ЦП вычислять при обмене.
- **Асинхронное управление** ВУ — появилась благодаря аппарату прерываний
  - Позволяла запускать обмен для одного процесса, поставив на счет другую задачу, а по окончании обмена в системе возникнет прерывание, сигнализирующее о возможности дальнейшего выполнения вычислительного процесса.
- Управление ВУ с использованием **DMA-контроллеров**
  - ЦП генерирует последовательность управляющих команд, указывая координаты в ОП, куда надо положить или откуда взять данные
  - DMA-контроллер перемещает данные между ОЗУ и внешним устройством.
- Управление ВУ с использованием **процессора ввода-вывода**
  - ЦП оперирует функционально-емкими макрокомандами
  - Решение всех задач осуществления непосредственного обмена — задача специализированного процессора

Иерархия архитектуры программного управлениия ВУ:

1. Аппаратура
2. Программы обработки прерываний
3. Драйверы физических устройств
4. Драйверы виртуальных устройств

Цели программного управления ВУ:

- **Унификация программных интерфейсов доступа к ВУ** — абстрагирование от аппаратных характеристик обмена.
- **Обеспечение конкретной модели синхронизации при выполнении обмена**
- **Выявление и локализация ошибок** и **устраниния их последствий** — система должна быть организована таким образом, чтобы она могла выявить момент появления сбоя и обработать эту ситуацию.
- **Буферизация обмена** — передача данных ВУ — медленная, буферизация позволяет временно хранить данные и обеспечить более плавный поток информации.
- **Обеспечение стратегии доступа к устройству**
  - Организация управления этим устройством и синхронизации
  - Организация распределенного доступа к устройству
  - Организация монопольного доступа к устройству
- **Планирование выполнения операций обмена**
  - Неправильная организация очереди заказов на обмен может привести к деградацию системы.

47. Управление внешними устройствами. Планирование дисковых обменов, основные алгоритмы.

Стратегии организации дисковых обменов:
Рассмотрим дисковое устройство, обмен с которыми осуществляется дорожками.
Имеется очередь запросов к следующим дорожкам: 4,40,11,35,7,14
Головка дискового устройства изначально находится на 15-ой дорожке.

- Стратегия **FIFO**.
  - Простая, но не эффективная в случаях, когда последовательность запросов с большим разбросом.
  - 15-4-40-11-35-7-14
- Стратегия **LIFO**.
  - В целом аналогично, однако она достаточно полезная, при поступлении цепочки связных обменов.
  - Проблема с появлениями новых запросов.
  - 15-14-7-35-11-40-4
- Стратегия SSTF, «жадный» алгоритм: на каждом шаге выбирается дорожка, требуюшая минимального перемещения головки.
  - Появляется проблема голодания крайних дорожек.
  - 15-14-11-7-4-35-40
- Алгоритм **PRI**: каждому процессу присваивается приоритет, тогда очередь заказов на обмен обрабатывается согласно приоритетам.
  - Проблема корректной выдачи приоритета, иначе возникает голодания низкоприоритетных процессов.
- Алгоритм **SCAN**: головка диска перемещается сначала в одну сторону (до предельной границы диска), выбирая каждый раз из очереди запрос с номером обозреваемой головкой дорожки, а затем в другую.
  - Заранее известно число запросов на перемещение (не более 2*число дорожек)
  - Приводит к деградации при интенсивном обмене с некоторой локальной областью.
  - 15-35-40-14-11-7-4
- Алгоритм **C-SCAN**: в очереди запросов ищется номер с минимальным номером, головка передвигается до него и за один проход по диску обрабатывает всю очередь запросов
  - Проблемы те же что и у SCAN
  - 15-4-7-11-14-35-40
- **N-step-SCAN**: очередь делится неким образом на N очередей, затем по какой либо стратегии выбирается очередь, которая будет обрабатываться. Во время обработки блокируется попадание новых запросов в эту подочередь. Сама обработка очереди может осуществляться по алгоритму **SCAN**
  - Решение проблемы «голодания»

48. Управление внешними устройствами. Организация RAID систем, основные решения, характеристики.

RAID-система - совокупность физических дисковых устройств, которая представляется в ОС как одно устройство, имеющее возможность организации параллельных обменов. Требуется избыточная информация, используемая для контроля и восстановления информации, хранимой на этих дисках.

- На разных устройствах, составляющих RAID-массив, размещаются **полосы** — порции данных фиксированного размера, которым осуществляется обмен.

Далее представлены все **RAID-устройства** , а на следующей странице — их схемы

RAID 0

- Полоса соизмерима с дисковыми блоками, соседние полосы — на разных устройствах
- С каждым диском обмен происходит параллельно, засчет чего доступ к информации быстрый.
- Устройства независимые
- Отсутствует хранение избыточной информации, засчет чего система может хранить объем информации равный суммарной емкости дисков

RAID 1 (система зеркалирования)

- Два комплекта дисков.
- При записи информации она сохраняется на соответствующем диске и на диске-дублере.
- При чтении информации запрос направляется лишь одному из дисков
- Обращение к диску дублеру происходит при утере информации из первого.
- Половина всей хранящейся информации — избыточна.

RAID 2

- Модель с синхронизированными головками, то есть совокупность дисков — синхронизирована.
- Содержит избыточную информацию, восстанавливающую данные в случае поломки одного из устройств.
- Для обнаружения и исправления ошибок используются коды Хэмминга
- Исправляет одинарные ошибки и выявляет двойные

RAID 3

- Такая же как RAID 2, но использует другую стратегию обнаружения ошибок.
- Один из дисков назначается для хранения избыточной информации — полос, дополняющих до четности соответствующие полосы на других дисках (в каждой позиции суммарное число единиц на всех дисках должно быть четным)
- При сбое одного из устройств можно восстановить утерянные данные

RAID 4

- `Является упрощением RAID 3, при котором все устройства несинхронизированы.
- Для поддержания корректности диска четности необходимо каждый раз производить пересчет.

RAID 5

- Служебная информация разносится по разным дискам
  - Изнашиваемость не накапливается на одном диске, а распределяется по всем.
- Избыточная информация распределяется по дискам циклическим образом

RAID 6

- Аналогично 5, но избыточная информация распределена по дискам циклически двухуровневым образом.

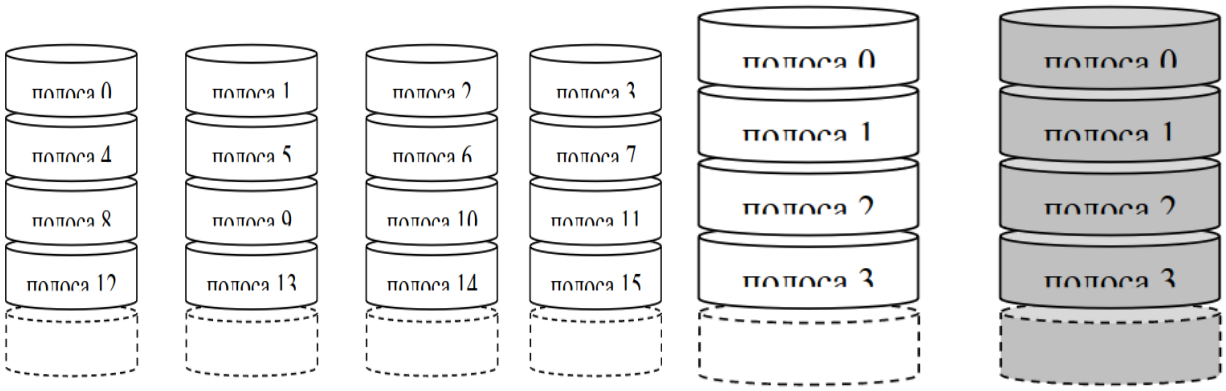


Рис. 151. RAID 0.

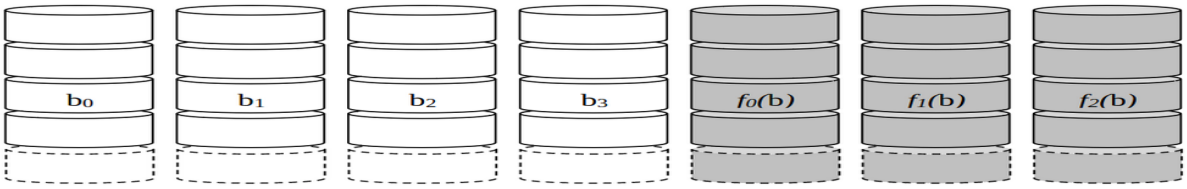


Рис. 153. RAID 2. Избыточность с кодами Хэмминга (Hamming, исправляет одинарные и выявляет двойные ошибки).

В данном случае имеется 4 диска данных и 1 диск четности. Тогда для диска четности:
$X_4(i)=X_0(i)xorX_1(i)xorX_2(i)xorX_3(i)$
Если произойдет потеря данных на первом диске, то для восстановления достаточно воспользоваться формулой:
$X_1(i)=X_0(i)xorX_2(i)xorX_3(i)xorX_4(i)$

Рис. 154. RAID 3. Четность с чередующимися битами.

В данном случае имеется 4 диска данных и 1 диск четности. Тогда для диска четности:
$X_4(i)=X_0(i)xorX_1(i)xorX_2(i)xorX_3(i)$

После обновления полосы на первом диске:
$X_{4\ new}(i)=X_4(i)xorX_1(i)xorX_{1\ new}(i)$

Рис. 155. RAID 4.

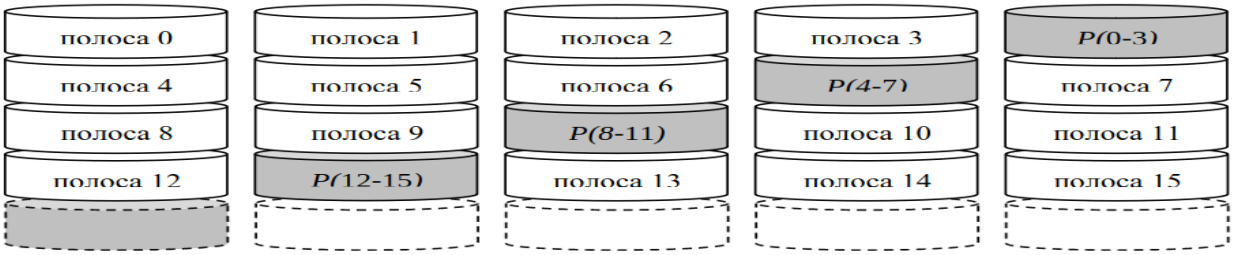


Рис. 156. RAID 5. Распределенная четность (циклическое распределение четности).

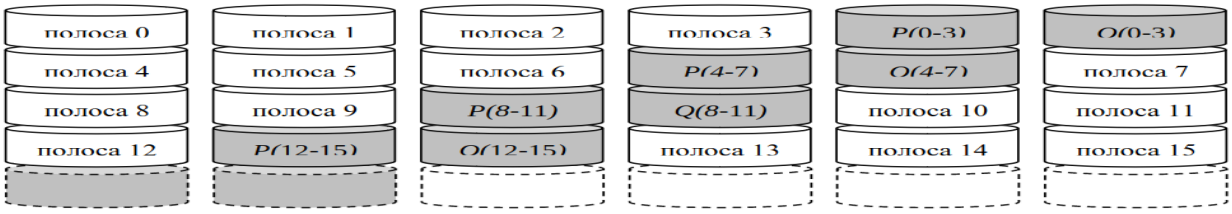


Рис. 157. RAID 6. Двойная избыточность (циклическое распределение четности с использованием двух схем контроля; требуется N+2 диска).

49. Внешние устройства в ОС UNIX. Типы устройств, файлы устройств, драйверы.

Типы ВУ:

- **Блок-ориентированные** — ВУ, обмен с которыми осуществляется порциями данных фиксированной длины — **блоками**.
- **Байт-ориентированные** — все остальные внешние устройства
  - Таймер — байт ориентированное устройства, но обменов в нем нет.

Файлы устройств обеспечивают решение задач:

- Именования устройств (точнее - именования **драйверов** устройств).
- Связывание имени с конкретным драйвером.

ИД файла устройств в качестве атрибута содержит:

- Информацию о том блок-ориентированное или байт-ориентированное устройство
- Поля **старший номер** и **младший номер**
  - Старший номер — номер драйвера в системной таблице драйверов
  - Младший номер — некоторая дополнительная информация, передаваемая драйверу при обращении

Системные таблицы драйверов

Для регистрации драйверов в системе используются 2 таблицы:

- **bdesvw** — для блок-ориентированных устройств
- **cdevsw** — для байт ориентированных устройств.
- Каждая запись этих таблиц содержит структуру, называемую **коммутатором устройства**. Он хранит указатели на всевозможные точки входа (функции) в соответсствующий драйвер. Коммутатор также может хранить ссылку на точку ядра.

Ситуации вызывающие обращения к функциям драйвера:

- Старт системы, определение ядром состава доступных устройств.
- Обработка запросов на обмен
- Обработка прерывания связанного с данным устройством
- Выполнения специальных команд управления устройством.

Изначально в ОС UNIX драйверы были жестко встроены в код ядра. Переписывать код ядра для каждого нового драйвера — неэффективно.

Возникло динамическое связывания драйверов.

- Именование драйвера
- Инициализация драйвера
- Добавления данного драйвера в таблицу драйверов устройств
- Установка обработчика прерывание — предоставление ядру информации.

50. Внешние устройства в ОС UNIX. Системная организация обмена с файлами. Буферизация обменов с блокоориентированными устройствами.

Системная организация обмена с файлами

Для организации обмена в ОС UNIX используются системные таблицы и структуры.

Таблица открытых файлов (ТОФ)

- Создается системой на ресурсах процесса.
- Каждая запись этой таблицы соответствует открытому в процессе файлу.
- ФД — номер записи в ТОФ.
- Размер таблицы определятся параметрами ОС
- Каждая запись ТОФ содержит много аттрибутов, один из которых — ссылка на номер записи в **таблице файлов** (ТФ) ОС.

Таблица файлов — системная таблица ОС

- В этой таблице регистрируются все открытые файлы
- В записях этой таблицы содержатся:
  - Указатель чтения/записи
  - **Счетчик кратности**
  - Ссылка на таблицу ИД открытых файлов

Таблица ИД открытых файлов (ТИДОФ) — системная структура данных, содержащая перечень ИД всех файл, открытых в системе в данный момент.

- Каждая запись содержит актуальную копию открытого в системе ИД
- Каждая запись хранит целый набор параметров, один из них **счетчик кратности**

Если в системе происходит сбой, то содержимое ТИДОФ будет потеряно, то есть будут потери и в ФС.

Буферизация обменов с блок-ориентированными устройствами

Для организации блок-ориентированных обменов система использует стандартную стратегию КЭШирования. Цель — минимизация обменов с ВУ.

КЭШ организован в терминах блоков и расположен в ОП ОС.

Минус:

- Кэширование дисковых обменов приводит к несоответствию реального содержимого диска и того содержимого, которое должно быть на нем. При сбое системы — потеря информации.

51. Управление оперативной памятью. Одиночное непрерывное распределение. Распределение разделами. Распределение перемещаемыми разделами.

Стратегии организации ОЗУ, а также методы управления ею:

Одиночное непрерывное распределение:

- Все дресное пространство разделяется на 2 компонента.
- В первой части распологается и функционирует ОС
- Во второй части выделяются ресурсы для выполнения прикладных процессов
- Если пользовательском режиме происходит попытка обратиться к области памяти ОС, то возникает прерывание

Необходимые аппаратные требования:

- Один регистр границ

Алгоритмы: просты и очевидны

Плюсы:

- Простота концепта
- Минимальные аппаратные требования

Минусы:

- Неэффективное использование физического ресурса
  - Часть памяти, выделенной под процесс, реально никогда не используется
- Бессмысленность выделения процессу всю память на все его время выполнения, хотя процесс работает лишь с локальными участками памяти
- Модель ограничивает размер процесса.

Распределение разделами:

- Все адресное пространство делится на две части
- Первая отводится под ОС
- Вторая отводится под работу прикладных процессов, причем оно разбивается на N **разделов**, кадлая из которых имеет произвольный фиксированный размер.
- Организовывается очередь прикладных процессов, распределения процессов по этим разделам

Два варианта организации очереди:

- Сквозная очередь, которая по каким то соображениям распределяет процессы между этими разделами
- Ассоциация с каждым разделом своей очереди, поступающий процесс сразу поступает на одну из очередей

Необходимы аппаратные требования:

- Регистр границ — для определения области (ОС или прикладных программ)
- Два регистра границ
  - Один отвечает за начало области процесса, второй за конец.

Алгоритмы:

- Для модели с N очередями входная очередь сортируется и процесс размещается в разделе минимального размера, достаточного для размещения процесса
  - Не гарантируется равномерная загрузка всех очередей
- Для модели со сквозной очередью из очереди выбирают первый процесс, помещающийся в освободившемся разделе
  - Дискриминация больших процессов маленькими
- Для модели со сквозной очередью из очереди выбирается процесс с максимальным размером, помещающийся в освободившийся раздел
  - Дискриминация маленьких процессов большими
  - Решение: Заводится **счетчик дискриминации**, который уменьшается на 1 при каждом освобождении раздела, достаточного для загрузки данного процесса, но при этом процесс был пропущен.

    - При просмотре очереди, в случае если счетчик дискриминации равен 0 и процесс помещается в освободившийся раздел, то он загружается в него.

Плюсы:

- Простота аппаратных средств мультипрограммирования
- Простота алгоритмов

Минусы:

- Внутренняя фрагментация в разделах
- Ограничение предельного размера прикладных процесссов размером максимального физизческого раздела ОЗУ
- Весь процесс размещается в памяти, неэффективное использование ОЗУ

Распределение перемещаемыми разделами

Модифицируем предыдущую организацию использованием **компрессии**, то есть сдвига исполняемого кода ближе друг к друга, для освобождения свободного места.

Подходы реализации компрессии:

- **Локальная** — сдвиг небольшого числа процессов для высвобождения нужного пространства системе.
- Приостановка выполнения всех процессов и сдвиг их исполняемого кода к началу ОЗУ, тогда в конце ОЗУ будет расположена свободная память.

Необходимые аппаратные требования:

- Аппарат защиты памяти (регистры границ)
- Аппаратные средства, позволяющие перемещать процессы (регистр базы)

Алгоритмы: простые, схожие с предыдущей организацией

Плюсы:

- Отсутствие фрагментации памяти

Минусы:

- Ограничение предельного размера прикладного процесса размером физ памяти
- Накладные расходы, связанные с компрессией.

52. Управление оперативной памятью. Страничное распределение.

Страничное распределение:
Адресное пространство представляется в виде совокупности блоков фиксированного размера — **страниц**.
Виртуальное адресное пространство — пространство, с адресами которого работает программа.
Физическое адресное пространство — пространство, которое есть в наличии в компьютере.

При страничном распределении необходимы программные средства, позволяющие устанавливает соответствие между виртуальными и физическими страницами.

Реализация аппаратной таблицы страниц:

- **Проблемы:**
  - Большой объем таблицы страни
  - Увеличение стоимости аппаратной поддержки
  - Необходимость полной перезагрузки таблицы при смене контекстов
- **Вывод**: такой способ подходит только при небольшом количестве страниц.

Хранение таблицы страниц в ОЗУ

- **Аппаратные требования**:
  - Наличие регистра, ссылающегося на начало таблицы в ОЗУ
  - Обращение в ОП, извлечение данных и осуществление преобразований по адресу, указанному в регистре должно поддерживаться аппаратно.
- **Проблема**: при каждом преобразовании данных происходит обращение к ОЗУ, что совсем неэффективно.
- Возможна оптимизация подхода, за счет использования КЭШей L1 и L2
  - Появляются проблемы «зависания» таблицы в КЭШе, в силу частых обращений к ней
- Хранение таблицы в ОЗУ, в силу огромного числа процессов, - дорогое решение
- При хранении в ОЗУ только оперативной части таблицы возникает проблема подкачки страниц таблицы.

Алгоритмы:
(!)Использование **иерархической организации** таблицы страниц
Структура записи таблицы:

- Номер физической таблицы
- Совокупность атрибутов:
  - Атрибут присутствия/отсутствия данной страницы
  - Аттрибут режима защиты страницы
  - Флаг модификации содержимого страницы
  - Аттрибут частоты обращений к страниице
  - Аттрибут блокировки КЭШирования и др.

(!) Использование **TLB-таблицы** — аппаратно реализованной таблицы, относительно небольшого размера.

- Первый столбец - номер виртуальной страницы.
- Второй столбец — номер физической страницы, в которой находится виртуальная.
- Третий столбец — содержит различные атрибуты.

Виртуальный адрес состоит из:

- Номера виртуальной страницы
- Смещения в ней

Алгоритм преобразования адреса из виртуального в физический:

- Система изымает из этого адреса номер виртуальной страниицы.
- Осуществляет оптимизированный(параллельный) поиск по TLB-таблице.
- Если искомый номер найден, то система автоматически осуществляет проверку соответствия атрибутов.
- Если проверка пройдена успешно, то происходит подмена номера виртуальной страницы номером физической страницы, получаем физический адрес
- Если при поиске номер виртуальной страницы не найден:
  - Система обращается в программную таблицу
  - Удаляет самую старую запись из TLB
  - Загружает в нее найденную запись из программной таблицы
  - Вычисляет физический адрес

Проблемы:

- Большой размер таблицы страниц (НЕ ПУТАТЬ С TLB)
- При смене контекста система обязана поменять таблицу страниц, а также содержимое TLB
- Необходимость решения того, где размещать все таблицы различных процессов.

Иерархическая организация таблицы страниц

- Информация о странице представляется не в виде одного номера страницы, а в виде совокупности номеров, используя которые можно получить номер соответствующей физической страницы посредством обращения к соответствующим таблицам, участвующим в иерархии
- Всю таблицу страниц хранить в памяти необязательно — в силу принципа локализации будет достаточно хранить только небольшую «внешнюю» таблицу групп страниц, а все необходимые страницы нижестоящих уровней подкачивать по необходимости

Решение основанное на Хэшировании:

- Базируются на использовании **хеш-функций**, которые решают следующую задачу:
  - Пусть имеется некоторое множество значений, которое необходимо каким то образом отобразить на множество фиксированного размера
  - Для осуществления такого отображения используют функцию, которая по входному значению определяет номер позиции.
  - При ее использовании возможны коллизиии.
- Из виртуального адреса аппаратно извлекается номер страницы
- Он подается на вход некоторой Хеш-функции, отображающей его значение на аппаратную таблицу (**Хеш-таблицу)** фиксированного размера.
- Каждая запись в данной таблицу хранит начало списка коллизий, где каждый элемент:
  - Номер виртуальной страницы, номер соответствующей ему физической страницы
- Перебирая список коллизий можно найти номер исходной виртуальной страницы и соответствующий ему номер физической страницы
- Из **недостатков**: проблемы с перемещением списков коллизий

Инвертированные таблицы страниц

- Виртуальный адрес трактуется как тройка значений:
  - PID процесса
  - Номер виртуальной страницы
  - Смещение в этой странице
- Каждая строка инвертированной таблицы страниц, соответствует физической странице (с номером равному номеру этой строки)
- Каждая запись данной таблицы содержит информацию о:
  - PIDe процесса
  - О соответствующем номере виртуальной страницы

- Имея пару PID процесса и номер виртуальной страницы, производится поиск по таблице страниц
- Смещение найденного результата — номер физической страницы.

Необходимые аппаратные требования:

- Все предыдущие
- Возможность процессора на аппаратном уровне работать с идентификаторами процессов.

Плюсы:

- Наличие единственной таблицы страниц, обновление которой при смене контекстов-нетрудоемкое: обновляются только те строки таблицы, для ктоорых происходилась загрузка процесса.
- Нет ограничения на размер процесса.

Алгоритмы выбора откачиваемых страниц:

- **NRU** — с любой страницей ассоциируются 2 признака:
  - M — признак — отвечающий за модификацию страницы
  - R — признак — отвечающий за обращение на чтение или запись к странице
  - Изначально они равны нулю
  - По таймеру происходит программное обнуление R-признака
  - При выборе страницы для откачки система определяет нужную следующим образом:
  - **Класс 0**: R=M=0 — самые непопулярные страницы
  - **Класс 1**: R=0,M=1 — в последний период не было обращений, но были изменения
  - **Класс 2**: R=1,M=0 — недавно читалась информация
  - **Класс 3**: R=1, M=1 — популярные страницы
  - Система выбирает для откачки страницу случайным способом из непустого класса минимального номера
- Алгоритм **FIFO**
  - ОС фиксирует время загрузки при загрузке очередной страницы в память
  - При выборе страницы на откачку выбирает ту, что дольше всего распологается в ОЗУ
  - Возможна откачка интенсивно используемой страницы
- Модификация **FIFO**
  - Выбирается старая страница
  - Если R = 0, то страница откачивается
  - Если R = 1, то страница обнуляется и перемещается в конец очереди (время загрузки данной страницы переопределяется текущим временем)
  - Рост накладных расходов при перемещении страниц по очереди
- Алгоритм **«Часы»**
  - Все страницы образуют циклический список
  - Имеется маркер, ссылающийся на некоторую страницу в списке и перемещается «по кругу»
  - Если R = 0 в точке маркера, то остраница выгружается, загружается новая
  - Маркер сдвигается на следующую страницу
  - Если R = 1, то признак обнуляется, а маркер сдвигаетсся на следующую позицию
- Алгоритм **LRU**
  - Пусть имеется N страниц. Для решения задачи в системе имеется битовая матрица размера N*N, которая изначально — нулевая
  - После обращения к i-ой странице — все биты i-ой строки увеличиваются на 1, а весь i-ый столбец — обнуляется
  - При выборе страницы на откачку выбирается страница, для которой строка матрицы хранит наименьшее двоичное число.

53. Управление оперативной памятью. Сегментное распределение. Сегментно-страничное распределение.

Сегментное распределение

- Каждый процесс совокупность сегментов, имеющих свой размер и свою функциональность:
  - Сегмент кода
  - Сегмент данных
  - Сегмент стека и т. д.
- Для организации работы с сегментами может использоваться таблица в которой хранится информация о каждом сегменте
- Виртуальный адрес — номер сегмента и величина смещения в нем
- Организована аппаратная таблица сегментов с фиксированным числом записей
- Каждая запись соответствует своему сегменту и хранит информацию о:
  - Размере сегмента
  - Адресе его начала
  - Различные атрибуты
- Имея виртуальный адрес система аппаратно извлекает из него номер сегмента i
- Обращается к i-ой строке таблицы и извлекает из нее информацию о сегменте
- Проверяет, не превосходит ли величина сегмента размер самого сегмента
- Если превосходит — происходит прерывание
- Иначе физический адрес = адрес начала сегмента + смещение

Плюсы:

- Простота организации

Минусы:

- Каждый сегмент должен целиком размещаться в ОЗУ
- Проблемы с откачкой/подкачкой:
  - Осуществляются всем процессом или целым сегментом, что неэффективно
- Ограничение на предельный размер сегмента.

Сегментно страничное распределение

- Виртуальный адрес — номер сегмента и смещение в нем
- Реализована аппаратная таблица сегментов, посредством которой из виртуального адреса получается **линейный** адрес
- Он в свою очередь представлен в виде номера страницы и величины смещения в ней.
- Используя таблицу страниц, он преобразовывается в физический адрес.
- В процессе имеется ряд виртуальных сегментов, которые дробятся на страницы.

Плюсы:

- Нет ограничения на предельный размер сегмента
- Процессу выделены различные диапазоны виртуальных адресов, в результате уходим от проблемы пересечения стека и кучи.

54. Вычислительная система. Кэширование информационных потоков на уровнях аппаратуры и ОС.

Кэш — высокоскоростная структура данных, призванная повысить эффективность обработки информационных потоков, через хранение часто используемых фрагементов памяти в быстром доступе

Три основных подхода к организации отображения блоков ОП в КЭШ:

- Полностью ассоциативный КЭШ
  - Блок ОП может быть размещен в любой позиции КЭШ
- КЭШ прямого отображения
  - Каждый блок ОП размещается в единственной позиции в КЭШе, заданной формулой : $A_K = A_O \bmod N_K$ (адрес блока в КЭШ, адрес блока в ОП, числов блоков в КЭШ)
- Частично ассоциативный КЭШ:
  - Множество блоков кэш разделяется на несколько непересекающихся подмножеств и блок ОП может быть отображен в любой блок КЭШ строго заданного множества по аналогичной формуле.

Алгоритмы вытеснения

- Случайный выбор вытесняемого блока
- **LRU** — вытеснение блока, неиспользумого дольше всех
- **LFU** — вытеснение наиболее редкого блока
- **MRU** — вытеснение последнего использованного блока и т. д.

Программные методы повышение эффективности КЭШ

- Объединение массивов и объединение циклов

Аппаратные методы повышения эффективности КЭШ

- Увеличение размера блока памяти
  - Понижается вероятность промаха
  - Уменьшение времени доступа к КЭШ
  - Возрастают накладные расходы на вытеснение
- Повышение ассоциативности КЭШ
- Не блокирующий при промахе КЭШ
- Увеличение числа уровней КЭШ
  - Понижение стоимости промаха

55. Язык программирования С. Общая характеристика. Типы, данные, классы памяти. Правила видимости. Структура программы. Препроцессор. Интерфейс с ОС UNIX.

Общая характеристика

Язык С был разработан в начале 1970-х годов и стал основой для многих современных языков. Он характеризуется низкоуровневым доступом к памяти, что позволяет прогрммистам эффективно управлять ресурсами.

Типы данных

С предоставляет встроенные типы данных:

- Целочисленные: char, short,int, long, unsigned int и т.д
- Вещественные: float double, long double

С поддерживает создание пользовательских типов данных с помощью:

- структур - struct
- объединений - union
- перечислений — enum

Классы памяти

- **Статическая**:
  - Используется для глобальных переменных и статических локальных переменных
  - Переменные инициализируется и существуют до завершения программы.
- **Автоматическая**:
  - Используется для локальных переменных внутри функций и блоков.
  - Эти переменные создаются при входе в функцию или блок и уничтожаются при выходе из нее
- **Динамическая**:
  - Выделяется при обращении к функциям malloc,calloc,realloc и освобождается с помощью free.

Правила видимости

- **Глобальная видимость**:
  - Глобальные переменные доступны из любой функции в программе
- **Локальная видимость**:
  - Локальные переменные доступны только внутри функции или блока, где они были определены

Структура программы

Программа на языке С состоит из:

- **Директивы предпроцессора**: начинаются с символа #
- **Функции main()**: точка входа в программу. Каждая программа должна содержать эту функцию
- **Объявления переменных** : могут быть размещены в начале функции или в глобальной области
- **Определения функций**: пользовательские функции могут определены до или после функции main()

Препроцессор — программа, обрабатывающая исходны код перед компиляцией. Она выполняет следующие задачи:

- Подключение заголовочных файлов
- Совершает макросную подстановку

Интерфейс с ОС UNIX

Язык С тесно интегрирован с данной ОС. Основные аспекты интерфейса:

- Системные вызовы — позволяют взаимодействовать с ядром ОС, для выполнения операций в привилегированном режиме
- Стандартные библиотеки: предоставляют функции для работы с ФС и процессами.
- Потоки ввода вывода: стандартные функции для работа с консолью и файлами