# Операционные системы Лекция 3

Основная задача ОС – управлять процессами и выделять процессам ресурсы, при этом к ресурсам системы относятся любая компонента как аппаратного, так и программного обеспечения. Два основных ресурса это процессорное время и объём физической памяти. … и реентерабельные коды самой ОС. (реентерабельные коды, по-другому: код чистой процедура – это процедура которая не модифицирует саму себя. Модификация – в коде можно изменять данные, поэтому)

Данные реентерабельных кодов, находится в системных “таблицах” (Таблица – условно). Массивы требуют дополнительных действий – поэтому в ОС используются связные списки т.к. изменение данных это только изменение адресов.

Следующие дисциплины:

1. Управление процессами
2. Управление памятью (Когда говорят память – имеется ввиду оперативная память)
3. Взаимодействие параллельных процессов (в этом семестре)
4. Управление устройствами (во втором семестре)
5. В этом семестра мы проведём Классификацию ядер ОС.
6. Также мы рассмотрим вопрос (монолитные ядра?)

В лабораторных работах, для того чтобы глубже понять, что такое процесс, как выполнять процесс, мы будем выполнять ЛР на Linux, в общем-то это идеология UNIX. Взгляд классического UNIX на процессы

## Управление процессами

На самом деле речь идёт о ?самих процессах? и об особенностях выполнения процессов операционными системами. Мы рассматриваем основной ресурс – процессорное время, и как оно выделяется процессам.

В любой ОС, основной абстракцией является процесс. Процесс — это основная абстракция ОС. Представление процесса в UNIX самое понятно и самое яркое. С точки зрения UNIX процесс выполняется в режиме ядра и тогда он выполняет реентерабельный код программы, а часть времени в режиме пользователя и тогда он выполняет свой собственный код. Уже в этом определении заложено, собственно, понимания того, как это… Из этого можно сделать вывод, что процесс постоянно переключается из одного режима в другой. При этом в ОС принято выделять диаграмму состояний процессов. (Это можно рассматривать как этапы жизни процесса, он крутится в этих этапах)

Самая обобщённая диаграмма состояний: содержит только те состояния процесса, которые будут в любой ОС. Любые состояния процесса – являются произволом разработчика. Каждый добавляет свои состояния, чтобы лучше построить работу ядра.

Мы запускаем программу на выполнение. Начинает работать ОС. Что же она делает? Вы просто нажали Enter. Мы должны знать больше, чем просто это. Первое состояние называется **порождение**. ОС работает по тем же принципам, что и любая другая программа. … для этого надо определить переменные. Сам процесс тоже должен быть определён, у него должно быть имя. Первый шаг, который выполняет ОС при запуске программы, она присваивает ей идентификатор. Но этого мало, нужно еще управлять. Управление заключается в выделении этому процессу ресурсов для его выполнения. Поэтому должна быть структура, описывающая этот процесс. Это очень большая структура которая имеет большое кол-во полей, большинство которых указывают на другие структуры. Линуксоиды говорят, что при запуске процесса ему выделяется строка в таблице… (Таблица это массив структур, таблица это условное название). На самом деле для того чтобы управлять процессом, в системе существует несколько связных списков. Поэтому кроме идентификации процесса, происходит инициализация структур, которые можно иниц. При порождении процесса. Для того чтобы процесс мог начать выполнять, ему нужно выделить память. После того как процессу выделена память (после того как процесс получил все необходимые ресурсы), он переходит в состояние **готовности**. В многозадачных ОС, к которым относятся *операционные мультизадачные системы пакетной обработки* и *система разделения времен*. То-есть в таких системах будет находится в одно и тоже время, большое кол-во процессов (программ). Поэтому процессы каким-то образом выстраиваются в очередь. В состоянии готовности, находятся все процессы, которые получилы все ресурсы, кроме последнего (процессорного времени). После получения процессорного времени, процесс переходит в состояние **выполнения**. Из состояния выполнения, он может перейти в состояние **блокировки**, если он вынужден ждать ресурс или запросил ввод/вывод (Например запросил ввод файл или ввод данных с клавиатуры/из файла). Получив необходимый ресурс, процесс снова переходит в состояние готовности. Из состояния выполнения, процесс может выйти в состояние **завершения.** Это очень важное состояние для системы т.к. при завершении, у него забираются все ресурсы и возвращаются системе. Когда переходит в состояние выполнения не значит, что он выполняет свой код. В пользовательском режим ?может выполнятся только свой код?

### Три типа событий, переводящих систему в режим ядра:

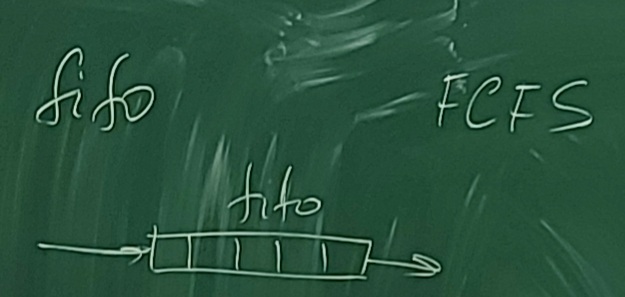
1. **Системные вызовы** (программные прерывания – software interrupts - traps)  
   Системный вызов или системные вызовы – это тот интерфейс, который предоставляет пользователи ОС (часто говорят API – application function interface). То-есть это набор функций, определённых системой, которую может использовать приложение, для того чтобы система обслужила его (обслужила процесс - сервис). Система всегда устроена, чтобы минимизировать кол-во системных вызовов. Ввод/вывод — это всегда обращение к внешним устройствам. … Потому что ни одна ОС не позволяет напрямую обратится к внешнему устройству, т.к. такую систему нельзя было бы защитить можно было бы получить доступ к ядру. Поэтому ввод/вывод – реализованы как системные вызовы. Но это не единственные системные вызовы.   
   Системные вызовы – это синхронные события, которые выполняются ?во время выполнения программы? (синхронно относительно выполняемой программы)
2. **Исключения.**  
   Исключения могут быть **исправимыми** и **неисправимыми**.  
   К неисправимым относятся, ошибки. (Например ошибка деления (на ноль)) Это может быть ошибка адресации…  
   К исправимым исключениям относятся страничные прерывание в результате которого…  
   Исключения являются синхронными событиями, по отношению к исполняемому коду
3. **Аппаратные прерывания.** Они также делятся на типы. Всегда, отдельно, рассматривается прерывание системного таймера – это особе прерывание, периодическое (единственное периодическое), на которое возложены важные функции системы (?например деление кванта? Понятие квант очень удобное для тех рассуждений, которые необходимо выполнить, чтобы понять как работает система разделения времени). Самую большую группу ап. прерываний…  
   … по завершении операции ввода/вывода. Таким образом внешнее устройство, “говорит” системе что ввод/вывод завершён. … об успешном или не успешном завершении. Всё это сделано, для того чтобы программа получила запрошенные данные, или получила сообщения о том, что … произошло успешно. Также принято отдельно выделять … прерывания (например ctrl+alt+del – вызов диспетчера задач).  
   Аппаратные прерывания, являются абсолютно асинхронными событиями… Они не зависит ни от какой-либо работы, выполняемой процессом.

### Планирование и диспетчеризация:

Планирование необходимо рассматривать в самом общем виде. Выделение дефицитного ресурса, всегда выполняется по некоторой дисциплине планирования. В состоянии готовности, в мультизадачных… может находится большое кол-во процессов. Необходимо организовать такую очередь.  
Планирование – это постановка процессов в очередь по выбранной ?дисциплине планировки?  
… это модуль ядра выполняющий указанную работу.  
Диспетчеризация – это непосредственное выделение, какого-либо ресурсов. (мы говорим про процессорное время). Диспетчеризация относится к выделению процессу процессорного времени. Ну если процессы в очереди, то соответственно процессорное время, получит процесс из головы такой очереди. (всё как у людей, мы щас в этом убедимся)

Существуют дисциплины планирования, и они связаны с … операционных систем. (к однозадачным ОС это не относятся – там нет очереди) Мы рассматривали *мультизадачные системы пакетной обработки* и *системы разделения времени*.  
Очевидна разница между ними, у них разные задачи. В первом случае, пользователь отделён от выполнения программы. (можно сказать что между ними находится пакет)  
Во втором случае наоборот – интерактивное взаимодействие (может влиять на программу и ?писать что-то туда?).

Мы можем классифицировать все алгоритмы планирования следующим образом: планирование бывает **без переключения** и **с переключением,** (правильно:) **с приоритетами** и **без приоритетов**, и **без вытеснения** и **с вытеснением** (?без вытеснения не бывает?). Сейчас поймём, что это значит:

Сначала рассмотрим алгоритмы планирования в системах пакетной обработки. (Всегда лучше идти от простого к сложному)   


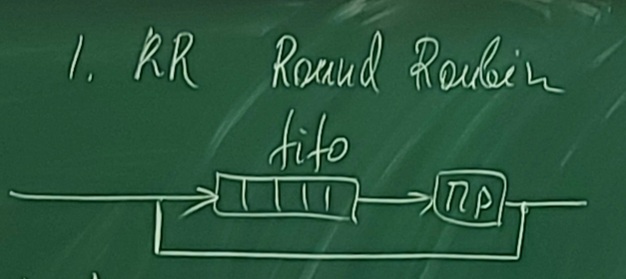
**fifo – первый алгоритм.** (часто в современной интерпретации FCFS)  
Без вытеснения, без …  
Выполняется от начала до конца

**SJF – второй алгоритм** (Shortest Job First)  
Очередь выстраивается по заявленному времени выполнения. Используя эту априорную (до опыта) информацию система выстраивала процессы в очередь, при это предпочтение отдавалось коротким процессам. (показатель производительности – кол-во программ, выполненных в единицу времени (для mainframe)).  
Но это не значит, что большую программу было невозможно выполнить.  
Негативное явление – а именно бесконечное откладывание. То-есть всё время приходят короткие задания и длинные задания не могут получить процессорное время.  
Без переключения, без вытеснения, с приоритетами.

**Третий алгоритм – SRT** (Shortest Remaining Time) – наименьшее оставшееся время.  
Согласно этой дисциплине планирования, выполняющийся процесс может быть прерван если поступит в очередь процесс, с меньшим оценочным временем выполнения. В этой дисцплине, при поступлении нового процесса сравниволось заявленное время выполнения процесса и время, которое осталось процессу до завершения (из заявленного вычесть полученное время = оценочное время – сколько еще понадобиться), и если это время больше то он вытесняется.  
С приоритетами и вытеснением.  
Этот алгоритм еще хуже в плане откладывания.

**Четвёртый алгоритм – HRN** (Highest Response Ration) – наибольшое .. время  
В этой системе, приоритеты пересчитываются по формуле: , где ts – оценочное время выполнения а tw – время ожидания => процесс не будет откладываться бесконечно, т.к. чем больше время ожидания (простоя) в очереди готовых процессов, тем больше приоритет.

### Планирование в системах разделения времени:

…Поэтому в этой системе процессорное время стали квантовать.  


1. **Первый алгоритм - RR** (Round Roubin)  
Процессы выстраиваются в очередь, но выполняются не до выполнения, а определённый квант времени. После этого процесс ставится в конец очереди.  
Без приоритетов, без вытеснения, с переключением.  
… считается статический квант времени.  
Несмотря на простоту этого алгоритма, существует огромное кол-во вариантов RR.  
Вот несколько из них:

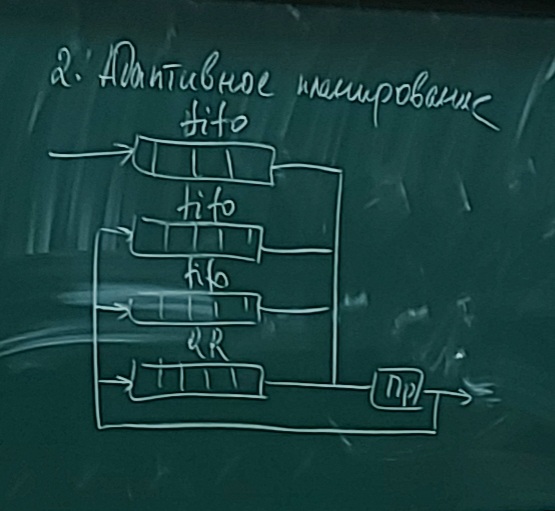
В современных системах совершаются попытка усовершенствования RR путём назначения кванта времени динамически (Не определена, зависит от ситуации в системе). Но это приведёт к дополнительным накладным …

Можно перечислить:

1. Min-Max RR (MMRR)
2. Average Max RR (AMRR)
3. Ettieient Dynamic RR (EDRR)

Мы говорим квантование (quantum) – они Time slicing (отрезание времени)

2. Другим классическим алгоритмом является **алгоритм приоритетного планирования.** (иногда можно сказать – многоуровневые очереди, но не всегда)  
Также этот алгоритм называется **адаптивное планирование**.

Много очередей, с разными приоритетами. RR в последнюю очередь  


### Лекция 4 (начало)

…Для этой очереди выбирается квант, таким образом, чтобы … или успело завершиться. Если процесс за выделенный ему квант времени не сделал ни то, ни другое, он переходит в более низкоприоритетную очередь. Переходят так, пока не попадут в самую низкоприоритетную очередь, которая выполняется по алгоритму RR. В итоге, в этой очереди оказываются процессы, требующие много проц. Времени (вычислительные процессы). Также этой очереди, крутится холостой процесс, т.к. система не может ничего не делать.