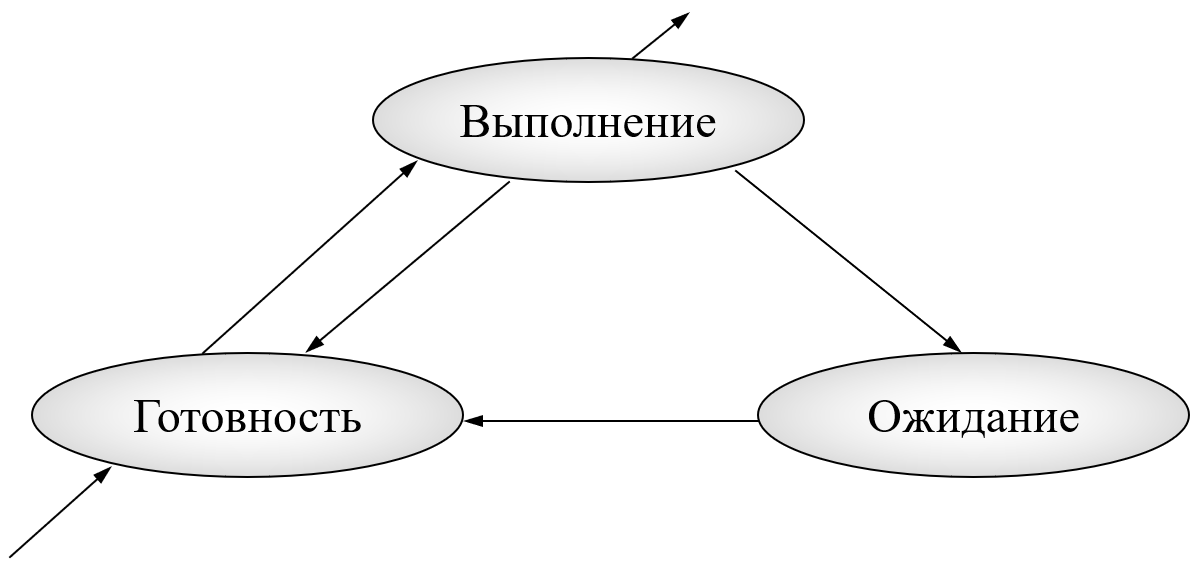
***Вторая лаба***

# Многозадачность.

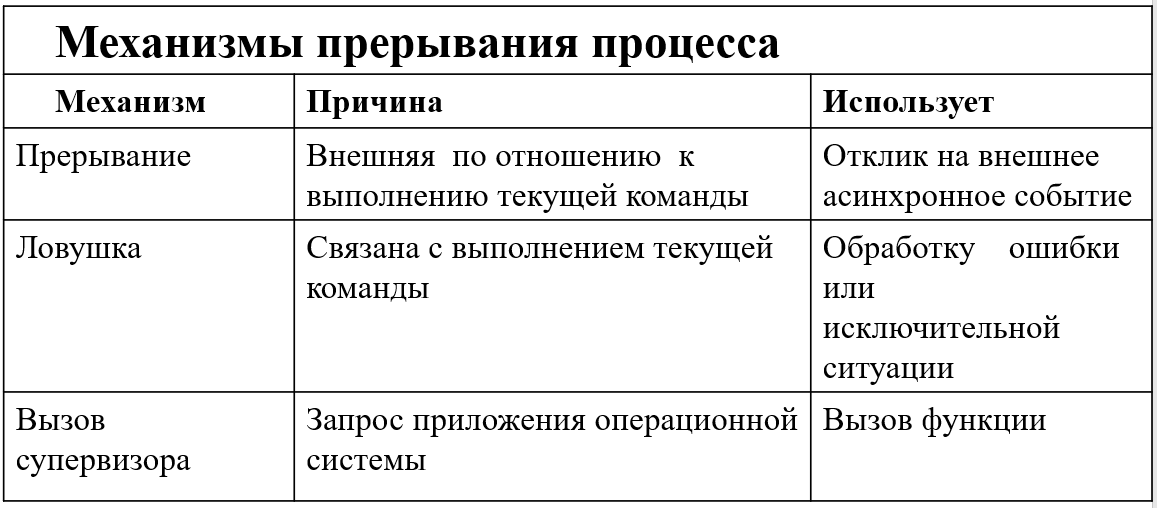
В многозадачной (многопроцессорной) системе процесс может находиться в одном из трех основных состояний:  
 1. Выполнение – активное состояние процесса, во время которого процесс обладаем всеми необходимыми ресурсами и непосредственно выполняется процессором  
 2. Ожидание – пассивное состояние процесса. Процесс заблокирован, он не может выполняться по своим внутренним причинам  
 3. Готовность – также пассивное состояние процесса, но в этом случае процесс заблокирован в связи с внешним по отношению к нему обстоятельствами: процесс имеет все требуемые для него ресурсы, он готов выполняться, однако процессор занят выполнением другого процесса.

В ходе жизненного состояния цикла каждый процесс переходит из одного состояния в другое в соответствии с алгоритмом планирования процессов, реализуемым в данной ОС:

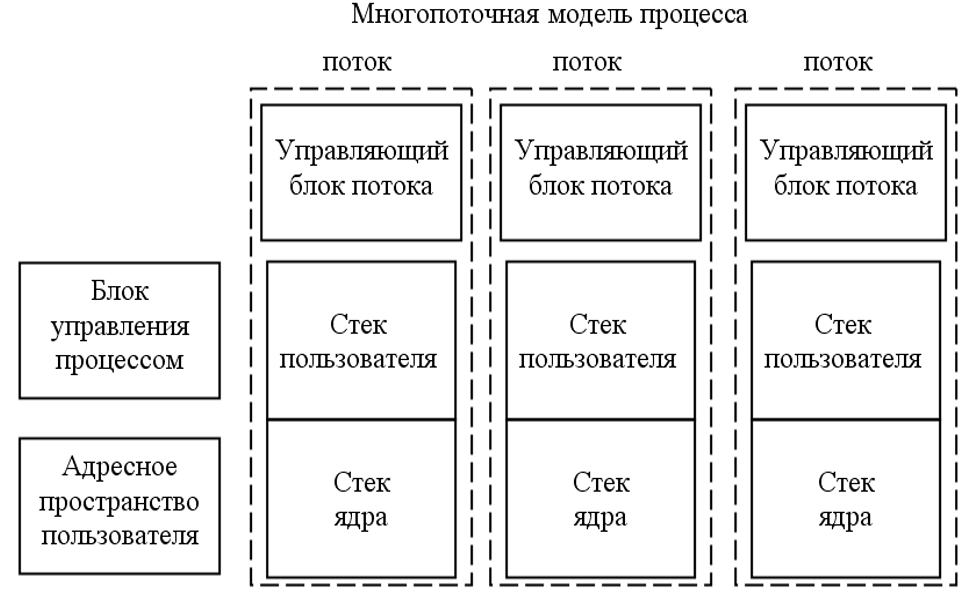


В состоянии выполнение в однопроцессорной системе может находиться только один процесс, а в каждом из состояний ожидание и готовность – несколько процессов, эти процессы образуют очереди соответственно ожидающих и готовых процессов.

# 2. Переходы между процессами.

Переключение процесса может произойти в любой момент, когда управление от выполняющегося процесса переходит к ОС.   
  
Имеются системные прерывания двух видов:  
- Обычные прерывания  
- Ловушки  
Прерывания первого вида происходят из-за событий определенного типа, не связанных с выполняющимся процессом и являющихся внешними по отношению к нему. Ловушки связаны с ошибкой или исключительной ситуацией, возникшей в результате выполнения текущего процесса.

# 3. Многопоточность.

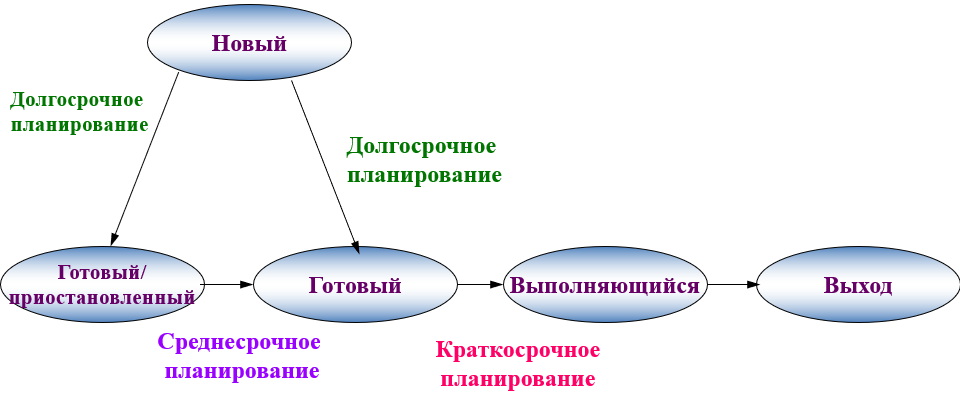
Многопоточностью называется способность ОС поддерживать в рамках одного процесса выполнение нескольких потоков.   
В многопоточной среде процесс определяется как структурная единица распределения ресурсов, а также структурная единица защиты.  
С каждым процессом связаны управляющий блок, адресное пространство. Для каждого потока создаются свои отдельные стеки, а также свой управляющий блок, в котором содержатся значения регистров, приоритет и другая информация о состоянии потока:  


Основные преимущества использования потоков с точки зрения производительности:  
1. Создание нового потока в уже существующем процессе занимает меньше времени, чем создание совершенно нового процесса  
2. Поток можно завершить быстрее, чем процесс  
3. Переключение потоков в рамках одного и того же процесса происходит быстрее  
4. При использовании потоков повышается эффективность обмена информацией между двумя выполняющимися программами (потоками)

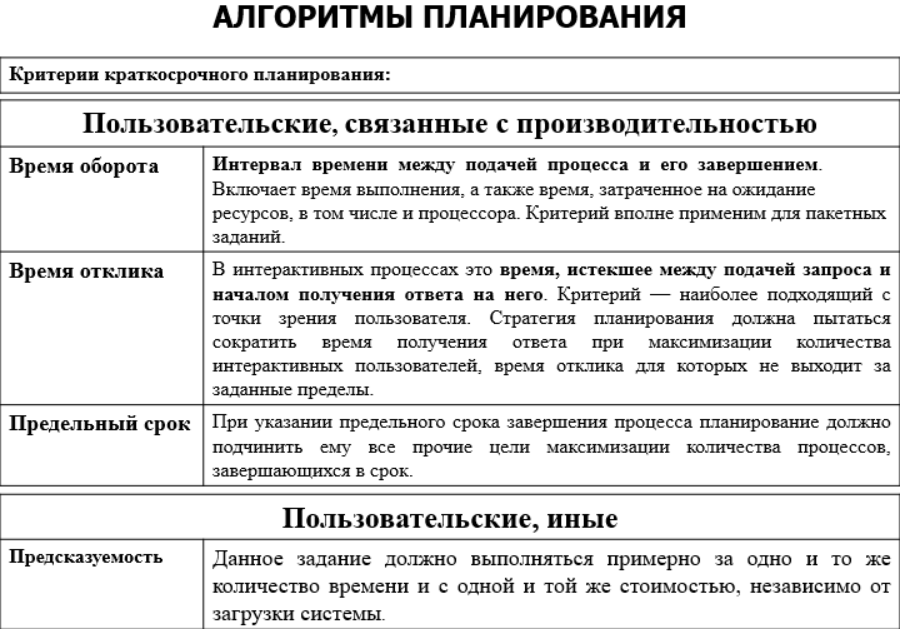
Планирование и диспетчеризация осуществляется на основе потоков. Таким образом, большая часть информации о состоянии процесса, имеющей отношение к его выполнению, поддерживается в структурах данных на уровне потоков.  
Однако есть несколько действий, которые затрагивают все потоки процесса и которые ОС должна поддерживать именно на этом уровне:  
Если процесс приостанавливается, то при этом предполагается, что его адресное пространство будет выгружено из основной памяти.  
Поскольку все потоки процесса используют одно и тоже адресное пространство, все они должны одновременно перейти в состояние приостановленных. Соответственно прекращение процесса приводит к прекращению всех составляющих его потоков.

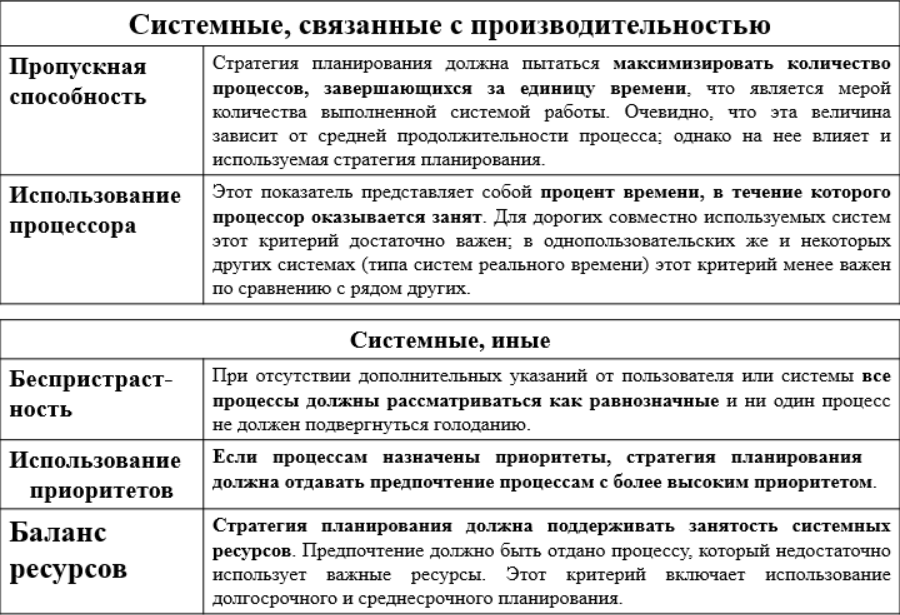
4.Типы планирования процессора.

Долгосрочное планирование – решение о добавлении процесса в пул процессов в состоянии готовности.  
 Среднесрочное планирование – решение о добавлении процесса к числу процессов частично или полностью размещенных в основной памяти.  
 Краткосрочное планирование – решение о том, какой из готовых процессов будет выполняться процессором.  
Планирование ввода-вывода – решение о том, какой из запросов процессов на операции ввода-вывода будет обработан свободным устройством ввода-вывода.



# 5. Критерии краткосрочного планирования. Функция выбора и режим решения.





Функция выбора определяет, какой из готовых к выполнению процессов будет выбран следующим для выполнения.  
Режим решения определяет, в какие моменты времени выполняется функция выбора. Режимы решения подразделяются на две категории:

* Не вытесняющие - процесс выполняется до тех пор, пока он не завершиться или пока не окажется в заблокированном состоянии ожидая завершения операции ввода-вывода или запроса некоторого системного сервиса.
* Вытесняющие - выполняющийся процесс может быть прерван. Решение может приниматься при запуске нового процесса по прерыванию, которое переводит заблокированный процесс в состояние готовности к выполнению, или периодически – на основе прерываний таймера.

6. Алгоритмы планирования FCFS и RR.

(FCFS) Первым поступил – первым обслужен.

Как только процесс становится готовым к выполнению, он присоединяется к очереди готовых процессов. При прекращении выполнения текущего процесса для выполнения выбирается процесс, который находился в очереди дольше других.

(RR) Круговое планирование.

Таймер генерирует прерывания через определенные интервалы времени. При каждом прерывании исполняющийся в настоящий момент процесс помещается в очередь готовых к выполнению процессов, и начинает выполняться очередной процесс, выбираемый в соответствии со стратегией FCFS. Эта методика так же известна как квантование времени.

# 7. Алгоритмы планирования SPN и SRT.

(SPN) Наиболее короткий процесс следующий.

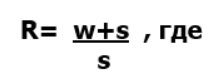
Если в очереди появляется короткий процесс, то он выполняется первым.  
Автоматически система не может определить какой процесс короче. Такой метод используется в пакетной обработке, т.к. можно указать ориентировочное время выполнения.

(SRT) Наименьшее оставшееся время.

Вытесняющая версия SPN. В этом случае планировщик выбирает процесс с наименьшим ожидаемым временем до окончания процесса. При присоединении нового процесса к очереди может оказаться что его оставшееся время меньше, чем оставшееся время выполняемого в настоящее время процесса и произойдет вытеснение.

8. Управление процессами. Алгоритмы планирования HRRN и FB.

(HRRN) Наивысшее отношение отклика.



R – отношение отклика, w – время, затраченное процессом на ожидание;  
s – ожидаемое время обслуживания.

То есть при завершении или блокировании текущего процесса для выполнения из очереди готовых процессов выбирается тот, который имеет наибольшее значение R.

(FB) Снижение приоритета.  
Достигается путём выполнения вытесняющего по квантам времени планирования с использованием динамического механизма. При выходе процесса в систему он помещается в очередь RQ0. После первого выполнения и возвращения в состояние готовности, процесс помещается в очередь RQ1 и в дальнейшем при каждом вытеснении этого процесса он вносится в очередь со всё меньшим приоритетом. По достижении очереди с наиболее низким приоритетом, процесс уже не покидает её, всякий раз после вытеснения попадая в неё вновь. Этот подход известен как многоуровневый возврат (multilevel feedback2), поскольку при блокировании или вытеснении процесса осуществляется его возврат на очередной уровень приоритетности.

***Третья лаба***

1. Синхронизация потоков

Потребность в синхронизации потоков возникает только в мультипрограммной операционной системе и связана с совместным использованием аппаратных и информационных ресурсов вычислительной системы. Синхронизация необходи­ма для исключения гонок и тупиков при обмене данными между потоками, раз­делении данных, при доступе к процессору и устройствам ввода-вывода.

2.Эффект гонок

При необходимости использовать один и тот же ресурс параллельные процессы (потоки) конкурируют друг с другом за получение доступа к ресурсу. В этом случае может возникать эффект гонок – ситуация в которой два или более процессов обрабатывают разделяемые данные, и конечный результат обработки зависит от соотношения скоростей процессов.

Чтобы исключить эффект гонок в каждый момент в крит секции должен находиться максимум один поток (процесс) – это взаимное исключение. Также можно использовать мьютекс или семафор.

3. Мьютексы

**Мью́текс** — в программировании необходим для сопоставления синхронно выполняющихся потоков.[[1]](https://ru.bmstu.wiki/%D0%9C%D1%8C%D1%8E%D1%82%D0%B5%D0%BA%D1%81#cite_note-1). Мью́текс представляет собой концепцию программирования, которая используется для решения вопросов многопоточности. Мьютекс допускает только один поток в контролируемом участке, заставляя другие потоки, которые пытаются получить доступ к этому разделу ждать, пока первый поток не вышел из этого раздела.

Принимает два значенения:

* открыт - поток может войти в свою критическую секцию;
* закрыт - поток не может войти в критическую секцию.

**Задача** мьютекса — защита объекта от доступа к нему других потоков, отличных от того, который завладел мьютексом. В каждый конкретный момент только один поток может владеть объектом, защищённым мьютексом. Если другому потоку будет нужен доступ к переменной, защищённой мьютексом, то этот поток засыпает до тех пор, пока мьютекс не будет освобождён.

4. Критическая секция

**Критическая секция** — участок исполняемого кода программы, в котором производится доступ к общему ресурсу (данным или устройству), который не должен быть одновременно использован более чем одним [потоком выполнения](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D0%BE%D0%BA_%D0%B2%D1%8B%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F). При нахождении в критической секции двух (или более) потоков возникает состояние «гонки» («состязания»). Во избежание данной ситуации необходимо выполнение четырех условий:

1. Два потока не должны одновременно находиться в критических областях.
2. В программе не должно быть предположений о скорости или количестве процессоров.
3. Поток, находящийся вне критической области, не может блокировать другие потоки.
4. Невозможна ситуация, в которой поток вечно ждет попадания в критическую область.

В операционных системах семейства [Microsoft Windows](https://ru.wikipedia.org/wiki/Microsoft_Windows) разница между мьютексом и критической секцией в том, что мьютекс является объектом ядра и может быть использован несколькими процессами одновременно, критическая секция же принадлежит процессу и служит для синхронизации только его потоков.

***Четвертая лаба***

1. Семафоры

Семафор – защищенная переменная значение которой можно опрашивать и менять только при помощи спец операций.

Семафоры которые принимают значение 0 и 1 – бинарные, остальные обобщенные.

P(S) = WaitForSingleObject -1;

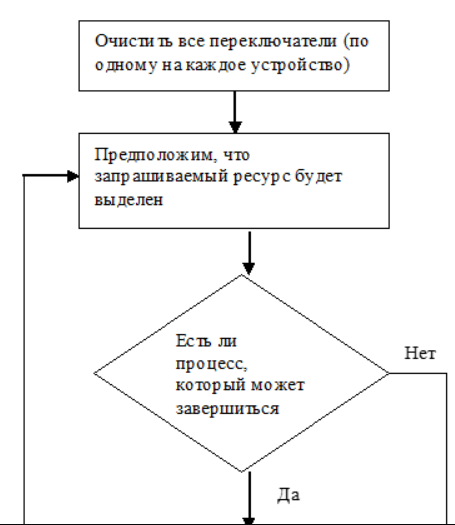
V(S) = Release +1;

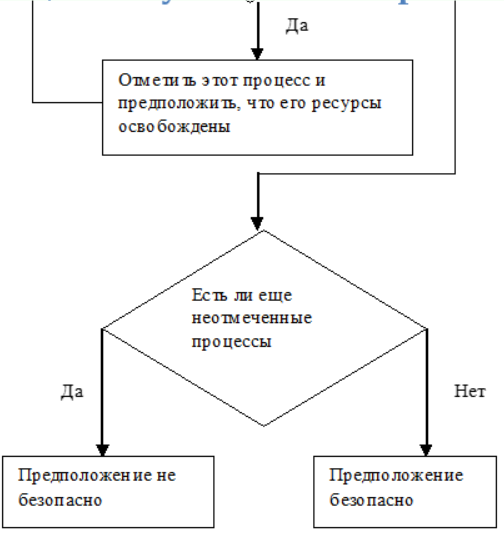
2. Алгоритмы избежания тупиков

Есть два процесса: A и B. Им нужны ресурсы R1 и R2 соответственно и каждый из них занимает свой ресурс при старте работы. Но в какой-то момент времени процессу A понадобился ресурс R2, а процессу B ресурс R1 и, так как каждый из этих ресурсов уже используется, а освобождение этих самых ресурсов не происходит, то оба процесса становятся в режим ожидания, тем самым и создавая тупик.

*Алгоритм банкира.*

Для каждого из ресурсов создаётся специальный переключатель (Выделен/Не выделен). Предполагаем, что ресурс будет выделен. Проверяем, есть ли процессы, обладающие данным ресурсом, которые могут завершится. Если такой процесс есть, отмечаем его как завершённый и предполагаем, что его ресурсы освобождены. Далее проверяем есть ли ещё процессы, владеющие данным ресурсом. Если нет, то предположение безопасно. Если да - не безопасно.





Недостатки:

1. Алгоритм исходит из фиксированного количества ресурсов. Количество единиц ресурсов определяется при генерации.
2. Требуются постоянные числа одновременно работающих процессов.
3. При существующей системе распределения ресурсов возможны длительные периоды нахождения процессов в состоянии приостановки.
4. Для пользователя проблематично указать те ресурсы, что ему потребны.

3. Алгоритм Медника. Выход из тупика.

Метод Медника (Обнаружения), не препятствуя возникновению тупика, а предполагает его обнаружение и восстановление прерванной им работы.

Для функционирования алгоритма необходимо использование таблиц, в которых собиралась бы информация:

* Назначение ресурсов процессами   
  (Таблица распределения ресурсов)
* Процессах, блокированных при попытке обращения к ресурсу   
  (Таблица блокированных процессов)

Существуют следующие подходы для восстановления тупиковых состояний:

1. Прекращение процессов. Прекращаем процессы до тех пор, пока не станет доступным достаточное количество ресурсов для устранения тупика. Прекращаем изначально те процессы, которые имеют минимальную цену прекращения (возобновления): имеющие минимальный приоритет, только что созданные или те процессы, которые используют наименьшее количество ресурсов. В худшем случае уничтожаем все процессы, находящиеся в тупике, кроме одного.

Перехват ресурсов. Если есть другие процессы и есть возможность перехватить ресурсы, то мы отнимаем у процессов достаточное количество ресурсов и отдаём процессам, находящимся в тупике, чтобы ликвидировать сам тупик. Также если есть некий резерв, то мы можем брать ресурсы из него.