Билет 1. Этапы развития вычислительной техники и программного обеспечения.

Вычислительную технику исторически разделяют на поколения. **Поколение** – это группа компьютеров, которые объединены по совпадению определенного набора признаков, таких как архитектура, элементная база, области применения и т. д.

Первое поколение компьютеров. 1940 — 1950. Элементная база: электронно-вакуумные лампы. Область применения: задачи обороны. Для ввода/вывода и в качестве запоминающего устройства использовались перфоленты.

Особенности: Однопользовательский, персональный режим, зарождение класса сервисных, управляющих программ, зарождение языков программирования (язык Assembler, программный транслятор из Assembler в машинные коды).

Проблемы: В случае возникновения ситуаций типа деления на нуль компьютер останавливался. Изменять программу также было очень тяжело, так как машинные коды завязаны на адресацию.

Второе поколение компьютеров. 1950 - 1960. Элементная база: полупроводниковые приборы, диоды и транзисторы. Область применения: управление бизнес-процессами. Размер компьютеров второго поколения на порядки уменьшился, уменьшилось энергопотребление, уменьшились габариты, увеличилась скорость.

Особенности: Пакетная обработка заданий, мультипрограммирование, языки управления заданиями (язык, который позволял бы до начала работы программы сформировать требования, которые необходимы для ее выполнения: максимальное время счета, необходимый объем ОП, объем памяти на магнитной ленте), файловые системы (возможность именовать данные и сохранять их. Можно не знать конкретного местоположения), виртуальные устройства (как результат обобщения особенностей управления внешними устройствами), операционные системы.

Проблемы: У каждого внешнего устройства своя управляющая программа, идентичные устройства от разных производителей не взаимозаменялись.

Третье поколение компьютеров. 1960 — 1970. Элементная база: интегральные схемы. Область применения: универсальные, но для продвинутых пользователей.

Особенности: Унификация устройств (использование идентичных расходных материалов), унификация аппаратных интерфейсов, создание семейств компьютеров (т. к. появились задачи, требующие компьютер с определенной архитектурой; стала возможна модернизация компьютера), развитие операционных систем (появление ОС UNIX).

Четвертое поколение компьютеров. 1970 — настоящее время. Элементная база: большие и сверхбольшие интегральные схемы. Область применения: универсальные.

Особенности: Дружественность интерфейсов, развитие сетевых технологий, важнейшая задача - обеспечение безопасности передачи и хранения информации.

Билет 2. Структура вычислительной системы. Ресурсы ВС. Уровень операционной системы.

Вычислительная система (ВС) – совокупность аппаратных и программных средств, функционирующих в единой системе и предназначенных для решения задач определенного класса.
Структура: 1. Аппаратные средства. 2. Управление физическими ресурсами. 3. Управление логическими ресурсами. 4. Системы программирования. 5. Прикладные системы.

Аппаратный уровень определяется наборами аппаратных компонентов и их характеристиками: правила программного использования, производительность или емкость, степень занятости или используемости.
Средства программирования, доступные на аппаратном уровне: система команд компьютера, аппаратные интерфейсы программного взаимодействия с физическими ресурсами.

Управление физическими ресурсами. Задача — стандартизация правил программного использования физических ресурсов, представление унифицированного интерфейса для программного использования.
Драйвер физического устройства – программа, основанная на использовании команд управления конкретного физического устройства и предназначенная для организации работы с данным устройством.
Решает задачи предоставления упрощённого интерфейса для доступа к данному физическому ресурсу, сокрытия некоторых деталей от пользователя.

Управление логическими ресурсами. Задача — не модифицировать программу для работы с устройством другого типа.
Логическое устройство - устройство, некоторые эксплуатационные характеристики которого, (возможно все) реализованы программно.
Драйвер логического устройства: программа, обеспечивающая существование и использование соответствующего ресурса.
Может применяться многоуровневая унификация интерфейса, иерархия драйверов.

Ресурсы вычислительной системы — совокупность всех физических и виртуальных ресурсов системы.
Операционная система – это комплекс программ, обеспечивающий управление ресурсами вычислительной системы. Пользователю доступна система команд ОС.
Уровень ОС — уровень управления ресурсами ВС. Одна из характеристик ресурсов ВС - конечность, т. е. возникает конкуренция. ОС должна решать задачу распределения ресурсов между программами.

Билет 3. Структура ВС. Ресурсы ВС. Уровень систем программирования.

Система программирования – это комплекс программ, обеспечивающий поддержание жизненного цикла программы в вычислительной системе.

Жизненный цикл программы в ВС состоит из четырех этапов:

Проектирование. *Объектная ВС* — ВС, на которой предполагается работа программного комплекса. *Инструментальная ВС* — ВС, на которой будет вестись разработка программного комплекса. Этот этап должен учитывать: характеристики объектной и инструментальной ВС, модель функционирования программного комплекса, алгоритмы и инструментальные средства, используемые при разработке. Результатом этапа проектирования является спецификация на создаваемое программное решение.

Кодирование (программная реализация). Построение кода при использовании языков программирования и трансляторов. Могут применяться системы поддержки версий, они фиксируют реализацию продукта в данный момент времени.

Тестирование и отладка. *Тестирование* — процесс проверки правильности работы программы на заранее определенных наборах входных данных. Возникает задача определения минимального набора тестов, наиболее полно проверяющих программу. *Отладка* - процесс поиска, локализации и исправления зафиксированных при тестировании ошибок.

Ввод в эксплуатацию. *Внедрение* — установка и первичная настройка программного комплекса на объектную ВС. *Сопровождение* - исправление недочетов внедрения и проектирования программного комплекса.

Модели разработки программных систем:

Каскадная. Все по порядку. Преимущества: детерминированность времени и затрат. Недостатки: возможно устаревание к моменту реализации.

Каскадно-итерационная. Возможен возврат на предыдущие этапы. Преимущества: в идеале максимальное удовлетворение заказчика. Недостатки: Недетерминированность затрат.

Спиральная. «По кругу». Наиболее современная и перспективная модель. Преимущества: Детерминированность времени выполнения каждой итерации, возможность рассмотрения системы до завершения разработки. Недостатки: Недетерминированность времени и затрат на конечный продукт.

На уровне систем программирования доступны программные средства, обеспечивающие поддержание жизненного цикла программы.

Билет 4. Структура ВС. Ресурсы ВС. Уровень прикладных систем.

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

Этапы развития прикладных систем:

1. Задача → Разработка, программирование → Решение.

2. Развитие систем программирования и появление средств создания и использования библиотек программ.

3. Этап характеризуется появлением пакетов прикладных программ, имеющих развитые и стандартизированные интерфейсы. Возможность совместного использования различных пакетов.

Примеры: MS office, Mathcad, …

Основные тенденции в развитии современных прикладных систем: Открытость систем, стандартизация моделей автоматизируемых бизнес-процессов(B2B, B2C, Enterprise resource planning, Customer relationship management).

Категории пользователей: Оператор или прикладной пользователь (доступны средства пользовательского интерфейса), системный программист (пользователь компонентов прикладной системы), системный администратор.

Билет 5. Структура ВС. Понятие виртуальной машины.

Мы можем сделать некий срез уровня любой вычислительной системы, основываясь на иерархии и классификации по уровням. Например, мы можем рассматривать только аппаратный уровень, или только уровень операционной системы. На каждом из этих уровней мы встретимся с понятием «виртуальной машины». Дело в том, что мы никогда не можем работать просто с «компьютером». Каждый раз нам приходится использовать некую программную прослойку между нами и машиной, будь то ассемблер или Windows 95.

Совокупность программных средств, обеспечивающих в любой момент времени нашу связь с компьютером, назовем **виртуальной машиной**. Виртуальная машина всегда разная. Например, если мы работаем с DOS, то наша виртуальная машина обладает следующими характеристиками: во-первых, она имеет систему команд ДОС, то есть в то время, как физически для нашего компьютера определена система команд низкого уровня, наша виртуальная машина обладает системой команд, которые включают в себя команды «dir» или «cd». Виртуальная машина DOS способна выполнять только одну задачу в один момент времени, она не предназначена для мультипрограммирования, хотя на деле мы можем работать за многопроцессорной рабочей станцией. С другой стороны, виртуальная машина Windows имеет больший объем оперативной памяти (речь идет о подкачке), по сравнению с компьютером, на котором она установлена. То есть можно сказать, что виртуальная машина никак или практически никак не связана с физической, за исключением, конечно же, того, что виртуальная машина в любом случае вынуждена использовать физическую.

Рассмотрим виртуальные машины по уровням.

Начнем с *уровня физических ресурсов*. Пусть у нас есть жесткий диск и драйвер этого диска. В этом случае драйвер представляет собой виртуальную машину, ведь если подумать, драйвер никак не связан с диском, драйвер можно скопировать на дискету и унести от диска. Но драйвер, с другой стороны, и есть для пользователя диск, поскольку именно драйвер – это то, что позволяет использовать диск. Таким образом, можно сказать, что без драйвера диск - не диск. Виртуальная машина здесь – это программа, представляющая собой лишь одну часть аппаратного обеспечения.

Уровень логических ресурсов. Пусть жесткий диск поделен на два логических раздела. В этом случае, интерфейс каждого из разделов – отдельная виртуальная машина. Каждый из логических дисков имеет меньше памяти, чем весь диск в целом, то есть представленные виртуальные машины обладают меньшим количеством ресурсов по сравнению с физическими характеристиками данного компьютера.

Уровень систем программирования. СП дают возможность создать виртуальную машину, имеющую определенный набор команд. Например, компилятор gcc позволяет эмулировать компьютер, чья система команд определена стандартом ANSI C.

Билет 6. Основы архитектуры компьютера. Основные компоненты и характеристики. Структура и функционирование ЦП.

ЦП обеспечивает выполнение программы, размещенной в ОЗУ. Осуществляется выбор машинного слова, содержащего очередную команду, дешифрация команды, контроль корректности данных, определение исполнительных адресов операндов, получение значений операндов и исполнение машинной команды.

ЦП состоит из: кэш памяти первого уровня, управляющее устройство, арифметико-логическое устройство, регистровая память.
Регистровая память – совокупность устройств памяти ЦП, предназначенных для временного хранения операндов, информации, результатов операций. Сверхоперативное запоминающее устройство.
Устройство управления – координирует выполнение команд программы процессором.
Арифметико-логическое устройство – обеспечивает выполнение команд, предусматривающих арифметическую или логическую обработку операндов.

Рабочий цикл процессора – последовательность действий, происходящая в процессоре во время выполнения программы.
РЦ (в регистре счетчика команд находится адрес первой команды программы):
1. по содержимому счетчика команд выбирается машинное слово, в котором находится очередная команда.
2. счетчик команд увеличиваем на 1.
3. анализируем код операции команды. Если код операции команды некорректный, то происходит прерывание.
4. Если код операции корректный, то происходит анализ типа этой операции.
5. Вычисление адресов операндов.
6. Выбираются значения операндов, которые помещаются в арифметико-логическое устройство.
7. Выполнение команды. Если выполняемая команда есть команда передачи управления, то она передается в устройство управления и, если условия выполняются, то вычисляется исполнительный адрес (адрес перехода), который заносится в счетчик команд. Переход на начало, к шагу 1.

Регистровая память делится на: *регистры общего назначения* (используются для организации индексирования и определения исполнительных адресов операндов, а также для хранения значений наиболее часто используемых операндов, в этом случае сокращается число реальных обращений в ОЗУ и повышается системная производительность ЭВМ), *специальные регистры* (качественный и количественный состав зависит от архитектуры).
Специальные регистры:
Регистр адреса (РА) - содержит адрес команды, которая исполняется в данный момент времени.
Регистр результата (РР) - содержит код, характеризующий результат выполнения последней арифметико-логической команды. Содержимое РР используется для организации ветвлений в программах, а также для программного контроля результатов.
Слово – состояние процессора (ССП или PSW) - регистр, содержащий текущие «настройки» работы процессора и его состояние.
Регистр указатель стека - в данном регистре размещается адрес вершины стека. Содержимое изменяется автоматически при выполнении «стековых» команд ЦП.

Билет 7. Основы архитектуры компьютера. ОЗУ. Расслоение памяти.

В ОЗУ размещается исполняемая в данный момент программа и используемые ею данные. ОЗУ состоит из ячеек памяти, содержащей поле машинного слова и поле служебной информации.
Машинное слово – поле программно изменяемой информации, в машинном слове могут располагаться машинные команды или данные, с которыми может оперировать программа. Машинное слово имеет фиксированный для данной ЭВМ размер.
Служебная информация (ТЭГ) – поле ячейки памяти, в котором схемами контроля процессора и ОЗУ автоматически размещается информация, необходимая для осуществления контроля над целостностью и корректностью использования данных, размещаемых в машинном слове (разряды контроля четности, разряды контроля данные-команды, машинный тип данных).
В ОЗУ все ячейки имеют уникальные адреса. Доступ к машинному слову осуществляется по адресу.

Скорость доступа к данным ОЗУ ниже скорости обработки данных в ЦП. Необходимо, чтобы итоговая скорость выполнения команды процессором как можно меньше зависела от скорости доступа к коду команды и к используемым в ней операндам из памяти. Эта проблема решается аппаратно.

Расслоение ОЗУ - один из путей аппаратного решения проблемы дисбаланса между скоростью доступа к данным, находящимся в ОП, и производительностью процесса.

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

Физически ОЗУ представимо в виде объединения k устройств, способных хранить одинаковое количество информации и способных взаимодействовать с процессором независимо друг от друга. При этом адресное пространство ВС организовано таким образом, что подряд идущие адреса, или ячейки памяти, находятся в соседних устройствах (блоках) оперативной памяти. Программа состоит (в большей степени) из линейных участков. Если использовать этот параллелизм, то можно организовать в процессоре еще один буфер, который организован так же, но в котором размещаются машинные команды. За счет того, что есть параллельно работающие устройства, то этот буфер автоматически заполняется вперед. Т.е. за одно обращение можно прочесть k машинных слов и разместить их в этом буфере. Далее, действия с буфером команд похожи на действия с буфером чтения/записи. Когда нужна очередная команда (ее адрес находится в счетчике команд), происходит ее поиск (по адресу) в буфере, и если такая команда есть, то она считывается. Если такой команды нет, то опять-таки работает внутренний алгоритм выталкивания строки, новая строка считывается из памяти и копируется в буфер команд. При последовательном чтении машинных слов по последовательным адресам время чтения уменьшается с времени цикла до времени доступа.

Билет 8. Основы архитектуры компьютера. Кэширование ОЗУ.

Регистровые буфера или КЭШ память предназначены для разрешения проблемы несоответствия скоростей работы ОЗУ и ЦП, на аппаратном уровне. Следует отметить, что результат этой оптимизации, в общем случае зависит от характеристик программы. Традиционно, в развитых ЭВМ используется аппаратная буферизация доступа к операндам команд, а также к самим командам.

Буфер операндов – аппаратная таблица, логически являющаяся компонентом ЦП, призванная аппаратно минимизировать количество обращений к «медленному» ОЗУ при записи и чтении операндов. Таблица состоит из фиксированного числа строк. Каждая строка имеет следующие поля:

- *адрес* – физический адрес машинного слова в ОЗУ;
- *значение* – значение машинного слова, соответствующего адресу;
- *признак изменения* – код, характеризующий факт изменения поля значения (в соответствующей ячейке ОЗУ значение отличается от значения в таблице);
- *код старения* – код, характеризующий интенсивность обращений к данной строке. По значению поля определяются наиболее «популярные» строки. Конкретный алгоритм изменения данного поля зависит от ЭВМ.

Алгоритм для чтения данных из ОЗУ

Пусть имеется команды чтения данных из по физическому адресу Аисп.

1. Поиск по таблице строки, содержащей адрес, совпадающий с Аисп. Если такой строки нет, то на п. 3.
2. Происходит обновление кода старения. Результатом команды чтения является содержимое поля «Значение».
3. По значениям поля «Код старения» осуществляется поиск строки, используемой наименее интенсивно.
4. Анализируется код изменения. Если значение изменялось в таблице, то происходит запись значения по адресу в ОЗУ.
5. Считывается машинное слово из ОЗУ по адресу Аисп и заполняется данная строка.
6. Считанное значение является результатом выполнения команды чтения.

Алгоритм для записи данных в ОЗУ

1. Поиск по таблице строки, содержащей адрес, совпадающий с Аисп. Если такой строки нет, то на п. 3.
2. Значение записывается в поле «Значение». Происходит обновление полей «Признак изменения», «Код старения». Выполнения команды записи завершено.
3. По значению поля «Код старения» осуществляется поиск строки, используемой наименее интенсивно.
4. Анализируется код изменения. Если значение изменялось в таблице, то происходит запись значения по адресу в ОЗУ.
5. Происходит обновление содержимого полей строки в соответствии с командой записи. Выполнение команды завершено.

Буфер команд – минимизация обращений в ОЗУ за машинными командами. Поля таблицы: адрес, значение, код старения.

Наибольший эффект достигается при небольших циклах, когда все операнды размещаются в буфере, и после этого циклический процесс работает без обращений к ОЗУ.

Билет 9. Основы архитектуры компьютера. Аппарат прерываний. Последовательность действий в вычислительной системе при обработке прерываний.

Аппарат прерываний ЭВМ - возможность аппаратуры ЭВМ стандартным образом обрабатывать возникающие в вычислительной системе события. *Прерывание* - одно из событий в вычислительной системе, на возникновение которого предусмотрена стандартная реакция аппаратуры ЭВМ. Количество различных типов прерываний ограничено и определяется при разработке аппаратуры ЭВМ.
Прерывания можно разделить на две группы:
Внутренние прерывания инициируются схемами контроля работы процессора. Это может быть реакция ЦП на программную ошибку. Например, деление на ноль.
Внешние прерывания – это средство, позволяющее ЭВМ корректно взаимодействовать с внешними устройствами. Это может быть событие, связанное с поступлением новой информации от ВУ или возникновение ошибки во ВУ.

Первый этап обработки прерываний (аппаратный):

1. Включается режим блокировки прерываний. Запрещается инициализация новых прерываний.
2. Малое упрятывание. Копирование в специальную регистровую память ЦП минимального количества регистров и настроек ЦП, достаточных для запуска программы ОС, обрабатывающей прерывания. Это заведомо счетчик команд, регистр результата, некоторое количество регистров общего назначения.
3. Переход на программный режим обработки. Аппаратуре известна точка входа в обработчик прерываний.

Второй этап (программный). Управление передано на точку ОС, занимающуюся обработкой прерывания. При входе в эту точку часть ресурсов ЦП, используемых программами освобождена (результат малого упрятывания). Поэтому будет запущена программа ОС, которая может использовать только освобожденные малым упрятыванием ресурсы ЦП (перечень доступных в этот момент ресурсов – характеристика аппаратуры). Выполняется следующая последовательность действий:

1. Анализ и предварительная обработка прерывания. Происходит идентификация типа прерывания, определяются причины. Если прерывание "короткое" обработка не требует дополнительных ресурсов ЦП и времени, то прерывание обрабатывается, происходит разблокировка прерываний и возврат в первоначальную программу (эта последовательность действий организована так, что гарантируется корректное восстановление всех регистров и настроек ЦП). Если прерывание требует использования всех ресурсов ЦП, то переходим к следующему шагу.
2. Осуществляется полное упрятывание состояния всех ресурсов ЦП, использовавшихся прерванной программой (все регистры, настройки, режимы и т.д.) в специальную программную таблицу. После данного шага программе обработки прерываний становятся доступны все ресурсы ЦП, а прерванная программа получает статус ожидания завершения обработки прерывания.
3. Отключение режима блокировки прерываний.
4. Завершение обработки прерывания.

Билет 10. Основы архитектуры компьютера. Внешние устройства. Организация управления и потоков данных при обмене с внешними устройствами.

Внешние устройства: ВЗУ, устройства ввода и отображения информации, устройства приема и передачи данных.

ВЗУ: последовательного доступа (чтобы добраться до определённой записи, нужно пройти все предыдущие; пример: магнитная лента), прямого доступа:
Магнитные диски. Для задания координат определенного сектора в управляющее устройства необходимо передать: номер цилиндра, где расположен сектор, номер дорожки, на которой находится сектор, номер сектора. Магнитный барабан. SSD.

Организация потоков данных. Через ЦП: при получении данных с внешнего устройства они попадают на специальные регистры ЦП и потом в ОЗУ. Обмен с использованием прямого доступа к памяти.

Организация управления внешними устройствами. Непосредственное управление внешними устройствами центральным процессором, синхронное управление ВУ с использованием контроллеров внешних устройств, асинхронное управление ВУ с использованием контроллеров, использование контроллера прямого доступа к памяти.

Пусть в системе имеется прерывание «обращение к системе» и «завершение обмена с устройством».

Синхронная работа с ВУ.
Программа будет приостановлена с момента обращения к ВУ до момента завершения обмена. Дисбаланс между скоростью работы ВУ и скоростью выполнения машинных команд колоссальный, поэтому задержки при синхронной работе велики.

Асинхронная работа с ВУ.
Программа инициирует прерывание «обращение к системе», происходит обработка прерывания, при которой драйверу устройства передается заказ на выполнение обмена. После завершения обработки прерывания программа продолжает выполнение до завершения обмена. Выполнение программы приостанавливается, так как возникает прерывание «завершение обмена с устройством». После обработки этого прерывания выполнение программы будет продолжено. Проблема дисбаланса решена.

Эта схема упрощенная. Она не затрагивает случаев синхронизации доступа к областям памяти, участвующим в обмене. Проблема состоит в том, что, например, записывая некую область данных на ВЗУ, после обработки заказа на обмен, но до завершения обмена, программа может попытаться обновить содержимое области, что является некорректным. Поэтому в реальных системах для синхронизации работы с областями памяти, находящимися в обмене, используется возможность ее аппаратного закрытия на чтение и/или запись. То есть при попытке обмена с закрытой областью памяти произойдет прерывание. Это позволяет остановить выполнение программы до завершения обмена, если программа попытается выполнить некорректные операции с областью памяти, находящейся в обмене.

Билет 11. Основы архитектуры компьютера. Иерархия памяти.

Данная иерархия строится с позиций близости к ЦП, стоимости памяти и системной составляющей. Элементами памяти в ЦП являются **регистры общего назначения и КЭШ 1-го уровня**.

Следующий уровень - это **уровень устройства, которое называется КЭШ 2-го уровня**. Оно находится между ЦП и ОЗУ, т.е. обычно это устройство, которое быстрее ОП, но может быть медленнее и дешевле КЭШа 1-го уровня, а также может обладать немножко большими размерами, чем КЭШ 1-го уровня, соответственного схема работы с КЭШем 2-го уровня аналогична схеме работы с КЭШем 1-го уровня.

По иерархии уровень после уровня ОЗУ – это уровень **внешнего запоминающего устройство с внутренней КЭШ-буферизацией**. Т.е. это устройства, аппаратное управление которых имеет КЭШ буферизацию. Менее эффективно, чем ОП, но достаточно эффективно, потому что опять-таки за счет внутреннего кэширования (при той же схеме кэширования, которая имеет место в схеме ЦП - ОЗУ), сокращается реальное количество обращений к устройству и тем самым получается существенное повышение производительности работы устройства.

Следующий уровень - **внешнее запоминающее устройство прямого доступа без КЭШ-буферизации**. Это устройства существенно менее эффективные, но также предназначенные для оперативного доступа к данным, т.е. это устройства, которые обычно используются в программе для организации внешнего хранения и доступа за данными, соответственно по производительности они могут быть разными, но для каких-то ситуаций категории этих двух устройств не принципиальны.

Последним уровнем этой иерархии является уровень **внешнего запоминающего устройства долговременного хранения данных**. Т.е. это устройства, предназначенные для архивирования и долговременного хранения информации, к этим устройствам могут относиться и как устройства прямого доступа, и устройства последовательного доступа.

Суть иерархии: на вершине находятся самые высокоскоростные, которые, в свою очередь являются также и самыми дорогими устройствами, но спускаясь вниз, мы получаем устройства менее дорогие, но обладающие худшими показателями по скорости доступа. Предусматриваются достаточно большие элементы сглаживания дисбаланса в производительности каждого из типов этих устройств.

Билет 12. Аппаратная поддержка ОС. Мультипрограммный режим.

Несмотря на возможность асинхронной работы с ВУ, имеют место периоды ожидания программой завершения обмена. Если система обрабатывает единственную программу, то в это время ЦП не производит никакой полезной работы, то есть простаивает (на самом деле термин простой достаточно условный, так как при этом работает операционная система). Решением проблемы простоя ЦП в этом случае является использование ВС в **мультипрограммном режиме**, в режиме при котором возможна организация переключения выполнения с одной программы на другую.

Для корректной организации мультипрограммной обработки необходима аппаратная поддержка ЭВМ. Как минимум аппаратура ЭВМ должна поддерживать следующие функции:
1. **Аппарат защиты памяти**. Аппаратная возможность ассоциирования некоторых областей ОЗУ с одним из выполняющихся процессов/программ. Настройка аппарата защиты памяти происходит аппаратно, то есть назначение программе/процессу области памяти происходит программно (т.е., в общем случае операционная система устанавливает соответствующую информацию в специальных регистрах), а контроль за доступом – автоматически. При этом при попытке другим процессом/программой обратиться к этим областям ОЗУ происходит прерывание “Защита памяти”.
2. Наличие специального режима **операционной системы (привилегированный режимом или режим супервизора)**. Суть заключается в следующем: все множество машинных команд разбивается на 2 группы. Первая группа – команды, которые могут исполняться всегда (пользовательские команды). Вторая группа – команды, которые могут исполняться только в том случае, если ЦП работает в режиме ОС. Если ЦП работает в режиме пользователя, то попытка выполнения специализированной команды вызовет прерывание – "Запрещенная команда”. Какова необходимость наличия такого режима выполнения команд? Простой пример – управление аппаратом защиты памяти. Для корректного функционирования этого аппарата необходимо обеспечить централизованный доступ к командам настройки аппарата защиты памяти. То есть эта возможность должна быть доступна не всем программам.
3. Необходимо наличие **аппарата прерываний**. Как минимум в машине должно быть прерывание по таймеру, что позволит избежать “зависания“ всей системы при зацикливании одной из программ.

Билет 13. Аппаратная поддержка ОС. Организация регистровой памяти ЦП (регистровые окна, стек).

Один из способов решения проблемы вложенных процедур – *регистровые окна*. В компьютере имеется k физических регистров. Система команд машины предоставляет l регистров общего назначения, l различных регистровых окна. Каждый из l регистров отображается на k физических регистров. В каждый момент времени программа работает с одним регистровым окном.

Аппарат позволяет привязать регистровые окна к множеству физических регистров. Окна перемещаются дискретно. Окно состоит из трех частей: входные, выходные регистры; локальные (внутренние) регистры. Аппаратура обеспечивает существование фиксированного количества окон. Окна расположены циклическим образом. Этот аппарат позволяет активизировать вложенные программы. При вызове подпрограммы происходит переключение текущего регистрового окно на следующее регистровое окно, при этом возможно пересечение 3-тей части текущего окна с первой частью последующего окна. Этим достигается, во-первых, практически автоматическая передача и прием параметров, во-вторых, всегда создается новый комплект локальных регистров, которые присутствуют в программе.

Рассмотрим, что происходит при непосредственной работе:
Обращаемся к 1-й программе, ей выделяется 0-е регистровое окно. Дальше пошли в глубину на 2-й уровень, выделилось 1-е регистровое окно и т.д. до тех пор, пока не дошли до последнего. Что будет, когда этот круг обойдем? Начинается откачка этих окон в ОП. Эта схема гарантирует эффективную работу программ с вложенностью не более фиксированного, если вложенность больше, то возникают проблемы, но все равно начинается работа с КЭШем и мы все равно не опускаемся на уровень общения с ОП. Соответственно система может иметь специальный регистр-указатель текущего окна и указатель сохраненного окна.

При обращении у функции: Увеличиваем указатель текущего окна на единичку по модулю L. Сравниваем: (указатель на новое содержимое текущего окна) = (указатель на сохраненное окно), если равны, то мы дошли до ситуации, в которой пытаемся обратиться за окном, которое уже занято, т.е. пошли по второму кругу этого цикла. Происходит прерывание. Мы откачиваем в память текущее окно, после этого меняем указатель на сохраненное окно и используем освобожденное текущее окно, так новое. Если не равно, то используем окно, на которое указывает cwp.

При выходе из функции: Уменьшаем cwp на 1 по модулю L. Сравниваем cwp с swp. Если он равны, то это означает, что мы сохраняли это окно. Происходит прерывание, мы восстанавливаем это окно, мы соответственно уменьшаем этот указатель и продолжаем выполнение, если не равны, это означает, что это окно у нас не сохранялось, и мы просто продолжаем выполнение.

Использование *системного стека* может частично решать проблему минимизации накладных расходов при смене обрабатываемой программы и/или обработке прерываний. Частично стек реализуется на регистрах, таким образом, существенно ускоряется работа.

Билет 14. Аппаратная поддержка ОС. Виртуальная оперативная память.

В результате трансляции программы получаем исполняемый модуль. Данный модуль представляет собой готовую к выполнению программу в машинных кодах. При этом внутри программы к моменту образования исполняемого модуля используется модель организации адресного пространства программы (эта модель, в общем случае не связана с теми ресурсами ОЗУ, которые предполагается использовать позднее). Для простоты будем считать, что данная модель представляет собой непрерывный фрагмент адресного пространства, в пределах которого размещены данные и команды программы. Будем называть подобную организацию адресации в **программе программной адресацией или логической/виртуальной адресацией**.

Возникает задача сопоставления виртуальным адресам физических. Элементарное программно-аппаратное решение – использование возможности ***базирования адресов***. Суть его состоит в следующем: пусть имеется исполняемый программный модуль со своим виртуальным адресным пространством. В ЭВМ выделяется специальный регистр базирования, который содержит физический адрес начала области памяти, в которой будет размещен код данного исполняемого модуля. При этом исполняемые адреса, используемые в модуле, будут автоматически преобразовываться в адреса физического размещения данных путем их сложения с регистром базирования. Таким образом, код используемого модуля может перемещаться по пространству физического ОЗУ. Эта схема является элементарным решением организации простейшего ***аппарата виртуальной памяти***.

Пусть имеется ВС, функционирующая в мультипрограммном режиме. При размещении новых программ/процессов в ОЗУ ЭВМ (для их мультипрограммной обработки) образуются свободные фрагменты ОЗУ между программами/процессами. Суммарный объем свободных фрагментов может быть достаточно большим, но, в то же время, размер самого большого свободного фрагмента недостаточно для размещения в нем новой программы/процесса. В этой ситуации возможна деградация системы – в системе имеются незанятые ресурсы ОЗУ, но они не могут быть использованы. Путь решения этой проблемы – использование более развитых механизмов организации ОЗУ и виртуальной памяти, позволяющие отображать виртуальное адресное пространство программы/процесса не в одну непрерывную область физической памяти, а в некоторую совокупность областей.

Билет 15. Аппаратная поддержка ОС. Пример организации страничной виртуальной памяти.

Страничная организация памяти предполагает разделение всего пространства ОЗУ на блоки одинакового размера – страницы. Обычно размер страницы равен 2^k. В этом случае адрес, используемый в данной ЭВМ, будет иметь следующую структуру: младшие k разрядов – номер в странице, старшие – номер страницы.

Пусть виртуальное адресное пространство программы/процесса может использовать для адресации команд и данных до m страниц. Физическое адресное пространство, в общем случае может иметь произвольное число физических страниц (их может быть больше m, а может быть и меньше). Соответственно структура исполнительного физического адреса будет отличаться от структуры исполнительного виртуального адреса за счет размера поля ”номер страницы”. В виртуальном адресе размер поля определяется максимальным числом виртуальных страниц – m. В физическом адресе – максимально возможным количеством физических страниц, которые могут быть подключены к данной ЭВМ (это также фиксированная аппаратная характеристика ЭВМ).

В ЦП ЭВМ имеется аппаратная таблица страниц (иногда таблица приписки) следующей структуры: Таблица содержит m строк. Содержимое таблицы определяет соответствие виртуальной памяти физической для выполняющейся в данный момент программы/процесса. Соответствие определяется следующим образом: i-я строка таблицы соответствует i-й виртуальной странице.

Рассмотрим последовательность действий при использовании аппарата виртуальной страничной памяти: Вычисляется виртуальный исполнительный адрес. Из него выделяется поле «номер страницы». По этому значению происходит доступ к соответствующей строке таблицы страниц. Если значение строки неотрицательно, то поле «номер страницы» заменяется значением строки, то есть получаем физический адрес. Если значение строки равно -1, то либо мы обращаемся в чужую память, либо ОС откачала некоторые страницы из ОЗУ. Происходит прерывание «защита памяти» и ОС определяет причину этого прерывания.

Страничная реализация решает проблему фрагментации ОЗУ. Некоторая фрагментация остаётся (если занят 1 байт, то занята вся страница), но она контролируема. Реализация позволяет простыми средствами организовать защиту памяти и свопинг страниц.

Недостаток – необходимость наличия в ЦП аппаратной таблицы значительных размеров.

Билет 16. Многомашинные, многопроцессорные ассоциации. Классификация. Примеры.

Есть поток управляющей информации – собственно команд (инструкций), и поток данных. Считаем потоки данных и команд независимыми (условно). Рассмотрим все возможные комбинации:
SISD - single instruction (одиночный поток команд), single data stream (одиночный поток данных). Традиционные компьютеры, которые мы называем однопроцессорными. То есть для каждой команды одиночные порции операндов, которые будут обрабатываться. Пример – классическая машина по Фон - Нейману.
SIMD – векторная или матричная обработка данных.
MISD – вырожденная категория. Но к примерам можно отнести параллельные специализированные графические системы, которые занимаются, предположим, распознаванием, то есть когда над одной порцией данных одновременно используются разные команды.
MIMD – Многомашинная ассоциация. Несколько процессоров, каждый обрабатывает свои данные. Делятся на:
Системы с общей оперативной памятью. Для всех процессоров общая ОП. Исполняемая программа берется из единого пространства, куда имеют доступ все процессоры. Любое слово памяти читается одновременно несколькими процессорами, следовательно, необходима синхронизация чтения и записи. С ростом числа процессоров рост производительности замедляется. И начиная с некоторого количества увеличивать число процессоров нет смысла. Делятся на:
UMA – системы с однородным доступом в память. Каждый из процессоров имеет равные возможности и скорость доступа к ОП. Как разновидность - SMP-системы состоят из нескольких однородных процессоров и массива общей памяти, который обычно состоит из нескольких независимых блоков. «Симметричный» - все процессоры имеют доступ напрямую (т.е. возможность адресации) к любой точке памяти, причем доступ любого процессора ко всем ячейкам памяти осуществляется с одинаковой скоростью.
NUMA – системы с неоднородным доступом к памяти. Промежуточный класс между системами с общей и распределенной памятью. Память в NUMA-системах является физически распределенной, но логически общедоступной. Это означает, что каждый процессор может адресовать как свою локальную память, так и память, находящуюся на других узлах, однако время доступа к удаленным ячейкам памяти будет в несколько раз больше. Единое адресное пространство и доступ к удаленной памяти и когерентность (согласованность) кэшей во всей системе поддерживаются аппаратно. Системы с неоднородным доступом к памяти строятся из однородных базовых модулей, каждый из которых содержит небольшое число процессоров и блок памяти. Модули объединены между собой с помощью высокоскоростного коммутатора. Взаимодействие через разделяемую память Масштабируемость NUMA ограничивается объемом адресного пространства, возможностями аппаратуры поддержки когерентности кэшей и возможностями операционной системы по управлению большим числом процессоров.
Системы с распределенной ОП. Каждый процессор имеет свою локальную ОП, к которой не имеет доступ другой процессор. Делятся на:
COW – cluster of workstations. В качестве узлов – обычные рабочие станции. Если узлы разной мощности, то кластер – гетерогенный. Для связи – стандартная сетевая технология, например Ethernet.
MPP – массивно-параллельные вычисления. Однородные вычислительные узлы. Связаны высокоскоростной средой. Могут присутствовать упр. Узлы и узлы i/o.

Билет 17. Многомашинные, многопроцессорные ассоциации. Терминальные комплексы. Компьютерные сети.

Терминальный комплекс – это многомашинная ассоциация, предназначенная для организации массового доступа удаленных и локальных пользователей к ресурсам некоторой вычислительной системы. Терминальный комплекс может включать в свой состав:
1) основную вычислительную систему – систему, массовый доступ к ресурсам которой обеспечивается терминальным комплексом;
2) локальные мультиплексоры – аппаратные комплексы, предназначенные для осуществления связи и взаимодействия вычислительной системы с несколькими устройствами через один канал ввода/вывода.
3) локальные терминалы – оконечные устройства, используемые для взаимодействия пользователей с вычислительной системой, подключаемые к вычислительной системе непосредственно через каналы ввода/вывода или через локальные мультиплексоры;
4) модемы – устройства, предназначенные для организации взаимодействия вычислительной системы с удаленными терминалами с использованием телефонной сети. В функцию модема входит преобразование информации из дискретного, цифрового представления в аналоговое, используемое в телефонии, и обратно. Со стороны вычислительной системы модем подключается либо через канал ввода/вывода, либо через мультиплексор.
5) удаленные терминалы – терминалы, имеющие доступ к вычислительной системе с использованием телефонных линий связи и модемов.
6) удаленные мультиплексоры – мультиплексоры, подключенные к вычислительной системе с использованием телефонных линий связи и модемов.

Компьютерная сеть – объединение компьютеров (или вычислительных систем), взаимодействующих через коммуникационную среду.
Коммуникационная среда – каналы и средства передачи данных.
Характеристики сети: может состоять из значительного числа компьютеров, предполагает распределенную обработку информации, расширяемость, возможность построения равноправной сети (в отличие от терминального комплекса).

С точки зрения организации потоков информации можно выделить следующие разновидности каналов:
- *Симплексные каналы* - каналы, по которым передача информации ведется в одном направлении (например, телевизионный канал – обеспечивает передачу информации только в одном направлении от передающей антенны к принимающей).
- *Дуплексные каналы* - каналы, которые обеспечивают одновременную передачу информации в двух направлениях (например, телефонный разговор, мы одновременно можем и говорить и слушать).
- *Полудуплексные каналы* - каналы, которые обеспечивают передачу информации в двух направлениях, но в каждый момент времени только в одну сторону (подобно рации).

Связь удаленных терминалов с вычислительной системой осуществляется с использованием *коммутируемого канала* (после завершения связи канал разрывается, каждый раз разный)*,* либо по *выделенным каналам* (фиксированный на период аренды)*.*

Билет 18. Операционные системы. Основные компоненты и логические функции. Базовые понятия: ядро, процесс, ресурс, системные вызовы. Структурная организация ОС.

Операционная система – это комплекс программ, обеспечивающий контроль над существованием (некоторые из ресурсов ВС, как мы знаем, являются программными или логическими/виртуальными и создаются под контролем операционной системой), распределением и использованием ресурсов ВС. Любая ОС оперирует некоторым набором базовых понятий, на основе которых строится логика функционирования системы. Например, подобными базовыми понятиями могут быть задача, задание, процесс, набор данных, файл, объект.

Одним из наиболее распространенных базовых понятий ОС является процесс. **Процесс** – это совокупность машинных команд и данных, исполняющаяся в рамках ВС и обладающая правами на владение некоторым набором ресурсов. Эти права могут быть эксклюзивными, когда ресурс принадлежит только этому процессу. Некоторые из ресурсов могут разделяться, т. е. одновременно принадлежать двум и более процессам, в этом случае мы говорим о разделяемых ресурсах.

Возможно два варианта выделения ресурсов процессу:
- предварительная декларация использования тех или иных ресурсов (до начала выполнения процесса в систему передается перечень ресурсов, которые будут использованы процессом);
- динамическое пополнение списка принадлежащих процессу ресурсов по ходу выполнения процесса при непосредственном обращении к ресурсу.

Реальная схема зависит от конкретной ОС. На практике возможно использование комбинации этих вариантов.
Любая ОС должна удовлетворять следующим свойствам: надежность, защита, эффективность, предсказуемость.

Типовая структура ОС.
Ядро – резидентная часть ОС, работающая в режиме супервизора. В ядре размещаются программы обработки прерываний и драйверы наиболее «ответственных» устройств. Это могут быть и физические, и виртуальные устройства. Например, в ядре могут располагаться драйверы файловой системы, ОЗУ. Обычно ядро работает в режиме физической адресации.
Следующие уровни структуры – динамически подгружаемые драйверы физических и виртуальных устройств. Это драйверы, добавление которых в систему возможно «на ходу» без перекомпоновки программ ОС. Они могут являться резидентными и нерезидентными, а также могут работать как в режиме супервизора, так и в пользовательском режиме.

Можно выделить следующие основные логические функции ОС:
- управление процессами;
- управление ОП;
- планирование;
- управление устройствами и ФС.

Ресурсы – ресурсы ВС, на которой работает ОС.
Системные вызовы – средство ОС, обеспечивающее возможность процессов обращаться к ОС за теми или иными функциями.

Билет 19. Операционные системы. Пакетная ОС. ОС разделения времени. ОС реального времени. Распределенные и сетевые ОС.

Пакетная ОС.
Пакет программ – совокупность программ, для выполнения каждого из которых требуется некоторое время работы процессора. Этот тип был на первых компьютерах. Пакет программ – стопка перфокарт.
Стратегия переключения с одного процесса на другой, если:
- выполняемый процесс завершен
- возникло прерывание по обмену в выполняемой программе
- зафиксировался факт зацикливания.

Системы разделения времени.
Квант времени ЦП – некоторый фиксированный ОС промежуток времени работы ЦП. ЦП предоставляется процессу на один квант времени. Меняя размер кванта можно получить различные характеристики ОС. Большой квант времени удобен для отладки. Если квант времени устремить к нулю, то у пользователя создается впечатление, что он работает один на этой ОС. Это происходит потому, что критерий эффективности с точки зрения человека – время, через которое компьютер реагирует на действия человека.
Переключение выполнения процессов происходит только в одном из случаев:
- Исчерпался выделенный квант времени
- Выполнение процесса завершено
- Возникло прерывание
- Был зафиксирован факт зацикливания процесса

Системы реального времени.
Являются специализированными системами, в которых все функции планирования ориентированы на обработку некоторых событий за время, не превосходящее некоторого предельного значения. Критерий качества – обработка любого события за некоторый гарантированный промежуток времени (бортовой компьютер, автопилот…)

Реально (за исключением систем реального времени, которые могут быть разные по областям применения, важности серьезности и т.д.) используются комбинации пакетных и систем разделения времени друг в друге и с различными стратегиями.

Сетевая ОС.
Мы имеем физическую сеть, в которой подключенные компьютеры взаимодействуют с помощью протоколов. Сетевая ОС предоставляет пользователям распределенные прикладные приложения.

Распределенная ОС.
Состоит из ядра, локализованного в рамках одного компьютера, и остальных функций распределенных по компьютерам сети.

Билет 20. Организация сетевого взаимодействия. Эталонная модель ISO/OSI. Протокол, интерфейс. Стек протоколов. Логическое взаимодействие сетевых устройств.

Сети создавались как корпоративные, локальные. Каждое решение было уникальным. (каналы связи, формат передаваемой информации, программный интерфейс), следовательно перенос сетевой программы с одного компьютера на другой был невозможен, либо сильно затруднен. Т.к. мир существует на объединении и разделении предприятий, это было очень неудобно. Возникла необходимость стандартизации. *OSI* – системы открытых интерфейсов.
1..7 – все возможные уровни взаимодействия компьютеров в сети. Каждый уровень использует логически целостный набор действий и форматов данных, предназначенных для передачи информации между взаимодействующими в сети ВС. В каждом уровне модель ISO/OSI предполагает наличие некоторого количества протоколов, каждый из которых может осуществлять взаимодействие с одноименным протоколом на другой взаимодействующей машине (возможно виртуальной).

1. Физический уровень. На этом уровне однозначно определяется физическая сфера передачи данных и форматы передаваемых сигналов, решаются вопросы взаимосвязи в терминах сигналов. Этот уровень однозначно определяется физической средой, используемой для передачи данных, и отвечает за организацию физической связи между устройствами и передачи данных в сети.
2. Канальный уровень. Обеспечивает управление доступом к физической среде передачи данных, в частности обеспечение синхронизации передачи данных. Формализуются правила передачи данных. Решаются задачи обнаружения.
3. Сетевой уровень. Решается вопрос управления связью между взаимодействующими компьютерами. Решается задача маршрутизации, адресации.
4. Транспортный уровень. Решаются проблемы управления и передачи данных локализация и обработка ошибок, сервис передачи данных.
5. Сеансовый уровень Управление сеансами связи. Синхронизация отправки и приема данных. Управление подтверждением полномочий. Обработка внештатных ситуаций. Прерывания/продолжения работы в тех или иных внештатных ситуациях, управление подтверждением полномочий.
6. Представительский уровень. Разрешается проблема унификации кодировок. Уровень представления данных. На этом уровне находятся протоколы, реализующие единые соглашения перевода из внутреннего представления данных конкретной машины в сетевое и обратно.
7. Прикладной уровень. Осуществляет стандартизацию взаимодействия с прикладными системами.

Протокол – формальное описание сообщений и правил, по которым сетевые устройства (вычислительные системы) осуществляют обмен информацией. Или правила взаимодействия одноименных уровней.
Интерфейс – правила взаимодействия вышестоящего уровня с нижестоящим.
Служба или сервис – набор операций, предоставляемых нижестоящим уровнем вышестоящему.
Стек протоколов – перечень разноуровневых протоколов, реализованных в системе.

Данные от одной прикладной программы до другой в сети проходят путь от уровня протоколов прикладных программ до физического уровня на ВС, отправляющей данные, и далее на ВС, принимающей данные, они проходят этот путь обратно.

Билет 21. Организация сетевого взаимодействия. Семейство протоколов TCP/IP, соответствие модели ISO/OSI. Взаимодействие между уровнями протоколов семейства TCP/IP. IP адресация.

Свойства протоколов семейства TCP/IP: Открытые стандарты протоколов, независимость от аппаратного обеспечения сети передачи данных, уникальное именование сетевых устройств, стандартизованные протоколы прикладных программ.

Уровень доступа к сети. Стандартизация доступа к сети. Состоит из подпрограмм доступа к физической сети. TCP/IP не разделяет канальный и физический уровни.

Межсетевой уровень. Работает с дейтаграммами, адресами, выполняет маршрутизацию и «прикрывает» транспортный уровень от связи с физической сетью. Однако, в отличие от сетевого уровня модели OSI, этот уровень не устанавливает соединений с другими машинами.

Транспортный уровень. Обеспечивает доставку данных, средства для поддержки логических соединений между прикладными программами. В отличие от транспортного уровня модели OSI, в функции транспортного уровня TCP/IP не всегда входят контроль за ошибками и их коррекция. Уровень транспортных протоколов семейства представляется двумя протоколами: TCP и UDP.

Уровень прикладных программ. Состоит из прикладных программ и процессов, использующих сеть и доступных пользователю. В отличие от модели OSI, прикладные программы сами стандартизуют представление данных.

Взаимодействие между уровнями протоколов TCP/IP.

Уровень доступа к сети. Протоколы этого уровня должны знать детали физической сети. Они определяют, как использовать сеть для передачи дейтаграмм IP.

Межсетевой уровень. Протокол IP. Функции IP: формирование дейтаграмм, адресация, обмен данными между транспортным уровнем и доступа к сети, разбиение и сборка дейтаграмм. IP не обменивается контрольной информацией для установления соединения, не обнаруживает и не исправляет ошибки.

Транспортный уровень. Протокол контроля передачи TCP. Надежная доставка данных с обнаружением и исправлением ошибок и с установлением соединения. Протокол пользовательских дейтаграмм UDP. Отправляет пакеты, не заботясь о доставке, следовательно, быстрее, меньше нагрузка на сеть. UDP для локальной сети, а TCP для межсетевого взаимодействия.

Уровень прикладных программ. Все протоколы этого уровня пользуются протоколами транспортного уровня. Протоколы на TCP: TELNET – для удаленного доступа, FTP – для передачи файлов, SMTP – для почты. На UDP: DNS – служба именования, устанавливает однозначное соответствие между IP адресами и именами сетевых устройств, RIP – протокол информации о маршрутизации.

Система адресации протокола IP.

IP адрес представляется последовательностью 4 байт. В адресе кодируется уникальный номер сети, номер сетевого устройства. Классы адресов: A – первый бит 0, номер сети – 1 байт. Номер сети меньше 127. Исторически принадлежат мировым корпорациям. B – Первые биты – 10, номер сети – 2 байта, C – 110, 3 байта. Самые распространенные. D – 1110, остальные биты – номер группы. E – 1111.

Основные понятия.

Пакет – блок данных с информацией для корректной доставки.

Дейтаграмма – пакет протокола IP. Контрольная информация в заголовке размером 6 32-битных слов.

Шлюз – устройство, передающее пакеты между сетями.

Маршрутизация – процесс выбора шлюза или маршрутизатора.

Билет 22. Управление процессами. Определение процесса, типы. Жизненный цикл, состояния процесса. Свопинг. Модели жизненного цикла процесса. Контекст процесса.

Процесс – это совокупность машинных команд и данных, исполняющаяся в рамках ВС и обладающая правами на владение некоторым набором ресурсов. В различных системах различные определения. Типы процессов:
Полновесные процессы – процессы, выполняющиеся внутри защищенной памяти, имеющие собственные виртуальные адресные пространства.
Легковесные процессы - не имеют собственных защищенных областей памяти. Работают одновременно с активировавшей их задачей и используют ее адресное пространство, в котором выделяется память под динамические данные.
Понятие процесс включает в себя: исполняемый код, адресное пространство, ресурсы системы, принадлежащие процессу, хотя бы одну исполняемую нить.

Совокупность этапов обработки процесса в системе называется **жизненным циклом** процесса в системе. Содержит этапы образования, ожидания постановки на выполнение, выполнения и завершения процесса. *Буфер ввод процессов* – пространство, в котором размещаются и хранятся процессы с момента образования до момента начала выполнения. *Буфер обрабатываемых процессов* – все процессы, находящиеся в обработке. Жизненный цикл: 1.После формирования в БВП 2.В БВП выбирается наиболее приоритетный процесс, он попадает в БОП 3.Прекращает выполнение, либо по причине ожидания операции ввода-вывода и попадает в БОП, тогда ожидает окончания, либо истек квант времени, тогда он попадает в БОП. 4.Завершение процесса, освобождение системных ресурсов.

С каждым из процессов из БОП система ассоциирует совокупность данных, характеризующих актуальное состояние процесса – **контекст процесса**. (в общем случае контекст процесса содержит информацию о текущем состоянии процесса, включая информацию о режимах работы процессора, содержимом регистровой памяти, используемой процессом, системной информации ОС, ассоциированной с данным процессом). Процессы, находящиеся в одном из состояний ожидания в своих контекстах содержат всю информацию, необходимую для продолжения выполнения. Контекст состоит из: пользовательской составляющей (состояние программы, как совокупности машинных команд и данных, размещенных в ОЗУ), системной составляющей (содержимое регистров, настройки аппарата защиты памяти, виртуальной памяти, принадлежащие процессу ресурсы).

Мультипрограммные ОС используют свопинг. В контексте процесса была переменная p_time – время непрерывного размещения процесса в область свопинга и обратно. p_time обнуляется и считает время, пока процесс находится в этой области. Затем запускалась функция обработки, которая анализировала область свопинга и выбирала процесс с максимальным p_time, затем система анализировала наличие свободной оперативной памяти. Если ее достаточно, то процесс в нее загружался, иначе операционная система смотрела процесс, который закрыт по обмену и имеет максимальный p_time, и его откачивала в область свопинга. Затем вновь анализ оперативной памяти, если недостаточно, то далее анализировались процессы в оперативной памяти, затем отыскивался процесс с максимальным p_time и освобождал память.

Билет 23. Реализация процессов в ОС UNIX. Определение процесса. Контекст, тело процесса. Состояния процесса. Аппарат системных вызовов в UNIX.

В любой системе, оперирующей понятием процесс, существует системно-ориентированное определение процесса. В ОС UNIX **процесс** – объект, зарегистрированный в таблице процессов. Или объект, порожденный системным вызовом fork().
Каждый процесс характеризуется уникальным именем – *идентификатором процесса* (PID). PID – целое число от 0 до некоторого предельного значения, определяющего максимальное число процессов, существующих в системе одновременно. С точки зрения организации данных PID – номер строки в таблице, в которой размещена запись о процессе.

Содержимое записи таблицы процессов позволяет получить **контекст процесса** (часть данных контекста размещается непосредственно в записи таблицы процессов, на оставшуюся часть контекста имеются прямые или косвенные ссылки, также размещенные в записи таблицы процессов). С точки зрения логической структуры контекст процесса UNIX состоит из:

- пользовательской составляющей или тела процесса. Тело состоит из сегмента кода и сегмента данных. Сегмент кода содержит машинные команды и неизменяемые константы, соответствующей процессу программы. Сегмент данных содержит область статических переменных, область стека.
- аппаратной составляющей*,* содержащей все регистры и аппаратные таблицы ЦП.
- системной составляющей ОС UNIX. Различные атрибуты процесса: идентификатор родительского процесса, текущее состояние процесса, приоритет, реальные и эффективные идентификаторы пользователя-владельца и его группы, список областей памяти, таблица открытых файлов, реакции на сигналы.

Чтобы работать в с ресурсами ВС нужно перейти в привилегированный режим ОС. Для этого существуют системные вызовы, предоставляемые ОС UNIX. К интересующим нас вызовам относятся вызовы
- для создания процесса;
- для организации ввода вывода;
- для решения задач управления;
- для операции координации процессов;
- для установки параметров системы.

Общие моменты, связанные с работой системных вызовов. Большая часть системных вызовов определены как функции, возвращающие целое значение, при этом при нормальном завершении системный вызов возвращает 0, а при неудачном завершении -1. При этом код ошибки можно выяснить, анализируя значение внешней переменной errno.

Билет 24. Реализация процессов в ОС UNIX. Базовые средства управления процессами в ОС UNIX. Загрузка системы, формирование нулевого и первого процессов.

Для создания новых процессов, кроме 0 и 1, применяется единая схема – системный вызов **fork()**. В таблицу процессов заносится новая запись, порожденный процесс получает уникальный идентификатор. Создается контекст, большая часть которого наследуется от родительского процесса (открытые файлы, способы обработки сигналов, разделяемые ресурсы, текущий рабочий каталог). По завершении fork(), каждый из процессов продолжит выполнение с точки возврата из вызова. Fork() возвращает сыну 0, а отцу PID сына, в случае неудачи -1.

Семейство системных вызовов **exec()** производит замену тела вызывающего процесса, после чего управление передается на точку входа другой программы. Возврат к первоначальной программе только при ошибке exec(). Происходит замена только сегмента кода и сегмента данных, режимов обработки сигналов. Могут измениться эффективные идентификаторы и закрыться некоторые файлы. Полезен в сочетании с fork().

Для завершения процесса используется системный вызов **_exit()**. Также причинами завершения процесса могут быть некоторые сигналы и оператор return функции main. В любом случае освобождается сегмент кода и сегмент данных процесса, закрываются открытые дескрипторы файлов, если у процесса есть потомки, то их предком назначается процесс с PID 1, процессу предку посылается SIGCHLD, освобождается большая часть контекста, но остается запись в таблице. Процесс переходит в состояние зомби.

Процесс предок имеет возможность получить информацию о завершении своих потомков с помощью вызова **wait(),** при обращении к которому выполнение останавливается, пока не завершится какой-нибудь потомок. Wait() вернет PID завершенного процесса, а через параметр будет возвращена информация о причине завершения этого процесса. Если нет потомков, вернет -1, если завершился, то wait не ждет.

Начальная загрузка системы. Аппаратный загрузчик читает нулевой блок системного устройства. После чтения программы, выполняющей начальную загрузку, она считывает файл /unix в память. Он запускается на исполнение. Происходит инициализация системы (устанавливаются часы, формируется диспетчер памяти), запускается процесс 0. Он не имеет сегмента кода, это просто структура данных, используемая ядром. Он существует в течении всего времени работы системы. Далее ядро копирует процесс 0 в процесс 1, в сегмент кода которого копируется программа, вызывающая exec(), необходимый для выполнения /etc/init. Init организует процесс getty для каждого активного канала связи, который ожидает входа кого-либо по каналу связи. Далее getty передает управление программе login, проверяющей пароль. Если пароль верный, то запускается shell, если неверный, то снова переходим на getty. Во время работы ОС init ожидает завершения одного из порожденных им процессов, после чего создает новую программу getty для соответствующего терминала. Таким образом, init поддерживает многопользовательскую структуру во время функционирования ОС.

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

Процессы, выполнение которых хотя бы частично перекрывается по времени, называются параллельными. Они могут быть независимыми и взаимодействующими. Независимые процессы – процессы, использующие независимое множество ресурсов; на результат работы такого процесса не должна влиять работа другого независимого процесса. Наоборот – взаимодействующие процессы совместно используют ресурсы и выполнение одного процесса может оказывать влияние на результат другого. Совместное использование ресурса двумя процессами, когда каждый из процессов полностью владеет ресурсом некоторое время, называют **разделением ресурса**. Разделению подлежат как аппаратные, так программные ресурсы. Разделяемые ресурсы, которые должны быть доступны в текущий момент времени только одному процессу – это так называемые **критические ресурсы**. Таковыми ресурсами могут быть как внешнее устройство, так и некая переменная, значение которой может изменяться разными процессами. Необходимо решать задачи разделения ресурсов и их защиты от неконтролируемого доступа со стороны других процессов. Важнейшим требованием мультипрограммирования с точки зрения распределения ресурсов является следующее: результат выполнения процессов не должен зависеть от порядка переключения выполнения между процессами.

Взаимное исключение – такой способ работы с разделяемым ресурсом, при котором постулируется, что в тот момент, когда один из процессов работает с разделяемым ресурсом, все остальные процессы не могут иметь к нему доступ. Проблему организации взаимного исключения можно сформулировать в более общем виде. Часть программы (фактически набор операций), в которой осуществляется работа с критическим ресурсом, называется **критической секцией**. Задача взаимного исключения в этом случае сводится к тому, чтобы не допускать ситуации, когда два процесса одновременно находятся в критических секциях, связанных с одним и тем же ресурсом.

Важно отметить, что при организации взаимного исключения могут возникнуть две неприятные проблемы:

1. Возникновение так называемых **тупиков (deadlocks**). Рассмотрим следующую ситуацию: имеются процессы **A** и **B**, каждому из которых в некоторый момент требуется иметь доступ к двум ресурсам **R1** и **R2.** Процесс **A** получил доступ к ресурсу **R1**, и следовательно, никакой другой процесс не может иметь к нему доступ, пока процесс **A** не закончит с ним работать. Одновременно процесс **B** завладел ресурсом **R2.** В этой ситуации каждый из процессов ожидает освобождения недостающего ресурса, но оба ресурса никогда не будут освобождены, и процессы никогда не смогут выполнить необходимые действия.
2. Ситуация блокирования (дискриминации) одного из процессов, когда один из процессов будет бесконечно находиться в ожидании доступа к разделяемому ресурсу, в то время как каждый раз при его освобождении доступ к нему получает какой-то другой процесс.

Любая реализация взаимного исключения должна решать эти проблемы, плюс не должно быть ситуации, при которой процесс, находясь вне критической секции, блокирует исполнение другого процесса и не должно делаться никаких предположений относительно взаимных скоростей выполнения процессов.

Билет 26. Взаимодействие процессов. Способы реализации взаимного исключения: семафоры Дейкстры, мониторы, обмен сообщениями.

Семафоры.
Дейкстра предложил новый тип данных – семафор, - над которым определены атомарные операции down и up. down проверяет значение семафора, и если оно больше нуля, то уменьшает его на 1. Если же это не так, процесс блокируется, причем операция down считается незавершенной. Операция up увеличивает значение семафора на 1. При этом, если в системе присутствуют процессы, блокированные ранее при выполнении down на этом семафоре, ОС разблокирует один из них с тем, чтобы он завершил выполнение операции down. Аналогия с N тележками, при N = 1 – взаимное исключение, двоичный семафор. Мощное средство синхронизации.

Мониторы.
Предложены Хоаром в 1974. Мониторы – языковые конструкции. Совокупность процедур и структур данных, объединенных в программный модуль. Три основных свойства: Структуры данных, входящие в монитор, доступны только для процедур, входящих в монитор; процесс входит в монитор посредством вызова одной из его процедур; в любой момент времени внутри монитора может находиться только один процесс. Если процесс пытается попасть в монитор, в котором уже находится другой процесс, он блокируется. Таким образом, чтобы защитить разделяемые структуры данных, их достаточно поместить внутрь монитора вместе с процедурами, представляющими критические секции для их обработки. Также есть переменные-условия, на которых определены операции wait и signal. Они используются для синхронизации. Если процесс внутри монитора обнаруживает, что не может продолжать выполнение, пока не выполнится определенное условие (например, буфер для записи данных переполнился), он вызывает операцию wait над переменной-условием. Дальнейшее выполнение блокируется, и это позволяет другому процессу, ожидающему входа в монитор, попасть в него. Если другой процесс что-то изменит, то он должен вызвать signal и разблокировать ожидающий процесс, причем вызов signal должен быть самым последним внутри монитора.

Обмен сообщениями.
Является более общим средством, решающим проблему синхронизации как для однопроцессорных систем и систем с общей памятью, так и для распределенных. Может быть использовано как для синхронизации, так и для обмена информацией. Основная функциональность реализуется системными вызовами send и receive. Возникает вопрос синхронизации. Как операция посылки сообщения, так операция приема могут быть блокирующими и неблокирующими. Для операции send это означает, что либо процесс-отправитель может блокироваться до тех пор, пока получатель не вызовет receive, либо выполнение процесса может продолжаться далее независимо от наличия получателя. Для операции receive подобная ситуация возникает, когда эта операция вызвана раньше, чем сообщение было послано Возможны следующие комбинации: обе блокирующие (схема рандеву, не требует буферизации сообщений), неблокирующий send и блокирующий receive (в системах клиент/сервер, необходимо обеспечивать гарантированную доставку сообщений, если блокирующий receive, то получатель может оказаться заблокированными навечно, нужны дополнительные средства), обе неблокирующие. Другая проблема – адресация. Либо прямая, либо специальная структура – почтовый ящик или очередь сообщений. Нужны дополнительные примитивы создания очереди и присоединения к ней. Длина как фиксированная, так и переменная. Во втором случае длину надо передавать в заголовке. Сообщения менее быстры, чем семафоры и мониторы.

Билет 27. Взаимодействие процессов. Классические задачи синхронизации процессов. «Обедающие философы».

Пять философов собираются за круглым столом, перед каждым из них стоит блюдо со спагетти, и между каждыми двумя соседями лежит вилка. Каждый из философов размышляет, затем берет две вилки, ест, затем кладет вилки обратно на стол и опять размышляет и так далее. Каждый из них ведет себя независимо от других, однако вилок запасено ровно столько, сколько философов, хотя для еды каждому из них нужно две. Таким образом, философы должны совместно использовать имеющиеся у них вилки (ресурсы). Задача состоит в том, чтобы найти алгоритм, который позволит философам организовать доступ к вилкам таким образом, чтобы каждый имел возможность насытиться, и никто не умер с голоду.
Алгоритм решения может быть представлен следующим образом (N = 5):

```
# define LEFT (i-1)%N              /* номер левого соседа для i-ого философа */
# define RIGHT (i+1)%N             /* номер правого соседа для i-ого философа*/
# define THINKING 0                /* философ думает */
# define HUNGRY 1                  /* философ голоден */
# define EATING 2                  /* философ ест */
int state[N];                      /* массив состояний философов */
semaphore mutex = 1;               /* семафор для критической секции */
semaphore s[N];                    /* по одному семафору на философа */
void philosopher (int i)           /* i : номер философа от 0 до N-1 */
{
        while (TRUE) {                    /* бесконечный цикл */
                think();                  /* философ думает */
                take_forks(i);            /* философ берет обе вилки или блокируется */
                eat();                    /* философ ест */
                put_forks(i); }           /* философ освобождает обе вилки */
}
void take_forks(int i)             /* i : номер философа от 0 до N-1 */
{
        down(&mutex);              /* вход в критическую секцию */
        state[i] = HUNGRY;         /* записываем, что i-ый философ голоден */
        test(i);                   /* попытка взять обе вилки */
        up(&mutex);                /* выход из критической секции */
        down(&s[i]);               /* блокируемся, если вилок нет */
}
void put_forks(int i)              /* i : номер философа от 0 до N-1 */
{
        down(&mutex);              /* вход в критическую секцию */
        state[i] = THINKING;       /* философ закончил есть */
        test(LEFT);                /* проверить, может ли левый сосед сейчас есть */
        test(RIGHT);               /* проверить, может ли правый сосед сейчас есть*/
        up(&mutex);                /* выход из критической секции */
}
void test(int i)                   /* i : номер философа от 0 до N-1 */
{
        if (state[i] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING) {
                state[i] = EATING;
                up (&s[i]);     }
}
```

Билет 28. Взаимодействие процессов. Классические задачи синхронизации процессов. «Читатели и писатели».

Множество конкурирующих процессов хотят читать и обновлять одни и те же данные. Несколько процессов могут читать данные одновременно, но когда один процесс начинает записывать данные (обновлять базу данных проданных билетов), ни один другой процесс не должен иметь доступ к данным, даже для чтения.

```
semaphore mutex = 1;      /* контроль за доступом к «rc» (разделяемый ресурс) */
semaphore db = 1;         /* контроль за доступом к базе данных */
int rc = 0;               /* кол-во процессов читающих или пишущих */
void reader (void)
{
        while (TRUE) /* бесконечный цикл */
        {
                down(&mutex);           /* получить эксклюзивный доступ к «rc»*/
                rc = rc + 1;            /* еще одним читателем больше */
                if (rc == 1) down(&db); /* если первый, то заблокировать доступ к бд */
                up(&mutex);             /* освободить ресурс rc */
                read_data_base();       /* доступ к данным */
                down(&mutex);           /* получить эксклюзивный доступ к «rc»*/
                rc = rc - 1:            /* теперь одним читателем меньше */
                if (rc == 0) up(&db);   /* последний - разблокировать доступ к бд */
                up(&mutex);             /* освободить разделяемый ресурс rc */
                use_data_read();        /* некритическая секция */
        }
}
void writer (void)
{
        while(TRUE)                     /* бесконечный цикл */
        {
                think_up_data();        /* некритическая секция */
                down(&db);              /* получить эксклюзивный доступ к данным*/
                write_data_base();      /* записать данные */
                up(&db);                /* отдать эксклюзивный доступ */
        }
}
```

Алгоритм дает преимущество процессам-читателям. Если имеется процесс-писатель, и постоянно появляются новые читатели, то писатель никогда не получит доступ. Решение – модифицировать алгоритм таким образом, чтобы в случае, если имеется хотя бы один ожидающий процесс-писатель, новые процессы-читатели не получали доступа к ресурсу, а ожидали, когда процесс-писатель обновит данные. В этой ситуации процесс-писатель должен будет ожидать окончания работы с ресурсом только тех читателей, которые получили доступ раньше, чем он его запросил. Однако, обратная сторона данного решения в том, что оно несколько снижает производительность процессов-читателей, так как вынуждает их ждать в тот момент, когда ресурс не занят в эксклюзивном режиме.

Билет 29. Базовые средства взаимодействия процессов в ОС UNIX. Сигналы. Примеры программирования.

Сигналы представляют собой средство уведомления процесса о наступлении некоторого события в системе. Посылать сигнал могут как другие процессы, так и ОС. Под определенные события в ОС зарезервированы определенные сигналы (SIGINT, SIGALRM, SIGKILL, SIGCHLD,…). При получении сигнала возможны 3 варианта: реакция по умолчанию, специальная обработка, игнорирование. Процесс может установить свою реакцию на сигнал (кроме SIGKILL, SIGSTOP) и менять ее в процессе работы. Сигналы и некоторые функции определены в файле signal.h. Для отправки сигнала существует системный вызов kill(pid, signal), для установки реакции на сигнал служит системный вызов signal(signal, handler), возвращающий старый обработчик. Вторым аргументом может быть SIG_IGN, SIG_DFL. После прихода сигнала обработка сбрасывается на обработку по умолчанию. Системный вызов alarm(sec) инициирует приход SIGALRM через sec секунд. Пример будильник:

```
void alrm(int s)        /*обработчик сигнала SIG_ALRM */
{
        printf("\n жду имя \n");
        alarm(5);                       /* заводим будильник */
        signal(SIGALRM, alrm);    /* переустанавливаем реакцию на сигнал */
}
int main(int argc, char **argv)
{
        char s[80];
        signal(SIGALRM, alrm);    /* установка обработчика alrm на  SIG_ALRM */
        alarm(5);                       /* заводим будильник */
        printf("Введите имя \n");
        for (;;)
        {
                printf("имя:");
                if(gets(s) != NULL) break;  /* ожидаем ввода имени */
        }
        printf("OK! \n");
        return 0;
}
```

-----Пример 2: обработка по умолчанию на 5й раз.

```
int count = 0;
void SigHndlr (int s) /* обработчик сигнала */
{
        printf("\n I got SIGINT %d time(s) \n",
        ++ count);
        if (count == 5) signal (SIGINT, SIG_DFL); /* ставим обработчик по умолчанию */
        else signal (SIGINT, SigHndlr);                 /* восстанавливаем обработчик */
}
int main(int argc, char **argv)
{
        signal (SIGINT, SigHndlr);                  /* установка реакции на сигнал */
        while (1);                                   /*"тело программы" */
        return 0;
}
```

Билет 30. Базовые средства взаимодействия процессов в ОС UNIX. Неименованные каналы. Примеры программирования.

Неименованный канал есть некая сущность, в которую можно помещать и извлекать данные, для чего служат два файловых дескриптора, ассоциированных с каналом: один для записи в канал, другой — для чтения. Для создания канала служит системный вызов pipe(int *fd), возвращающий через массив fd 2 дескриптора для чтения и записи в канал. Отличия от файла: невозможен доступ по имени; канал не существует вне процесса, доступ только FIFO, невозможна операция lseek(). Отличия при чтении: если прочитано меньше байт, чем находится в канале, то оставшиеся остаются в канале; если пытаемся прочесть больше, и есть открытые дескрипторы записи, то будет прочитано что есть, и процесс заблокируется; при закрытии записывающей стороны в канал помещается EOF. Отличия при записи: если пытаемся записать больше, чем помещается, то записываем что помещается и блокируемся; если записываем в канал, с которым не ассоциирован ни один дескриптор чтения, то процесс получит сигнал SIGPIPE. При отсутствии переполнения операция записи атомарная. Чаще всего канал используется для обмена данными между разными процессами. Используется факт наследования процессом-сыном таблицы файловых дескрипторов отца. Схема работы с каналом: создаем массив 2х интов, канал, делаем форк, в каждом процессе закрываем ненужный дескриптор. В читающем процессе while(read()). Пример конвейер:

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
int main(int argc, char **argv)
{
	int fd[2];
	pipe(fd); /*организован канал*/
	if (fork())
	{
		/*процесс-родитель*/
		dup2(fd[1], 1); /* отождествили стандартный вывод с файловым дескриптором канала, предназначенным для записи */
		close(fd[1]); /* закрыли файловый дескриптор канала, предназначенный для записи */
		close(fd[0]); /* закрыли файловый дескриптор канала, предназначенный для чтения */
		exelp(“print”, ”print”, 0); /* запустили программу print */
	}
	/*процесс-потомок*/
	dup2(fd[0], 0); /* отождествили стандартный ввод с файловым дескриптором канала, предназначенным для чтения*/
	close(fd[0]); /* закрыли файловый дескриптор канала, предназначенный для чтения */
	close(fd[1]); /* закрыли файловый дескриптор канала, предназначенный для записи */
	execl(“/usr/bin/wc”, ”wc”, 0); /* запустили программу wc */
}
```

Dup2 дублирует дескриптор, print печатает текст, wc выводит количество строк.

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

К именованным каналам может подключиться любой процесс в любое время благодаря наличию у них имен. FIFO-файл – отдельный тип файла в файловой системе UNIX. Для его создания служит системный вызов mkfifo(pathname, mode). Первый аргумент – имя создаваемого канала, второй – права доступа к нему, флаг S_IFIFO, указывающий, что объект является именно FIFO файлом. После создания канала любой процесс может установить с ним связь с помощью вызова open(). При этом: если процесс открывает FIFO-файл для чтения, то он блокируется, пока другой процесс не откроет его на запись, и наоборот. Можно избежать блокирования установкой в open() флага O_NONBLOCK. Пример клиент-сервер:

```
/* процесс-сервер*/
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
int main(int argc, char **argv)
{
        int fd;
        int pid;
        mkfifo("fifo", S_IFIFO | 0666); /*создали специальный файл FIFO с открытыми
        для всех правами доступа на чтение и запись*/
        fd = open("fifo", O_RDONLY | O_NONBLOCK); /* открыли канал на чтение*/
        while (read (fd, &pid, sizeof(int)) == -1);
        printf("Server %d got message from %d !\n", getpid(), pid);
        close(fd);
        unlink("fifo"); /*уничтожили именованный канал*/
        return 0;
}

/* процесс-клиент*/
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
int main(int argc, char **argv)
{
        int fd;
        int pid = getpid( );
        fd = open("fifo", O_RDWR);
        write(fd, &pid, sizeof(int));
        close(fd);
        return 0;
}
```

Билет 32. Базовые средства взаимодействия процессов в ОС UNIX. Взаимодействие процессов по схеме «подчиненный-главный». Трассировка.

Используется для отладки. Один процесс получает возможность управлять ходом выполнения, данными и кодом другого. В ОС UNIX трассировка возможна только между родственными процессами, причем только после того, как процесс-потомок даст разрешение.

Схема взаимодействия процессов путем трассировки: выполнение отлаживаемого процесса приостанавливается всякий раз при получении сигнала и при выполнении exec(), и пока процесс приостановлен, процесс-отладчик имеет возможность анализировать и изменять данные в адресном пространстве отлаживаемого процесса ив пользовательской составляющей его контекста, затем процесс-отладчик возобновляет выполнение трассируемого процесса до следующей остановки.

Основной системный вызов – **ptrace(cmd, pid, addr, data)**. Cmd – код выполняемой команды, pid – идентификатор процесса-потомка, addr – некоторый адрес в адресном пространстве потомка, data – слово информации.

cmd = PTRACE_TRACEME — сыновний процесс позволяет трассировать себя**.** Все остальные обращения к вызову **ptrace()** осуществляет процесс-отладчик.
cmd = PTRACE_PEEKDATA — чтение слова из адресного пространства отлаживаемого процесса по адресу **addr**, **ptrace()** возвращает значение этого слова.
cmd = PTRACE_PEEKUSER — чтение слова из контекста процесса**.** В этом случае параметр **addr** есть смещение относительно начала структуры контекста. **ptrace()** возвращает значение считанного слова.
cmd = PTRACE_POKEDATA — запись данных, размещенных в параметре **data**, по адресу **addr** в адресном пространстве процесса-потомка.
cmd = PTRACE_POKEUSER — запись слова из **data** в контекст трассируемого процесса со смещением **addr**.
cmd = PTRACE_GETREGS — чтение регистров общего назначения (в т. ч. с плавающей точкой) трассируемого процесса и запись их значения по адресу **data**.
cmd = PTRACE_SETREGS — запись в регистры общего назначения трассируемого процесса данных, расположенных по адресу **data** в трассирующем процессе.
cmd = PTRACE_CONT — возобновление выполнения трассируемого процесса**.**
cmd = PTRACE_SYSCALL, PTRACE_SINGLESTEP — эта команда, аналогично **PTRACE_CONT**, возобновляет выполнение трассируемой программы, но при этом произойдет ее остановка после того, как выполнится одна инструкция. Таким образом, используя **PTRACE_SINGLESTEP**, можно организовать пошаговую отладку. С помощью команды **PTRACE_SYSCALL** возобновляется выполнение трассируемой программы вплоть до ближайшего входа или выхода из системного вызова.
cmd = PTRACE_KILL — завершение выполнения трассируемого процесса.

Общая схема: форк, сыновний процесс разрешает трассировать себя, тело замещается телом процесса, который нужно трассировать; в родителе: ptrace(PTRACE_SINGLESTEP, pid, 0, 0); wait(NULL); далее любые действия над трассируемым процессом.

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

Для всех средств IPC общая схема именования объектов. Для именования используется уникальное целое число – ключ, зная который можно получить доступ к объекту. Процессу возвращается дескриптор, для дальнейших операций с объектом. Необходим некий механизм получения заведомо уникального ключа для именования ресурса и нужно, чтобы этот механизм позволял всем процессам, желающим работать с одним ресурсом, получить одно и то же значение ключа. Для этого существует функция ftok(), генерирующая ключ по строке и добавочному символу. Для создания ресурса с заданным ключом либо подключения к существующему ресурсу используются системные вызовы с суффиксом get. Параметры – ключ и флаги. В качестве ключа может быть IPC_PRIVATE, получим ресурс, доступный только породившему процессу. Система не удаляет разделяемые ресурсы автоматически, для управления – функции с суффиксом ctl.

Очередь сообщений представляет собой некое хранилище типизированных сообщений, организованное по принципу FIFO. Любой процесс может помещать новые сообщения в очередь и извлекать из очереди имеющиеся там сообщения. Каждое сообщение имеет тип, представляющий собой некоторое целое число. благодаря наличию типов сообщений, очередь можно интерпретировать двояко — рассматривать ее либо как сквозную очередь неразличимых по типу сообщений, либо как некоторое объединение подочередей, каждая из которых содержит элементы определенного типа. Извлечение сообщений из очереди происходит согласно принципу FIFO – в порядке их записи, однако процесс-получатель может указать, из какой подочереди он хочет извлечь сообщение.
msgget(key, msgflag), доступ к очереди, возвращает дескриптор очереди.
msgsnd(msqid, *msgp, msgsize, msgflg), передача сообщений. Первый аргумент – дескриптор, второй – указатель на буфер с данными (структура с полями тип сообщения и тело сообщения). Третий аргумент – размер буфера, четвертый – либо 0 либо IPC_NOWAIT. Если 0, то если дина сообщений в очереди больше максимально допустимого, процесс блокируется.
msgrcv(msqid, *msgp, msgsize, msgtype, msgflag), для получения сообщений. Первые три аргумента аналогичны, четвертый – тип. Если 0, то любой тип. Если меньше 0, то минимального типа. Пятый – флаги. IPC_NOWAIT, MSG_NOERROR – может прочитать сообщение, если размер буфера меньше длины сообщения(первые «размер буфера» байт).
msgctl(msqid, cmd, struct msgid_ds *buf), для управления. В структуре, в третьем аргументе, хранятся права доступа к очереди, размер, статистика обращений к очереди. Возможные значения cmd - IPC_STAT – скопировать структуру, описывающую управляющие параметры очереди по адресу, указанному в параметре buf; IPC_SET – заменить структуру, описывающую управляющие параметры очереди, на структуру, находящуюся по адресу, указанному в параметре buf; IPC_RMID – удалить очередь. Удалить очередь может только процесс, у которого эффективный идентификатор пользователя совпадает с владельцем или создателем очереди, либо процесс с правами привилегированного пользователя.

+Пример. Клиент-сервер.

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

Для всех средств IPC общая схема именования объектов. Для именования используется уникальное целое число – ключ, зная который можно получить доступ к объекту. Процессу возвращается дескриптор, для дальнейших операций с объектом. Необходим некий механизм получения заведомо уникального ключа для именования ресурса и нужно, чтобы этот механизм позволял всем процессам, желающим работать с одним ресурсом, получить одно и то же значение ключа. Для этого существует функция ftok(), генерирующая ключ по строке и добавочному символу. Для создания ресурса с заданным ключом либо подключения к существующему ресурсу используются системные вызовы с суффиксом get. Параметры – ключ и флаги. В качестве ключа может быть IPC_PRIVATE, получим ресурс, доступный только породившему процессу. Система не удаляет разделяемые ресурсы автоматически, для управления – функции с суффиксом ctl.

Механизм разделяемой памяти позволяет нескольким процессам получить отображение страниц из своей виртуальной памяти на общую область физической памяти. Благодаря этому, данные, находящиеся в этой области памяти, будут доступны для чтения и модификации всем процессам, подключившимся к данной области памяти. Процесс, подключившийся к разделяемой памяти, может затем получить указатель на некоторый адрес в своем виртуальном адресном пространстве, соответствующий данной области разделяемой памяти. После этого он может работать с этой областью памяти аналогично тому, как если бы она была выделена динамически.

shmget(key, size, flag), если подключаемся к существующей, то size должен быть не более размера памяти. Меньше – размер уменьшится. В случае успеха возвращает дескриптор памяти.
shmat(shmid, shmaddr, shmflg), shmid – дескриптор, shmaddr – адрес в виртуальном пространстве, с которого присоединяется память, если 0, то система сама выберет адрес. Shmflg – флаги (SHM_RDONLY – только для чтения). Функция возвращает адрес, начиная с которого будет отображаться разделяемая память, в случае неудачи -1.
shmdt(shmaddr) позволяет отсоединить память по указанному адресу. В случае успешного завершения функция возвращает 0.
shmctl(shmid, cmd, struct shmid_ds *buf), для управления, наложения или снятия блокировки. В полях структуры содержится информация о правах доступа к области памяти, размер, число процессов, присоединенных к ней в данный момент, статистика обращений к памяти. Возможные значения аргумента cmd: IPC_STAT – скопировать структуру, описывающую управляющие параметры области памяти по адресу, указанному в параметре buf IPC_SET – заменить структуру, описывающую управляющие параметры области памяти, на структуру, находящуюся по адресу, указанному в параметре buf. Выполнить эту операцию может процесс, у которого эффективный идентификатор пользователя совпадает с владельцем или создателем очереди, либо процесс с правами привилегированного пользователя, при этом процесс может изменить только владельца области памяти и права доступа к ней. IPC_RMID – удалить память. SHM_LOCK, SHM_UNLOCK – блокировать или разблокировать область памяти.

+ Пример.

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

Для всех средств IPC общая схема именования объектов. Для именования используется уникальное целое число – ключ, зная который можно получить доступ к объекту. Процессу возвращается дескриптор, для дальнейших операций с объектом. Необходим некий механизм получения заведомо уникального ключа для именования ресурса и нужно, чтобы этот механизм позволял всем процессам, желающим работать с одним ресурсом, получить одно и то же значение ключа. Для этого существует функция ftok(), генерирующая ключ по строке и добавочному символу. Для создания ресурса с заданным ключом либо подключения к существующему ресурсу используются системные вызовы с суффиксом get. Параметры – ключ и флаги. В качестве ключа может быть IPC_PRIVATE, получим ресурс, доступный только породившему процессу. Система не удаляет разделяемые ресурсы автоматически, для управления – функции с суффиксом ctl.

Семафоры используются для синхронизации доступа процессов к разделяемым ресурсам, так как другие средства IPC не предоставляют механизма синхронизации. Семафор представляет собой особый вид числовой переменной, над которой определены две неделимые операции: уменьшение ее значения с возможным блокированием процесса и увеличение значения с возможным разблокированием одного из ранее заблокированных процессов. Объект IPC представляет собой набор семафоров. Использование семафоров в качестве средства синхронизации доступа к другим разделяемым объектам предполагает следующую схему: с каждым разделяемым ресурсом связывается один семафор из массива, положительное значение семафора – ресурс свободен, неположительное – занят; перед входом в критическую секцию – down, перед выходом - up.

semget(key, nsems, semflag), для доступа. Второй аргумент – количество семафоров. Процесс, имеющий право доступа к массиву семафоров по чтению, может проверять значение семафоров; процесс, имеющий право доступа по записи, может как проверять, так и изменять значения семафоров. В случае, если среди флагов указан IPC_CREAT, аргумент nsems должен представлять собой положительное число, если же этот флаг не указан, значение nsems игнорируется.
semop(semid, struct sembuf *semop, nops), для операций над семафорами. semid – дескриптор массива семафоров; semop – массив из объектов типа struct sembuf, каждый из которых задает одну операцию над семафором; nops – длина массива semop. Количество семафоров, над которыми процесс может одновременно производить операцию в одном вызове semop(), ограничено константой SEMOPM. Структура имеет sembuf вид: struct sembuf {short sem_num; */* номер семафора в векторе */* short sem_op; */* производимая операция */* short sem_flg; */* флаги операции */* } Ненулевое значение поля sem_op обозначает необходимость прибавить к текущему значению семафора значение sem_op, а нулевое – дождаться обнуления семафора.В sem_flg – комбинация флагов. IPC_NOWAIT – не блокировать процесс, SEM_UNDO – по завершении процесса отмена сделанных изменений.
semctl(semid, num, cmd, union semun arg), num – индекс в массиве, union semnum с полями val (значение семафора), struct semid_ds *buf (параметры массива семафоров) ushort *arr (массив значений семафоров). Значения cmd: IPC_STAT, IPC_SET, IPC_RMID. GETALL/SETALL – считать/установить значения всех семафоров. GETVAL/SETVAL – одного семафора.
+Пример.

Билет 36. Сокеты. Типы сокетов. Коммуникационный домен. Схема работы с сокетами с установлением соединения.

Нужен унифицированный механизм, который позволял бы абстрагироваться от расположения процессов и давал бы возможность использования одних и тех же подходов для локального и нелокального взаимодействия и использования различных сетевых протоколов. Решение – сокеты. Сокеты делятся на **типы**, в зависимости от типа соединения:
Соединение с использованием виртуального канала. Последовательный поток байтов с гарантированной и корректной доставкой. Несетевой аналог – каналы. Границы сообщений при такой передаче не сохраняются, получатель должен сам делить поток. Поддерживает передачу экстренных сообщений вне основного потока.
*Датаграммное соединение. И*спользуется для передачи порций данных – дейтаграмм. Не гарантируется надежность, но соединение более быстрое.

При создании сокета указывается **коммуникационный домен**, к которому он будет принадлежать. AF_UNIX для локального взаимодействия и AF_INET для сетевого. Для первого формат адреса – допустимое имя, для второго – имя хоста + номер порта. Для первого – внутренние протоколы ос, для второго – протоколы семейства TCP/IP.

Создание и конфигурирование сокета. *Socket(domain, type, protocol)*, Первый аргумент – коммуникационный домен. Второй – SOCK_STREAM или SOCK_DGRAM. Третий – конкретный протокол. 0 – автоматически. IPROTO_TCP/IPROTO_UDP, если домен – AF_INET. Функция возвращает дескриптор сокета, в случае ошибки -1.
Связывание. Необходимо присвоить сокету адрес. Либо имя либо IP-адрес + номер порта. b*ind(sockfd, struct sockaddr *myaddr, addrlen),* первый аргумент – дескриптор сокета, второй – указатель на структуру, содержащую адрес сокета. Для локального домена структура sockaddr_un состоит из поля семейство (== AF_UNIX) и адрес. Для сетевого домена описана структура sockaddr_in с полями sin_family == AF_INET, sin_port – номер порта, struct in_addr sin_addr – IP адрес хоста, и пустые 8 байт. Последний аргумент bind – размер структуры. Если домен локальный, то создается файл с соответствующим именем, необходимо, чтобы такого файла не существовало. В случае успеха возвращает 0, неудачи –1.
Установление соединения. Различают сокеты с и без предварительного установления соединения. Если протокол – поток, то устанавливать соединение необходимо. *connect(sockfd, struct sockaddr *serv_addr, addrlen),* успех 0, иначе -1. Следующие 2 вызова используются сервером только при предварительном соединении. *listen(sockfd, int backlog),* сервер сообщает, что он готов принимать соединение. Второй аргумент – максимальный размер очереди запросов на соединение. Переполнение – зависит от протокола. *accept(sockfd, struct sockaddr *addr, addrlen)*, Этот вызов применяется сервером для удовлетворения поступившего клиентского запроса на соединение с сокетом, который сервер к тому моменту уже прослушивает. Извлекает первый запрос из очереди. Если очередь пуста, то блокируется. Создает новый сокет и возвращает его дескриптор, при этом старый сокет остается для обработки новых запросов. Так сервер поддерживает много соединений. Второй аргумент – указатель на структуру, в которой возвращается адрес клиентского сокета.
Прием и передача данных. Через сокет с предварительно установленным соединением: *send(sockfd, *msg, len, flags), recv(sockfd, *buf, len, flags).* Аргументы понятны. В случае успеха возвращают количество прочитанных/записанных байт. Флаги – MSG_OOB – для передачи экстренных сообщений. MSG_PEEK – для recv прочитать данные, не удаляя из сокета. Можно использовать read()/write(). Без установления соединения – sendto/recvfrom. Первые 4 аргумента аналогичны, пятый – адрес.
Завершение работы с сокетом. *shutdown(sockfd, mode),* mode = 0 – закрывается и для чтения, 1 – для записи, 2 – для чтения и записи. Дескриптор закрываем close().
Общая схема работы с сокетами. Сервер: socket -> bind -> listen -> accept -> send/recv -> shutdown -> close.(Плюс shutdown -> close для старого сокета). Клиент: socket -> bind -> connect -> send/recv -> shutdown -> close.

Билет 37. Сокеты. Схема работы с сокетами без установления соединения.

Нужен унифицированный механизм, который позволял бы абстрагироваться от расположения процессов и давал бы возможность использования одних и тех же подходов для локального и нелокального взаимодействия и использования различных сетевых протоколов. Решение – сокеты.

Создание и конфигурирование сокета. *Socket(domain, type, protocol)*, Первый аргумент – коммуникационный домен. Второй – SOCK_STREAM или SOCK_DGRAM. Третий – конкретный протокол. 0 – автоматически. IPROTO_TCP/IPROTO_UDP, если домен – AF_INET. Функция возвращает дескриптор сокета, в случае ошибки -1.
Связывание. Необходимо присвоить сокету адрес. Либо имя либо IP-адрес + номер порта. b*ind(sockfd, struct sockaddr *myaddr, addrlen),* первый аргумент – дескриптор сокета, второй – указатель на структуру, содержащую адрес сокета. Для локального домена структура sockaddr_un состоит из поля семейство (== AF_UNIX) и адрес. Для сетевого домена описана структура sockaddr_in с полями sin_family == AF_INET, sin_port – номер порта, struct in_addr sin_addr – IP адрес хоста, и пустые 8 байт. Последний аргумент bind – размер структуры. Если домен локальный, то создается файл с соответствующим именем, необходимо, чтобы такого файла не существовало. В случае успеха возвращает 0, неудачи –1.
Прием и передача данных. *Sendto(sockfd, *msg, len, flags, const struct sockadr *to, addrlen)/recvfrom().*Аргументы понятны. В случае успеха возвращают количество прочитанных/записанных байт. Флаги – MSG_OOB – для передачи экстренных сообщений. MSG_PEEK – для recvfrom прочитать данные, не удаляя из сокета.
Завершение работы с сокетом. *shutdown(sockfd, mode),* mode = 0 – закрывается и для чтения, 1 – для записи, 2 – для чтения и записи. Дескриптор закрываем close().
Общая схема работы с сокетами. socket -> bind -> sendto/recvfrom -> shutdown -> close.

Билет 38. Общая классификация средств взаимодействия процессов в ОС UNIX.

Элементарные средства: сигналы, неименованные и именованные каналы, трассировка.
Средства IPC: очередь сообщений, разделяемая память, массив семафоров.
Взаимодействие процессов в сети: сокеты.

???

Билет 39. Файловые системы. Структурная организация файлов. Атрибуты файлов. Основные правила работы с файлами. Типовые программные интерфейсы работы с файлами.

Файловая система - часть операционной системы, представляющая собой совокупность организованных наборов данных, хранящихся на внешних запоминающих устройствах, и программных средств, гарантирующих именованный доступ к этим данным и их защиту. Данные называются файлами.
Несколько разновидностей *структурной организации* файлов:
- Файл, как последовательность байтов. Просто набор данных без особой структуры. Выделение структуры – вопрос программы. Самая распространенная модель.
- Файл, как последовательность записей переменной длины. Каждая запись кроме содержательной информации должна иметь специальную информацию: длина записи либо маркер начала и конца записи. Небольшая фрагментация, хорошая эффективность хранения, но, чтобы добраться до какой-то записи нужно просмотреть предыдущие. Редактировать файлы – ад.
- Файл, как последовательность записей постоянной длины. Такая организация появилась из-за перфокарт, удобно писать порциями. Прямой доступ – достаточно эффективна по скорости, но фрагментация.
- Файл – дерево. В каждом узле – информация о записи (поле ключа и поле данных). Записи отсортированы по ключам. Данные произвольной длины, физически могут быть расположены как угодно. Имеются дополнительные расходы, связанные с древовидной организацией.

Атрибуты файла: имя, права доступа, персонификация (создатель, владелец), тип, размер записи, размер файла, указатель чтения / записи, время создания, время последней модификации, время последнего обращения, предельный размер, итд. Полный состав атрибутов файла и способ их представления определяется конкретной файловой системой.

Основные правила работы с файлами. ОС и ФС обеспечивают регистрацию возможности того или иного процесса работать с содержимым файлов. «Сеанс работы» с содержимым файла: «открытие» файла (регистрация в системе возможности работы процесса с содержимым файла). Создание внутрисистемной структуры данных, которая описывает состояние этого файла, проверяет права доступа, объявляет операционной системе тот факт, что с данным файлом будет работать тот или иной процесс. *Файловый дескриптор* – системная структура данных, содержащая информацию о актуальном состоянии «открытого» файла. Файловый дескриптор содержит актуальную информацию о открытом файле. Через ФД можно получить информацию о значении указателей чтения\записи. Далее работа с содержимым файла, с атрибутами файла. Завершение «закрытие» файла – информация системе о завершении работы процесса с «открытым» файлом. Операция закрытия файла имеет 2 вида: закрыть и сохранить текущее содержимое файла; уничтожить файл.

Типовые программные интерфейсы работы с файлами: open – открытие/создание файла, close – закрытие, read/write – чтение/запись относительно указателя из ФД, delete – удалить файл из ФС, seek – позиционирование указателей, rename – переименование, read/write attributes – чтение/модификация атрибутов.

Билет 40. Файловые системы. Модели реализации файловых систем. Понятие индексного дескриптора.

Файловая система - часть операционной системы, представляющая собой совокупность организованных наборов данных, хранящихся на внешних запоминающих устройствах, и программных средств, гарантирующих именованный доступ к этим данным и их защиту. Данные называются файлами. Некоторые модели:
-Одноуровневая ФС. В ФС существует один каталог, в котором находятся все файлы, находящиеся в системе. Проблемы: 1.Коллизия имен. Каждое имя должно быть единственно. 2. нагрузка на работу с системой, если много файлов. 3. неудобно структурировать.
-Двухуровневая ФС. В системе существует объединение каталогов пользователей, для каждого пользователя реализована одноуровневая модель. Проблемы 1 и 2 исчезают, а 3 остается.
-Иерархические ФС. За основу логической организации такой файловой системы берется дерево. В корне дерева находится, так называемый, корень файловой системы – каталог нулевого уровня. В этом каталоге могут находиться либо файлы пользователей, либо каталоги первого уровня. Каталоги первого и следующих уровней организуются по аналогичному принципу. Файлы пользователя в этом дереве представляются листьями. Пустой каталог также может быть листом. *Полное имя файла* – это путь от корневого каталога до листа (такой путь всегда будет уникальным). Существует также относительное именование, т.е. когда нет необходимости указания полного пути при работе с файлами. Это происходит в случае, когда программа вызывает файл и подразумевается, что он находится в том же каталоге, что и программа. В данном случае появляется понятие *текущего каталога*, т. е. каталога, на работу с которым настроена файловая система в данный момент времени. В рамках одного каталога имена файлов одного уровня должны быть разными.

ФС организует компактное хранение информации о размещении блоков файлов в структуре, называемой *индексным дескриптором*. Существует таблица индексных дескрипторов. При открытии файла осуществляется поиск по таблице и считывается только индексный дескриптор открытого файла. Достоинства: не надо размещать в памяти информацию обо всех файлах системы, только об открытых. Недостатки: размеры файла и индексного дескриптора. Решение: иерархическая организация индексных дескрипторов, ограничение размера файла.

Виды связи между именем и содержимым: *“Жесткая” связь*. Есть содержимое файла, есть атрибут файла, и одним из полей атрибутов является количество имен у этого файла, и есть произвольное количество имен, которые как-то распределены по каталогам ФС. В этом случае каждое из этих имен равнозначно, т.е. имеет место некоторая симметричная организация: каждое из имен синонимов равноправно, т.е. нет никакого старшинства в не зависимости от порядка образования этих файлов.
“Символическая” связь есть файл с именем Name2, этому имени соответствуют атрибуты и соответствует содержимое, и есть специальный файл Name1, который ссылается на имя Name2. В этом случае имеет место ассиметричное именование файлов. Имя Name2 позволяет организовывать более широкую работу с файлом, можно удалять, если файл будет удален, соответственно содержимое пропадет. Имя Name1 имеет свои правила интерпретации, поскольку они идут через ссылку на имя Name2,т.е. можно осуществлять доступ к содержимому, но если удалить файл Name1, то содержимое Name2 останется неизменным.

Билет 41. Файловые системы. Координация использования пространства внешней памяти. Квотирование пространства ФС. Надежность ФС. Проверка целостности ФС.

Одним из ресурсов ВС является лимитированное пространство ФС, выделенное для пользователя. Количество файлов ограничено сверху размером раздела. Определяется две квоты на этот ресурс: на блоки и на файлы. Существуют жесткие и гибкие лимиты. С *гибким лимитом* связывается 2 параметра: значение, непосредственно лимитирующее использование ресурса и счетчик предупреждений. Если используемое количество блоков или файлов больше, чем гибкий лимит, то пользователю дается предупреждение о превышении и счетчик предупреждений уменьшается на 1, работа пользователя блокируется, когда счетчик уменьшится до 0. *Жесткий лимит* – лимит, который нельзя перейти никак. Если пользователь попытается перейти жесткий лимит, то будет сразу блокирован.

Критерий надежности файловой системы: нужно обеспечить работу таким образом, чтобы потеря информации при возникновении внештатной ситуации была минимальной. Очевидное решение – резервное копирование. Резервное копирование должно проходить за ограниченный промежуток времени. *Модели резервного копирования:*
-Копируются не все файлы ФС. Не копируются объектники, временные файлы.
-Создается копия архива по расписанию. Копируются все файлы, которые были изменены с создания последней копии. Минус – очень большие копии.
-Использование компрессии при архивации. Риск потерять все файлы
-Распределение резервных копий. Лучше держать много копий в разных местах.
Возникает проблема архивирования «на ходу» - файлы могут измениться в процессе архивации.
Стратегии архивирования: Физическая: «один в один», только используемые блоки. Проблема архивации дефектных блоков. Логическая архивация – копирование файлов по какому-то критерию, например измененных после определенной даты.

Проверка целостности ФС. Проблема – при аппаратных или программных сбоях возможна потеря информации, как в обычных файлах, так и системной информации (списки блоков, индексные дескрипторы). Стратегия контроля:
-создаются 2 таблицы – занятых и свободных блоков.
-Анализируется список свободных блоков. Для каждого свободного блока увеличивается на 1 соответствующая запись в таблице свободных блоков.
-Анализ индексных дескрипторов. Для каждого блока в индексном дескрипторе увеличивается на 1 его счетчик в таблице занятых.
-Анализ содержимого таблиц и коррекция ситуаций.
Возможные результаты анализа таблиц: 1.Таблицы занятых и свободных блоков дополняют друг друга до всех единиц, тогда целостность системы соблюдена.
2. Пропавший блок – не числится ни среди свободных, ни среди занятых. Можно оставить как есть, и ждать претензий со стороны пользователя, но система замусоривается. Считаем свободным.
3. Таблица занятых блоков корректна, а какой-то из свободных блоков дважды или более раз посчитан свободным, т.е. список свободных блоков (таблица) не корректен. В этом случае нужно запустить процесс пересоздания списка свободных блоков.
4. Блок повстречался в 2-х индексных дескрипторах. Рассматриваем проблему на уровне файлов.

Билет 42. Примеры реализации файловых систем. Организация ФС ОС UNIX. Виды файлов. Права доступа. Логическая структура каталогов.

Файловая система ОС UNIX является примером многопользовательской иерархической файловой системой с трехуровневой организацией прав доступа к содержимому файлов. **Файл UNIX** – это специальным образом именованный набор данных, размещенный в файловой системе. Виды файлов:
•**обычный файл** – традиционный тип файла, содержащий данные пользователя. Интерпретация файла производится программой, обрабатывающей файл.
•**каталог** – файл, обеспечивающий иерархическую организацию файловой системы. С каталогом ассоциируются все файлы, которые принадлежат данному каталогу.
•**файл устройств** – система позволяет ассоциировать внешние устройства с драйверами и предоставляет доступ к внешним устройствам, согласно общим интерфейсам работы с файлами.
•**именованный канал** – специальная разновидность файлов, позволяющая организовывать передачу данных между процессами;
•**ссылка** – система позволяет создавать дополнительные ссылки к содержимому файла из различных точек файловой системы; Они могут нарушать древовидность организации ФС.
•**сокет** – средство взаимодействия процессов в пределах сети.

Категории пользователей: 1. пользователь (владелец). 2. группа (все пользователи, которые принадлежат группе владельца за исключением самого владельца) 3. все пользователи системы (все пользователи системы, за исключением группы владельца и самого владельца.)
Права: 1. на чтение. 2. на запись. 3. на исполнение (исполняемым файлом может быть только файл, полученный в результате сборки или командный файл).

Интерпретация этих прав зависит от типа файла. Так для обычных файлов это традиционные права на чтение, запись данных файла и исполнение содержимого файла в качестве процесса. Интерпретация прав доступа для других типов файлов может различаться. Например, для каталогов это: право на чтение – получение списка имен файлов; право на исполнение каталога – получение дополнительной информации о файлах, право на использование каталога в качестве текущего; право на запись – возможность создания, переименования и удаления файла в каталоге.

Все UNIX-системы имеют соглашения о логической структуре каталогов, расположенных в корне файловой системы. Это упрощает работу операционной системы, ее обслуживание и переносимость. **Корневой каталог /** является основой любой файловой системы ОС UNIX. Все остальные файлы и каталоги располагаются в рамках структуры, порожденной корневым каталогом, независимо от их физического положения на диске. Содержимое непосредственно корневого каталога: **/unix –** файл загрузки ядра ОС, **/bin –** общедоступные системные команды, **/etc –** файлы, определяющие настройки системы, **/tmp –** временные файлы. При перезагрузке не гарантируется сохранность содержимого. **/mnt –** монтирование дополнительных ФС, чтобы получить логически единое дерево. **/dev** – файлы устройств, **/lib –** библиотечные файлы, **/usr -** размещается вся информация, связанная с обеспечением работы пользователей. Подкаталоги: содержащий часть библиотечных файлов (/usr/lib), подкаталог /usr/home, который становится текущим при входе пользователя в систему, содержащий файлы заголовков (/usr/include).

Билет 43. Примеры реализации файловых систем. Внутренняя организация ФС. Модель версии UNIX SYSTEM V.

Структура ФС: Суперблок – область индексных дескрипторов – блоки файлов. *Суперблок* содержит оперативную информацию о текущем состоянии ФС: размер логического блока, размер файловой системы, размер области индексных дескрипторов, число свободных блоков, число свободных ИД, массив номеров свободных блоков, свободных ИД. В оперативной памяти постоянно находится актуальная копия суперблока.
Область индексных дескрипторов. Индексный дескриптор - структура данных, которая содержит: тип файла, права доступа, число имен, ассоциируемых с данным ИД, идентификатор владельца, размер файла, время последней модификации, массив номеров блоков.
Блоки файлов. Пространство на системном устройстве, в котором размещается вся информация, хранящаяся непосредственно в файлах.

Работа с массивом свободных блоков. Все свободные блоки ФС организованы в однонаправленный список. Первый элемент этого списка – массив из ссылок, который размещается в суперблоке. 0-й элемент этого массива – номер блока, в котором находится продолжение списка, итд. Освобождение блоков – добавляется в массив. Занятие – берется из массива.
Работа с массивом свободных ИД. Освобождаем ИД – добавляем в массив. Не помещается – забываем. Занимаем ИД – берем из массива. Нету – просмотр области ИД, выбор свободных. Массив свободных ИД – буфер.
Адресация блоков файла. Физически блоки файла могут быть расположены как угодно, но логически объединены в цепочку. В ИД находится массив, содержащий список 13 номеров блоков на диске. Первые 10 указывают на 10 блоков файла, если файл занимает более 10 блоков, то 11й элемент указывает на блок, содержащий до 128 адресов дополнительных блоков (если размер блока 512 байт, номер блока умещается в 4 байта), 12-й элемент – ссылка на блок, содержащий 128 указателей на блоки, в которых номера 128 блоков. 13 – еще больше. Получаем формулу для максимального размера файлов.
При *открытии файла* его ИД считывается в память и ОС становятся доступны номера всех блоков файла. Для файла открытого несколько раз в памяти только один ИД. Система фиксирует число открытий файла. Когда этот счетчик обнуляется, ИД записывается на диск.
Файл каталог представляет собой таблицу, каждая запись которой состоит из 16 байтов. Первые 2 байта – номер индексного дескриптора, 14 байт – имя файла. Первая строчка – ссылка на самого себя с именем «.», вторая строка – ссылка на родительский каталог с именем «..»

Имя файла «отделено от других атрибутов». Один файл можно внести в несколько каталогов, может иметь разные имена, но они будут ссылаться на один дескриптор. Связи нарушают древовидность. Жесткая (имена равноправны) и символическая связь (создается ссылка, счетчик количества имен не увеличивается).

Достоинства ФС SYSTEM V: оптимизирована работа со списками номеров свободных блоков и ИД, организация косвенной адресации блоков.
Недостатки: концентрация важной информации в одном месте – суперблоке. Проблема надежности, так как много ссылочных структур. Фрагментация файла по диску. Организация каталога ограничивает длину имени файла

Билет 44. Примеры реализаций файловых систем. Внутренняя организация ФС. Принципы организации ФС FFS UNIX BSD.

Основной идеей данной модели файловой системы является кластеризация дискового пространства файловой системы, с целью минимизации времени чтения/записи файла, а также уменьшения объема неиспользуемого пространства внутри выделенных блоков. Идея: дисковое пространство разделено на области одинакового размера – группы цилиндров. ФС старается разместить содержимое файлов в пределах одной группы цилиндров, при этом стараясь разместить файлы в ту же группу, что и каталог, в котором они расположен. Каждая группа цилиндров содержит копию суперблока и массив индексных дескрипторов. Стратегия размещения: Новый каталог помещается в группу цилиндров, число свободных ИД в которой больше среднего арифметического в ФС, а также имеющий минимальное число дескрипторов каталогов в себе. Для равномерности файл разбивается на несколько частей, первая часть располагается в той же группе цилиндров, что и его ИД, при размещении последующих частей используются группы цилиндров, в которой число свободных блоков больше, чем в среднем. Длина первой части выбирается так, чтобы она непосредственно адресовалась индексным дескриптором, остальные части имеют равный размер. Последовательные блоки файлов размещаются исходя из оптимизации физического доступа (выгоднее размещать не последовательно, а через один-два, так как диск проворачивается и иначе придется ждать полного поворота).
Блоки разбиваются на фрагменты. Формат индексного дескриптора аналогичен S5, основная единица – блок. Правило: все блоки в ИД, кроме последнего используются только для одного файла. То есть для хранения информации об использовании последнего блока необходимо хранить информацию о фрагментах этого блока. Для хранения информации о свободных фрагментах используется битовая маска. Использование фрагментов уменьшает фрагментацию.
Выделение пространства для файла происходит только в момент когда процесс выполняет системный вызов write:
1. Если в уже выделенном файлу блоке есть достаточно места, то новые данные помещаются в это свободное пространство.
2. Если последний блок файла использует все фрагменты и свободного в нём места недостаточно для записи новых данных, то частью новых данных заполняется всё свободное место. Если остаток данных превышает по размеру один полный блок, то выделяется новый полный блок итд, пока остаток не окажется меньше чем полный блок. В этом случае ищется блок с необходимыми по размеру фрагментами.
3. Файл содержит один или более фрагмент и последний фрагмент недостаточен для записи новых данных. Если размер новых данных в сумме с размером данных, хранимых в неполном блоке, превышает размер полного блока, то выделяется новый полный блок. Содержимое старого неполного блока копируется в начало выделенного блока и остаток заполняется новыми данными. Далее – пункт 2. В противном случае (если размер новых данных в сумме с размером данных, хранимых в неполном блоке, не превышает размер полного блока) ищется блок с необходимыми по размеру фрагментами или выделятся новый полный блок. Остаток данных записывается в этот блок.
Структура каталога – таблица с полями: номер ИД, размер записи, тип файла, длина имени файла, имя файла. Суть использования того и другого параметра заключается в том, что при удалении информации (какого-то имени) из каталога свободное пространство присоединяется к предыдущей записи и получается, что размер больше информации, которая имеется. Может появиться внутренняя фрагментация.

Билет 45. Управление внешними устройствами. Архитектура организации управления внешними устройствами, основные подходы, характеристики.

Непосредственное управление Внешними устройствами ЦП. В основном требуется переместить данные из ВУ в ОЗУ (и наоборот). ЦП по своей инициативе почти никогда не обращается к ОЗУ. Историческая модель основана на том, что управление осуществлялось с помощью ЦП. Когда говорится о том, что организовано управление внешним устройством, то подразумевается, что реализуется два потока информации: поток управляющей информации (команды) и поток данных, т.е. поток той информации, которая начинает двигаться от ОП к внешнему устройству, за счет потока управляющей информации. Оба потока обрабатывает ЦП, что «отвлекает» его от других задач пользователя.

Синхронное управление внешними устройствами с использованием контроллеров внешних устройств. В результате развития аппаратной части компьютера появляются контролеры внешнего устройства. Они упростили жизнь ЦП. Все равно поток команд идет через ЦП, контролер взял некоторые функции: обнаружение ошибок, обеспечение более высокоуровневого интерфейса по управлению ВУ, позволяет использовать команды типа «вывести головку на нужный сектор», появилось разделение функций синхронизации. ЦП подавал сигнал и ждал.

Асинхронное управление внешними устройствами с использованием контроллеров внешних устройств. Появление контролеров прямого доступа позволяет вывести поток данных, который появляется при обмене с ВУ из ЦП. Это имеет смысл для блок-ориентированных устройств, подразумевающих большой поток информации. Поток управляющей информации остается в ведении ЦП.

Управление внешними устройствами с использованием процессора или канала ввода/вывода. Наличие процессоров ввода-вывода позволяет обеспечить высокоуровневый интерфейс для ЦП при управлении внешними устройствами. ЦП предоставляются различные макрокоманды. («записать на диск…начиная с…места»)

Программное управление внешними устройствами. Цели, которые стоят перед программным обеспечением:
1. унификация программных интерфейсов доступа к внешним устройствам (унификация именования, абстрагирование от свойств конкретных устройств);
2. обеспечение конкретной модели синхронизации при выполнении обмена (синхронный, асинхронный обмен);
3. обработка возникающих ошибок (индикация ошибки, локализация ошибки, попытка исправления ситуации); корректно обработать эту ситуацию, минимизировать негативные последствия.
4. буферизация обмена – в системе очень многоуровневая, применяется на всех этапах: развитые каналы ввода-вывода могут иметь встроенный КЭШ, который управляется внутри этих каналов. Эта функция остается на уровне ОС, этот КЭШ ОС полностью программно ориентирован.
5. обеспечение стратегии доступа к устройству (распределенный доступ, монопольный доступ);
6. планирование выполнения операций обмена – возникает, когда возникает конкуренция за доступ к ресурсу.

Билет 46. Управление внешними устройствами. Планирование дисковых обменов, основные алгоритмы.

Планирование дисковых обменов. Возможна ситуация, когда поток заказов на обмен больше пропускной способности системы в некоторые моменты.
Тогда есть несколько вариантов действий: Принимаем решения о порядке обработки запросов / начинаем учитывать приоритеты / осуществляем случайный выбор.

Пусть наш диск может сразу переходить i-ой дорожки на j-ую без начального позиционирования. Пусть имеется очередь запросов к дорожками. Возможные варианты решения:
-простейшая модель – случайная выборка из очереди.
-FIFO
-SSTF. “Жадный” алгоритм. Приоритет имеет обмен, для которого потребуется наименьшее время.
-LIFO
-Приоритетный алгоритм (RPI) – это алгоритм, когда последовательность обменов (очередь) имеет характеристику приоритетов. При использовании приоритетных алгоритмов может возникать проблема голодания или дискриминации. Проблема дискриминации возникает при непрерывном поступлении более приоритетных запросов на обмен, в это время как менее приоритетные запросы простаивают.
-SCAN. Находясь в начальной позиции сначала двигаемся в одну сторону до конца, затем в другую до конца.
-C-SCAN. Выходим на минимальную/максимальную дорожку и движемся в одну сторону
-N-step-SCAN. Разделение очереди на подочереди. Последовательная обработка очередей. Распространенный пример: 2 очереди, одна обрабатывается, другая собирает вновь поступающие запросы.

Билет 47. Управление внешними устройствами. Организация RAID систем, основные решения, характеристики.

Существуют проблемы с организацией больших потоков данных. В общем случае для дисковых систем имеют место как минимум две проблемы:
1. Эффективность. Допустим, в системе присутствуют все уровни КЭШ, но производительности не хватает, так как обмены, которые производятся на дисковых устройствах, медленные.
2. Надежность. Является одним из основных качеств любого программного решения. Соответственно есть необходимость создания надежных дисковых систем.
Все это обусловило появление так называемых RAID систем. Избыточный массив независимых дисков. Диски рассматриваются ОС как единое дисковое устройство, где данные представляются в виде последовательности записей – полос. Семь уровней RAID систем:
RAID 0 (без избыточности). Данные распределяются по всем дискам массива. Лучше, чем один большой диск, так как появляется вероятность того, что два различных блока, к которым поступили запросы, размещены на различных дисках, и можно обработать два запроса параллельно. Логически – один диск. Достоинство: можно обработать параллельно «количество дисков» запросов.
RAID 1 (зеркалирование). Имеется 2 группы дисков. Вторая – копия первой. Достоинства: запрос может быть обработан любым из двух дисков, запись – обновление обеих полос, но выполняется параллельно и несложно; простота восстановления. Недостатки: дорогостоящая конструкция. Для хранения важной информации (системные диски).
RAID 2 (с кодами Хэмминга). Код Хэмминга исправляет одинарные и выявляет двойные ошибки. Полосы малы. Часть дисковых устройств содержательной информацией, несколько реализуют коды Хэмминга. При считывании одновременный доступ ко всем дискам. Данные запроса и код коррекции ошибок передаются контролеру массива. При наличии однобитовой ошибки контролер способен быстро ее откорректировать, так что доступ для чтения в этой схеме не замедляется. При записи - одновременное обращение ко всем дискам массива. Проблемы: избыточность меньше RAID 1, но все равно есть; зависимые обмены.
RAID 3 (четность с чередующимися битами). 4 диска содержательные, 5й – контрольная информация (XOR). Можно восстановить один диск.
RAID 4. Аналогично 3, но по полосам. Аппаратно проще.
RAID 5. (циклическое распределение четности). В предыдущих двух сильно нагружено контрольное устройство. Здесь контрольный диск циклично распределен по всем устройствам. Для приложений интенсивно использующих диск.
RAID 6. (двойная избыточность). Еще одно устройство контроля. Две схемы контроля. Исключительно высокая надежность.

Некоторые можно реализовать программно, некоторые только аппаратно.

Билет 48. Внешние устройства в ОС UNIX. Типы устройств, файлы устройств, драйверы.

Особенность UNIX - все устройства в системе обслуживаются в виде файлов. Взаимодействие с помощью иерархии драйверов. Единый интерфейс взаимодействия с ВУ – через файлы устройств из каталога /dev. Файл устройства позволяет ассоциировать некоторое имя с драйвером устройства.
В системе существуют два типа специальных файлов устройств:
- файлы байт-ориентированных устройств (драйверы обеспечивают возможность побайтного обмена данными и, обычно, не используют централизованной внутрисистемной кэш-буферизации);
- файлы блок-ориентированных устройств (обмен с данными устройствами осуществляется фиксированными блоками данных, обмен осуществляется с использованием специального внутрисистемного буферного кэша).
В системе поддерживается две таблицы драйверов. Следует отметить, файловая система может быть создана только на блок-ориентированных устройствах. Тип файла определяется свойствами конкретного устройства и организацией драйвера. Конкретное физическое устройство может иметь, как байт-ориентированные драйверы, так и блок-ориентированные.

Содержимое файлов устройств целиком находится в соответствующем индексном дескрипторе, который содержит: тип файла устройства, старший номер устройства – номер драйвера в таблице драйверов, младший номер – служебная информация, передающаяся драйверу.
Каждая запись таблицы драйверов содержит коммутатор устройства – структуру, в которой размещены точки входа драйвера. Типовой набор точек входа:
- bopen() открытие устройства, обеспечивается инициализация устройства и внутренних структур данных драйвера;
- bclose() закрытие драйвера устройства, например в том случае, если ни один из процессов не работает с драйвером;
- bread() чтение данных;
- bwrite() запись данных;
- bioctl() управление устройством, задание режимов работы драйвера, определение набора внутренних операций/команд драйвера;
- bintr() – обработка прерывания, вызывается ядром при возникновении прерывания в устройстве, с которым ассоциирован драйвер;
- bstrategy() управление стратегией организации блок-ориентированного обмена (некоторые функции оптимизации организации обмена, обработка специальных ситуаций, связанных с функционированием конкретного устройства и т.п.).

В системе возможно обращение к функциям драйвера в следующих ситуациях:
1. старт системы, определение ядром состава доступных устройств.
2. обработка запроса ввода/вывода;
3. обработка прерывания, связанного с данным устройством, в этом случае ядро вызывает специальную функцию драйвера;
4. выполнение специальных команд управления (например, остановка устройства, приведение устройства в некоторое начальное состояние и т.п.).
Два способа включения новых драйверов в систему: путем жесткого встраивания в код ядра и путем динамического встраивания (загрузка и динамическое связывание драйвера с кодом ядра, инициализация драйвера, формирование данных коммутатора устройства, связывание обработчика прерываний ядра с драйвером).

Билет 49. Внешние устройства в ОС UNIX. Системная организация обмена с файлами. Буферизация обменов с блок-ориентированными устройствами.

На практике чаще всего обращаемся к обычным файлам. Для организации интерфейса работы с файлами ОС использует информационные структуры и таблицы двух типов: ассоциированные с процессом и ассоциированные с ядром ОС.
Для каждого открытого в рамках системы файла формируется запись в **таблице индексных дескрипторов открытых файлов,** содержащая: копию индексного дескриптора открытого файла, кратность – счетчик открытых в системе файлов, связанных с данным ИД. Вся работа с содержимым открытого файла происходит с помощью копии ИД, размещенной в ТИДОФ. Эта таблица размещается в памяти ядра ОС. **Таблица файлов** содержит сведения о всех файловых дескрипторах открытых в системе файлов. Каждая запись содержит указатель чтения/записи. Правила соответствия между открытыми в процессах файлами и записями ТФ: При каждом обращении к функции открытия файлов таблице файлов появляется новая запись, то есть если будет многократно открыт файл, то в каждом случае определяется новый файловый дескриптор со своими указателями чтения/записи. Если файловый дескриптор образуется в результате наследования, то новой записи не появляется, а увеличивается счетчик наследования. Таблица в памяти ОС. С каждым процессом связана **таблица открытых файлов.** Номер записи в данной таблице – номер ФД, который может использоваться в процессе. Каждая строка этой таблицы имеет ссылку на запись в таблице файлов. Первые три строки этой таблицы используются для файловых дескрипторов стандартных устройств ввода-вывода.

Особенностью работы с блок-ориентированными устройствами является возможность организации буферизации при обмене. В RAM организуется пул буферов, каждый буфер имеет размер в один блок. Каждый из блоков может быть ассоциирован с драйвером одного из физических блок-ориентированных устройств.
«+» оптимизация и минимизация обмена с реальными устройствами.
«-» Существенная критичность к несанкционированному выключению машины
«-» проблемы разорванности во времени операции записи
Пусть поступил заказ на чтение блока из устройства. Среди буферного пула осуществляется поиск блока, если находится, то обращения к физическому устройству не происходит. Обнуляем счетчик времени в буфере, в остальных увеличиваем на 1. Если не нашли, то ищем какой можно заменить. Если есть свободный буфер(возможно только при старте системы), то фиксируем его, иначе смотрим, к какому буферу долго не было обращений, если он изменялся, то записываем его. Считываем в этот буфер в нужный блок, меняем счетчики времени, получаем результат.
Преимущества и недостатки выше. Система должна работать на надежной аппаратуре. В системе имеется параметр, который может оперативно меняться и определяет периоды времени, через которые осуществляется сброс системных данных. Также имеется команда, доступная пользователю, по сбросу данных на диск. Плюс система обладает избыточностью, позволяя в случае потери информации произвести действия, которые восстановят некоторые данные.

Билет 50. Управление оперативной памятью. Одиночное непрерывное распределение. Распределение разделами, перемещаемыми разделами.

Основные задачи управления ОП: *Контроль состояния единицы памяти* (свободна/распределена). Эта функция обеспечивается аппаратно и программными средствами ОС. *Стратегия распределения памяти.* Правила, по которым будет распределена память. *Выделение памяти.* Принятие решения о выделении конкретного объема памяти. *Стратегия освобождения памяти.* Принятие решений об отборе памяти навсегда или на время.

Стратегии управления ОП:
Одиночное непрерывное распределение. ОП делится на две области. В одной находится ОС, другая для задач пользователя. Необходимые аппаратные средства: регистр границ, режим ОС/режим пользователя. Прерывание, если от пользователя обращаемся в область ОС. Алгоритм – процесс завершился, меняем на следующий. Достоинства: простота. Недостатки: часть памяти не используется, вся память резервируется на все время выполнения, ограничения на размеры процесса.

Распределение неперемещаемыми разделами. Есть ОС и оставшаяся физическая память. Оставшуюся физическую память делим на конкретное количество разделов. В каждом может быть свое задание и свой процесс. Внутри каждого раздела все аналогично рассмотренному выше. Необходимые аппаратные требования: 2 регистра границ. Алгоритм – вся очередь процессов разбивается на очереди, с каждой из которых связан свой раздел. Процесс размещается в разделе минимального размера, достаточного для размещения данного процесса. Достоинства: простая организация мультипрограммирования, минимальная аппаратная поддержка, простые алгоритмы. Недостатки: может возникнуть ситуация, когда очередь больших процессов пуста, а маленьких очень много, и мы не сможем перегрузить. Размер процесса ограничен максимальным размером раздела.

Распределение перемещаемыми разделами. Система имеет фиксированное количество разделов. Через некоторое время ее использования начинается внешняя фрагментация. Решение: перемещение разделов и освобождение одного большого куска. Но это требует очень больших затрат. Необходимые аппаратные средства: Регистры границ + регистр базы. Алгоритм - аналогично предыдущему. Достоинства: потенциальная ликвидация внешней фрагментации. Недостатки: внутренняя фрагментация. Ограничение размером физической памяти. Затраты на перекомпоновку. Операция освобождения одного большого куска ОП очень тяжела.

Билет 51. Управление оперативной памятью. Страничное распределение.

В ОС есть таблица страниц, определяющая соответствия виртуальной памяти физической для выполняющегося процесса. При замене процесса таблицу надо менять. Проблемы: размеры таблицы страниц, скорость отображения. Аппаратные средства: полностью аппаратная таблица страниц в сверхоперативной памяти, но это дорого / регистр начала таблицы страниц в памяти / гибридные решения (программно + аппаратно). Решение проблемы размера таблицы – иерархическая организация. Содержимое каждой записи – информация о виртуальной странице. Поля: присутствие/отсутствие, если этот признак установлен, то в поле «номер физической страницы» находится нужная страница, если нет, то либо доступ запрещен, либо страница откачана, в любом случае – прерывание; поле защиты, при обращении к таблице страниц знаем с какой целью обращаемся, если незаконное обращение – узнаем по этому полю; признак изменения, будет установлен, если писали в эту страницу; признак блокировки кэширования.
Для разрешения всех проблем со скоростью, размерами используются гибридные решения. Одно из них основывается на TLB буферах. TLB (Translation Lookaside Buffer) – Буфер быстрого преобразования адресов. В процессоре есть буфер (не большой), который используется в качестве КЭШ таблицы страниц. Структура буфера: Каждая запись содержит 2 поля - № виртуальной страницы и № физической страницы. TLB буфер – буфер оперативной памяти. Поиск идет параллельно: за одну операцию просматривается наличие всей таблицы. Мы имеем виртуальный адрес, в котором традиционно есть поле: «виртуальная страница» и есть поле «смещение». Процессор выбирает поле «виртуальная страница» и обращается к TLB буферу. Если мы фиксируем факт попадания, то в этом случае автоматически происходит замена поля виртуальной страницы на содержимое поля физической страницы – так мы получили физический адрес со всеми вытекающими параметрами, которые могут находиться в TLB. Если мы фиксируем промах, то в этом случае происходит прерывание, управление передается ОС. И ОС уже программно находит необходимую строчку и обновляет TLB буфер и соответственно дообрабатывает команду преобразования виртуального в физический.
Многоуровневая организация. Идея проста. Виртуальный адрес состоит из нескольких индексов по таблицам.
Использование хэш таблиц. Хэш функция по номеру виртуальной страницы определяет номер записи хэш таблицы, по которой получим физическую страницу.
Инвертированные таблицы страниц. Используются в более развитых системах, системах аппаратно поддерживающих pid обрабатываемого процесса. Каждая строка таблицы соответствует конкретной физической странице. Проблема – поиск по таблице.
Проблема замещения страниц. Нужно загрузить новую страницу, а места в памяти нет. Нужно выбрать страницу для удаления. Варианты: NRU – вытеснить не использовавшуюся последнее время. Используются биты обращения и модификации. FIFO – удаляем самую «старую» страницу. Может быть несправедливо. Модификация – алгоритм «вторая попытка» - выбираем самую старую, если используется, то перемещаем в конец очереди. LRU – имеется N страниц, составляем битовую матрицу NxN. При обращении к i-й странице присваиваем 1 всем битам i-й строки и обнуляется i-й столбец. Строка с наименьшим двоичным числом – нужная страница. NFU – для каждой физической страницы программный счетчик. По таймеру к нему прибавляется признак доступа. В момент принятия решения выбираем страницу с наименьшим счетчиком.
+ нет фрагментации + не ограничены размером ОП – проблема выбора.

Билет 52. Управление оперативной памятью. Сегментное распределение.

Основные задачи управления ОП: *Контроль состояния единицы памяти* (свободна/распределена). Эта функция обеспечивается аппаратно и программными средствами ОС. *Стратегия распределения памяти.* Правила, по которым будет распределена память. *Выделение памяти.* Принятие решения о выделении конкретного объема памяти. *Стратегия освобождения памяти.* Принятие решений об отборе памяти навсегда или на время.

Сегментное распределение. Основные концепции:
•Виртуальное адресное пространство представляется в виде совокупности сегментов.
•Каждый сегмент имеет свою виртуальную адресацию (от 0 до N-1)
•Виртуальный адрес: <номер сегмента, смещение>
Необходимые аппаратные средства: таблица сегментов, по которой при вычислении физического адреса из виртуального мы можем индексироваться по номеру сегмента. Соответственно каждая запись таблицы сегментов содержит размер сегмента и адрес начала сегмента.
«+» простота реализации
«+» размер таблицы сегментов может быть много меньше размера таблицы страниц
«-» наличие внешней фрагментации
«-» сегмент рассматривается как единое целое

Преобразование происходит достаточно просто: мы индексируемся по таблице, получаем запись, после этого сравниваем смещение с размером сегмента: если смещение выходит за пределы размера – происходит прерывание, иначе мы значению базы прибавляем смещение и получаем физический адрес.

Билет 53. Кэширование информационных потоков на уровне аппаратуры и ОС.

ЩИТО???

Билет 54. Язык программирования C. Общая характеристика. Типы, данные, классы памяти. Правила видимости. Структура программы. Препроцессор. Интерфейс с ОС UNIX.

Ну ты понел.. (c)