**Лекция 1**

***Системы пакетной обработки*** – предназначены для решения задач,

в основном вычислительного характера, не требующих быстрого получения результатов. Максимальная производительность ЭВМ достигается путем минимизации простоя ее устройств.

***Системы разделения времени*** – предоставляют возможность интерактивной работы сразу с несколькими приложениями. Всем приложениям попеременно предоставляется квант времени центрального процессора. Если время кванта выбрано достаточно маленьким, то у пользователя складывается ложное впечатление об одновременной работе с несколькими приложениями. T кванта = 0,02 мс

***Система реального времени*** – предназначена для управления техническими объектами, технологическими процессами и системами обслуживания разного рода.

Для всех этих систем существует предельно допустимое время, в течение которого должна быть выполнена та или иная программа. Критерием эффективности таких ОС является способность выдерживать заранее заданные интервалы времени между запуском программы и получением результата. Это время называют **временем реакции системы**. Такие системы должны обладать запасом вычислительной мощности на случай пиковой загрузки, а также обладать повышенной надежностью.

***Мультипроцессорные системы*** – способ организации вычислительного процесса в системе с несколькими процессорами, при котором несколько задач могут выполняться на разных процессорах систем.

Эти системы делят на **симметричные и ассиметричные**.

***Симметричная архитектура*** предполагает ?? и большую, единую, разделяемую между процессорами память. **Масштабируемость** (возможность наращиваемости числа процессоров) ограничена размерами памяти, а также требованиям расположения процессоров и памяти в одном корпусе. Этот подход называют **масштабированием по вертикали**, он ограничивает число процессоров до 4-8 штук.

Операционная система реализует принцип **симметричного мультипроцессирования**, так как является общей для всех процессоров системы. Все процессоры участвуют как в управлении вычислительным процессом, так и в решении прикладных задач. Возможны случаи применения нескольких процессоров для решения одной задачи.   
Система полностью **децентрализована** и ее модули могут выполняться на любом процессоре, поэтому при отказе одного или более процессора система снижает производительность, но продолжает работу.

***Ассиметричная архитектура*** позволяет объединять различные по характеристикам и функциональной роли процессоры. Это снимает требования одного корпуса и единой памяти. Операционная система располагается на главном процессоре и является **централизованной**. Именно она распределяет задачи управления и прикладные задачи по остальным процессорам. Масштабирование называют **горизонтальным,** а саму систему **кластерной**. Управление организуются по принципу **ведущий-ведомый**, где все управление сосредоточено на главном процессоре.

**Лекция 2**

***Система управления процессами и потоками***

**Функциями** этой системы являются:

1) Создание процессов

2) Обеспечение процессов и потоков необходимыми ресурсами

3) Изоляция

4) Планирование выполнения процессов и потоков

5) Диспетчеризация потоков

6) Синхронизация процессов и потоков

7) Завершение и уничтожение процессов и потоков

! Независимые процессы не могут создавать друг друга

***События, приводящие к созданию процессов:***

1. Инициализация (загрузка) ОС
2. Запрос процесса на создание дочернего процесса
3. Запрос пользователя на создание процесса (например, при входе в систему в интерактивном режиме)
4. Инициирование пакетного задания
5. Создание операционной системой процесса какой-либо службы

Для того, чтобы процессы не могли вмешиваться в распределение ресурсов, а также не могли повредить коды и данные друг друга, ОС выполняет задачу изоляции процессов. Для этого ОС обеспечивает каждый процесс отдельным виртуальным адресным пространством, так что ни один процесс не может получить прямого доступа к командам и данным другого процесса.

Переход от одного процесса к другому выполняется при помощи **диспетчеризации**. Работа по определению того, в какой момент времени необходимо прервать текущий процесс и в какое время передать возможность выполнения называется **планированием**. Планирование осуществляется на основе информации, хранящейся в **написателях** процессов и потоков.

***Диспетчеризация*** заключается в реализации найденного в результате планирования решений.

Диспетчеризация делится на **три этапа**:

1. Сохранение контекста выбранного потока
2. Загрузка контекста потока, выбранного в результате планирования
3. Запуск нового потока на выполнение

Каждый раз процесс завершается в результате **одного из событий**:

1. Обычный выход по достижению конца процесса
2. Выход по ошибке
3. Выход по неисправимой ошибке
4. Уничтожение другим процессом

ОС сама убирает следы пребывания процесса в ЭВМ –

закрывает все файлы, с которыми работал процесс, освобождает по возможности области оперативной памяти, выгружает ненужные библиотеки, выполняет коррекцию всевозможных информационных таблиц ОС, контролирующих выполнение распределение всех ресурсов

***Модели процессов и потоков:***

**Создать процесс** – это в первую очередь создать его описание, включающий несколько информационных структур, содержащих все сведения о процессе, необходимые ОС для управления им:

- идентификатор процесса

- адрес расположения в памяти исполняемого модуля

- привилегии процесса, приоритеты и права доступа

- загрузка входа и данных исполняемой программы с диска в оперативную память. Для этого процессу выделяется его адресное пространство и формируется стек, с помощью которого определяются вызовы процедур и передача параметров

**Лекция 3**

Множество, в которое входит программа, данные, стеки и атрибуты процессов называют **образом процесса**.

При управлении процессами ОС использует два основных типа управления процессом: **блок управления процессом (дескриптор), контекст**

**Дескрипторы** процессов объединяются в таблицу процессов, размещенную в области ядра. На основании этой информации ОС осуществляет планирование и синхронизацию процессов.

В **дескрипторе** содержится такая информация о процессе, которая необходима ядру в течение всего жизненного цикла процесса. В зависимости от того, находится ли он в оперативной памяти или на диске, он является ??

***Дескриптор процесса содержит:***

1. *Информацию по идентификации процесса* (идентификатор процесса, пользователя, родительского и дочерних процессов).
2. *Информацию по состоянию процессора*
3. *Информация по приоритету и правах доступа*

***Информация по состоянию и управлению процессом***

Скриншот

***Контекст процесса***

Содержит информацию, позволяющую ОС приостанавливать и возобновлять процесс с прерванного места:

1. Содержимое регистров процессов, доступных пользователю (обычно 8-32 и до 100 регистров в RISC)
2. Содержимое счетчика команд
3. Состояние управляющих регистров и регистров состояния
4. Коды условия, отражающие результат выполнения последней арифметической или логической операции (например, равенство нулю, переполнение)
5. Указатели вершин стеков, хранящие параметры и адреса вызова процедур и системных служб

**Лекция 4**

Обычно модели обслуживания очередей ??. Так как ЦП работает намного быстрее выполнения операции ввода-вывода вскоре после начала работы все находящиеся в памяти процессоры окажутся **в состоянии ожидания операции ввода-вывода**. Простое увеличение объема оперативной памяти для большего размещения в ней процессов не позволит решить данную проблему, так как возрастает стоимость системы в целом, а во-вторых аппетиты программистов в использовании памяти возрастают пропорционально ее объемам, что приведет к увеличению размера процессора, а не просто их количеству.

Выходом из данной ситуации является **создание второй очереди**, в которую будут перемещаться процессы, выполнение которых по какой-то причине невозможно.

При создании потоков ОС также генерирует специальную информационную структуру – **описатель потока (дескриптор)**. Он содержит идентификатор потока, данные о правах доступа и приоритете, о состоянии потока и другое.

**Описатель потока** условно делят на две части:

1. Атрибуты блока управления
2. Контекст потока

Есть **два** способа реализации **пакета потоков**:

1. В пространстве пользователя (ядро их видеть не будет, в этом случае необходима таблица потоков: счетчик команд, управляющие регистры и регистры состояний
2. В ядре

Когда поток **в состоянии паузы или блокировки** вся информация необходимая для запуска хранится **в таблице потоков**. По умолчанию приложение всегда начинается с **одного потока**. Такое приложение с вместе составляющим его потоком размещается в процессоре.

В последующем выполняющийся поток может **породить** **новый поток**, который будет выполняться в пределах того же процесса. **Новый поток** создается с помощью вызова специальной программы из библиотеки.

**Лекция 5**

***Поток на уровне пользователя***

Все события, связанные с управлением процессом, проходят в пространстве пользователя в рамках одного процесса. **Ядро** не подозревает об этой деятельности и продолжает осуществлять управление процессом как единым целым.

**Достоинства:**

1. Можно реализовать в ОС, не поддерживающей потоки без каких-либо изменений в ОС
2. Высокая производительность, поскольку процессу не нужно переключаться в режим ядра и обратно
3. Ядро о потоках ничего не знает и управляет однопоточными процессами
4. Имеется возможность использования любых алгоритмов планирования потоков с учетом их специфики
5. Управление потоками возлагается на программу пользователя

**Недостатки:**

1. Системный вызов блокирует не только работающий поток, но и все потоки того процесса, к которому он относится
2. Приложение не может работать в многопроцессорном режиме, так как ядро закрепляет за каждым процессором только один процесс
3. При запуске одного потока ни один другой поток в рамках одного процесса не будет запущен пока первый добровольно не отдаст процессор
4. Внутри одного потока нет прерываний по таймеру, в результате чего невозможно создать планировщик по таймеру для поочередного выполнения потоков

***Поток на уровне ядра***

?? вместо этого строится **единая таблица потоков**, отслеживающая все потоки в системе, расположенная на **уровне ядра**. Если потоку необходимо создать новый поток или завершить имеющийся, он выполняет **запрос ядра**, который создает или завершает поток, **внося изменения** в таблицу потоков.

**Любое приложение** можно создать как **многопоточное**, при этом все потоки приложения поддерживаются в **рамках единого процесса**. Ядро поддерживает информацию **контекста процесса**, **как единого целого**, а также контексты **каждого отдельного потока**. Планирование осуществляется ядром исходя из состояния потока.

**Достоинства:**

1. Возможно планирование работы нескольких потоков одного итого же процесса на нескольких процессорах
2. Реализуется мультипрограммирование в рамках всех процессоров (в том числе одного)
3. При блокировании одного из потоков ядро может выбрать другой поток этого же или другого процесса
4. Процедуры ядра могут быть многопоточными

**Недостатки:**

1. Необходимость двукратного переключения режима пользователь-ядро, ядро-пользователь для передачи управления от одного потока к другому в рамках одного и того же процесса

***Планирование заданий, процессов и потоков***

**Основная цель** планирования вычислительного процесса **заключается в распределении времени** между выполняющимися заданиями в соответствии с требованиями пользователя.

Все виды планирования делятся на   
**долгосрочное, среднесрочное, краткосрочное** и **планирование ввода-вывода**

В большинстве ОС планирование осуществляется динамически, то есть решения принимаются на основе анализа текущей ситуации, не имея никаких предположений о том, какие задачи будут запущены пользователем в ближайшее время. Поэтому найденное **оперативное решение** в таких случаях не может быть **оптимальным**

**Лекция 6**

??

[**Другой тип планирования** основан на использовании специализированных систем с заданным набором решения задач и способом их решения.](https://www.google.ru/search?newwindow=1&espv=2&biw=929&bih=927&q=%D0%94%D1%80%D1%83%D0%B3%D0%BE%D0%B9+%D1%82%D0%B8%D0%BF+%D0%BF%D0%BB%D0%B0%D0%BD%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F+%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%B0%D0%BD+%D0%BD%D0%B0+%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B8+%D1%81%D0%BF%D0%B5%D1%86+%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC+%D1%81+%D0%B7%D0%B0%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D0%BC+%D0%BD%D0%B0%D0%B1%D0%BE%D1%80%D0%BE%D0%BC+%D1%80%D0%B5%D1%88%D0%B5%D0%BD%D0%B8%D1%8F+%D0%B7%D0%B0%D0%B4%D0%B0%D1%87+%D0%B8+%D1%81%D0%BF%D0%BE%D1%81%D0%BE%D0%B1%D0%BE%D0%BC+%D0%B8%D1%85+%D1%80%D0%B5%D1%88%D0%B5%D0%BD%D0%B8%D1%8F.&spell=1&sa=X&ved=0ahUKEwjt87m1z_3SAhVoYJoKHSd5AXcQvwUIFygA) В этом случае переходят к **статическому** **решению** задач, когда планировщик приступает к решению **заранее**. Для этого создана информационная таблица, которая указывает какому процессу, когда и на какое время должен быть предоставлен ЦП. При этом сопутствующие расходы на такое планирование, на исполнение составленного расписания **существенно меньше**, чем при динамическом планировании.

?? Он вызывается при наступлении события, которое может предоставить текущий процесс и предоставить возможность прекратить выполнение данного процесса в пользу другого.

Такими событиями **могут быть:**

1. Прерывание таймера
2. Вызов ОС
3. Специальными сигналами, связанными с программным кодом, созданные программистом

**Среднесрочное планирование** является частью системы **своппинга**, зависящей от состояния процесса, позволяющий при необходимости выгружать процесс на диск (если он по какой-то причине заблокирован на длительное время) или наоборот загружать его обратно в оперативную память при наступлении события, возвращающего ему активность

В общем случае **диспетчеризация сводится к следующему:**

1. Сохранение контекста текущего потока, который нужно сохранить
2. Загрузка контекста нового потока, выбранного на выполнение
3. Запуск нового потока

?? ***Состояния потока***

1. Выполнение – активное состояние потока, во время которого он обладает всеми необходимыми ресурсами и непосредственно обрабатывается на ЦП
2. Ожидание – пассивное состояние, находясь в котором поток заблокирован по своим внутренним причинам. Например, ждет наступления события, освобождающего необходимые ресурсы.
3. Готовность – пассивное состояние, но в этом случае поток заблокирован по внешним причинам (имеет все требуемые ресурсы, готов выполняться, но процессор занят обработкой другого потока)

??

Потоки образуют **очереди ожидающих и готовых процессов**, которые ведутся с помощью **объединения в списки описателей отдельных потоков.**

*Все множество алгоритмов принято делить на два небольших кластера:*

1. **Не вытесняющие алгоритмы**

Основаны на том, что активному потоку позволяется выполняться, пока он сам не отдаст управление ОС для того, чтобы она выбрала из очереди готовый к выполнению поток.

* Планирование распределяется между ОС и прикладными программами
* Необходимость частых передач управлений ОС, что позволяет избежать возможности избежать монопольного захвата ЦП приложением
* Зависания приложений могут привести к краху системы

1. **Вытесняющие**

Способ планирования потоков при котором решение о переключении процессора с одного потока на другой принимает ОС.

* Функции планирования сосредоточены в ОС
* Планирование на основе квантования процессорного времени
* Планирование на основе приоритетов потоков: статических, динамических, абсолютных, относительных, смешанных

В настоящее время доминируют **вытесняющие алгоритмы**. В них смена активного потока происходит если:

1. Поток завершается и покинул систему
2. Ошибка взаимодействия
3. Поток перешел в состояние ожидания
4. Исчерпан квант времени

***Особенности функционирования вытесняющих алгоритмов:***

1. Переключение контекстов потоков связано с потерей процессорного времени, которое не зависит от величины кванта, но зависит от частоты переключения, поэтому, чем больше квант, тем меньше суммарные затраты времени на переключения
2. С увеличением кванта времени ухудшается качество обслуживания пользователей, так как растет время реакции системы
3. ОС не знает ничего о характеристиках решаемой задачи, поэтому обслуживание происходит механически и основано на истории существования потоков в системе

Алгоритмы **приоритетного планирования**

А) в основе большинства алгоритмов лежат приоритеты ?? обслуживания. Это предполагает наличие у потоков некоторой начально известной характеристики – **приоритета.** Чем выше приоритет, тем меньше поток будет простаивать в очередях.

Б) ??

**Лекция 8**

***Виртуальная память***

На диск загружаются **неактивные процессы,** ожидающие освобождение каких-либо ресурсов, в том числе **кванта времени**. Когда подойдет очередь выполнения выгруженного процесса, при условии наличия всех необходимых ресурсов, он будет возвращен в оперативку.

Если обнаруживается, что свободного места для загрузки процесса в ОП **не хватает**, средствами ОС на диск выгружается **другой процесс**, который в настоящее время   
**не обслуживается** ЦП.

Подмена ОП дисковой называется **виртуализацией** и позволяет повысить уровень **мультипрограммирования**, так как объем оперативной памяти уже не так жестко ограничивает число одновременно выполняемых процессов.

Виртуализация ОП осуществляется совокупностью аппаратных и программных средств автоматически **без участия программиста.**

**Виртуализация** возможна на основе **двух** подходов:

1. ***Своппинг*** - образы на диск выгружаются и загружаются целиком

Достоинства своппинга:

- малые затраты времени на преобразование адресов в кодах программы

Недостатки своппинга:

- избыточность перемещаемых данных

- замедление работы системы

- неэффективное использование памяти

- невозможность загрузить процесс, адресное пространство которого превышает объем свободной оперативной памяти

1. ***Виртуальная память*** – между ОП и диском перемещаются части образов процессов (сегменты, страницы и т.д.)

Достоинства виртуальной памяти:

- снижение времени на копирование между диском и ОП

Недостатки виртуальной памяти:

- необходимость преобразования виртуальных адресов в физические

- сложность аппаратной и программной поддержки

***Функции ОС по управлению памятью:***

В **однопрограммных ОС** ОП делится на две части – одна для ОС, вторая для программ.

**1 Мб:**

* BIOS
* Скрытая память
* **Программа пользователя**
* ОС в ОЗУ

В **многопрограммных ОС** пользовательская часть должна предоставляться не одной программе, а нескольким. Эта задача выполняется ОС динамически с помощью специальной подсистемы управления памятью

**1 Мб:**

* BIOS
* Скрытая память
* **Куча программ пользователя**
* ОС в ОЗУ

Для МПОС **функции** ОС по управлению памяти существенно **усложнились**:

1. Отслеживание и учет свободной и занятой памяти
2. Первоначальное и динамическое распределение памяти процесса приложений и самой ОС
3. Освобождение памяти при завершении процессов
4. Настройка адресов программы на конкретную область физической памяти
5. Полное или частичное вытеснение кодов и данных процессов из ОП на диск, когда размеры ОП ограничены
6. Защита памяти, выделенной процессу, от возможных вмешательств со стороны других процессов
7. Дефрагментация памяти

Для **идентификации переменных и команд** на разных этапах жизненного цикла команды используются **символьные имена, виртуальные (математические, условные, логические) и физические адреса**:

1. **Символьные имена** – идентификаторы переменных в программе на алгоритмическом языке
2. **Виртуальные имена** - так как во время трансляции неизвестно какое место оперативной памяти будет загружено, транслятор присваивает переменным и командам виртуальные адреса, считая по умолчанию, что начальным адресом программы будет нулевой адрес
3. **Физические адреса** – номера ячеек физической памяти

Виртуальные имена

Транслятор

Физические адреса

Символьные имена

**Лекция 9**

**Совокупность виртуальных адресов процессов называется виртуальным адресным пространством.**

Существуют два принципиально разных способа преобразования виртуальных адресов в физические:

1. **Перемещающий загрузчик** (статическое преобразование) – в этом случае преобразование выполняется **один раз** для каждого процесса **во время его загрузки в память** **процессора.** Преобразование выполняет **перемещающий загрузчик** на основе имеющихся у него данных о начальном адресе, физической памяти, в которую будет загружена информация, а также информация, предоставляемая транслятором об адреснозависимых элементах (смещение относительно стартового адреса)
2. **Динамическое преобразование** – программа загружается в память   
   **в** **виртуальных адресах**. Во время обращения к памяти каждый раз будет происходить преобразование ОС виртуальных адресов в физические

***Распределение памяти***

- без использования внешней памяти

- с использованием внешней памяти

**Без использования внешней памяти:**

При использовании этого метода применяют **три подхода** оформления разделов:

**В первом случае** используются разделы **одинакового размера**. В этом случае любой процесс, **не превышающий размер** раздела может быть загружен в **любой** доступный раздел. **Если все разделы заняты** и нет ни одного процесса в состоянии готовности, ОС может выгрузить процесс из любого раздела и загрузить туда **другой процесс** готовый к работе, тем самым обеспечивая ЦП работой. Так как все разделы имеют **одинаковый размер** все процессы попадают в первый попавшийся раздел.

При использовании этого подхода выделяют две сложности:

1. Программа может быть **слишком велика** для размещения в разделе и тогда погромист должен писать программу так, чтобы в любой момент времени использовался один раздел памяти, то есть он должен уметь делать **оверлэи**
2. Память используется **неэффективно**. Любая программа, независимо от ее размера занимает весь **раздел целиком**, при этом могут оставаться участки незанятой памяти значительного размера. Этот феномен называют **внутренней фрагментацией.**

Простейшим способ выхода из этого положения заключается в создании фиксированных разделов разного размера. В этом случае есть **два подхода** назначению разделов запускаемого приложения:

А) для каждого раздела существует **своя очередь** - каждый процесс размещается в **наименьшем разделе** способном вместить данный процесс, **даже если там очередь.** Для каждого процесса требуется очередь планировщика. **Достоинством** этого подхода является **минимизация** внутренней **фрагментации**, а **недостатком** – высокая вероятность **несбалансированности очередей**,что приведет к тому, что какие-то разделы будут постоянно заняты, а какие-то не использованы вообще**.**

Б) использование **одной очереди** – в момент, когда необходимо загрузить процесс в память выбирается наименьший доступный раздел, способный вместить его. Усложняется аналитическая деятельность планировщика, но возрастает эффективность памяти в целом. **Достоинства** – простота, минимальные требования к ОС. **Недостатки** – количество разделов, определенных во время генерации ОС, ограничивает число активных процессов