Операционные системы

Тема 3. Управление процессами

Процесс **(или по-другому, задача) - абстракция, описывающая выполняющуюся программу или часть программы. Для операционной системы процесс представляет собой единицу работы, заявку на потребление системных**

Процесс

- **Процесс** –
  - **Это программа в состоянии выполнения**
  - **Некоторый объект, который выполняется на процессоре**

  По сути своей все ПО, которые работает под управлением ОС на нашем ПК, даже включая иногда и саму ОС организованно в виде множества процессов. Процессы это минимальный примитив, который позволяет организовать некоторую многозадачность.
- Как исполняемый объект, процесс позволяет параллельное выполнение нескольких программ в системе (ЦП переключается между программами)
- Все ПО, работающее на компьютере, включая саму ОС, организованно в виде множества процессов.

Мультизадачность

- Способ организации вычислительного процесса, при котором на одном процессоре попеременно выполняется сразу несколько программ.
- Общее время выполнения сокращается, хотя время выполнения самих процессов обычно увеличивается.

Управление процессами

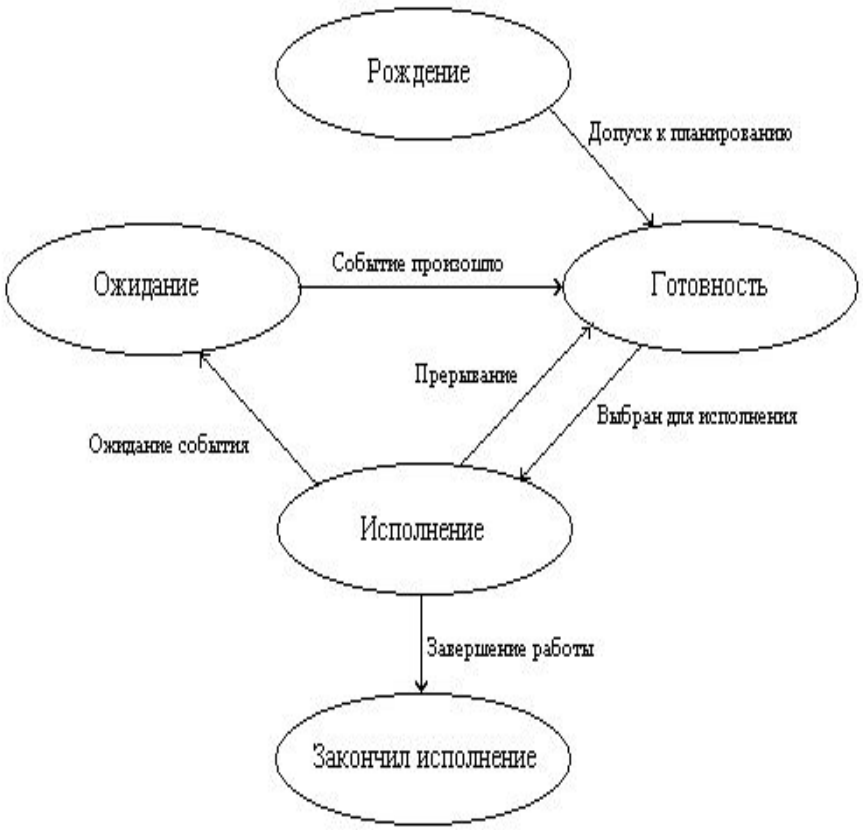
При работе системы запускается множество процессов, о которых пользователь зачастую не подозревает.

Например:

Управление процессами

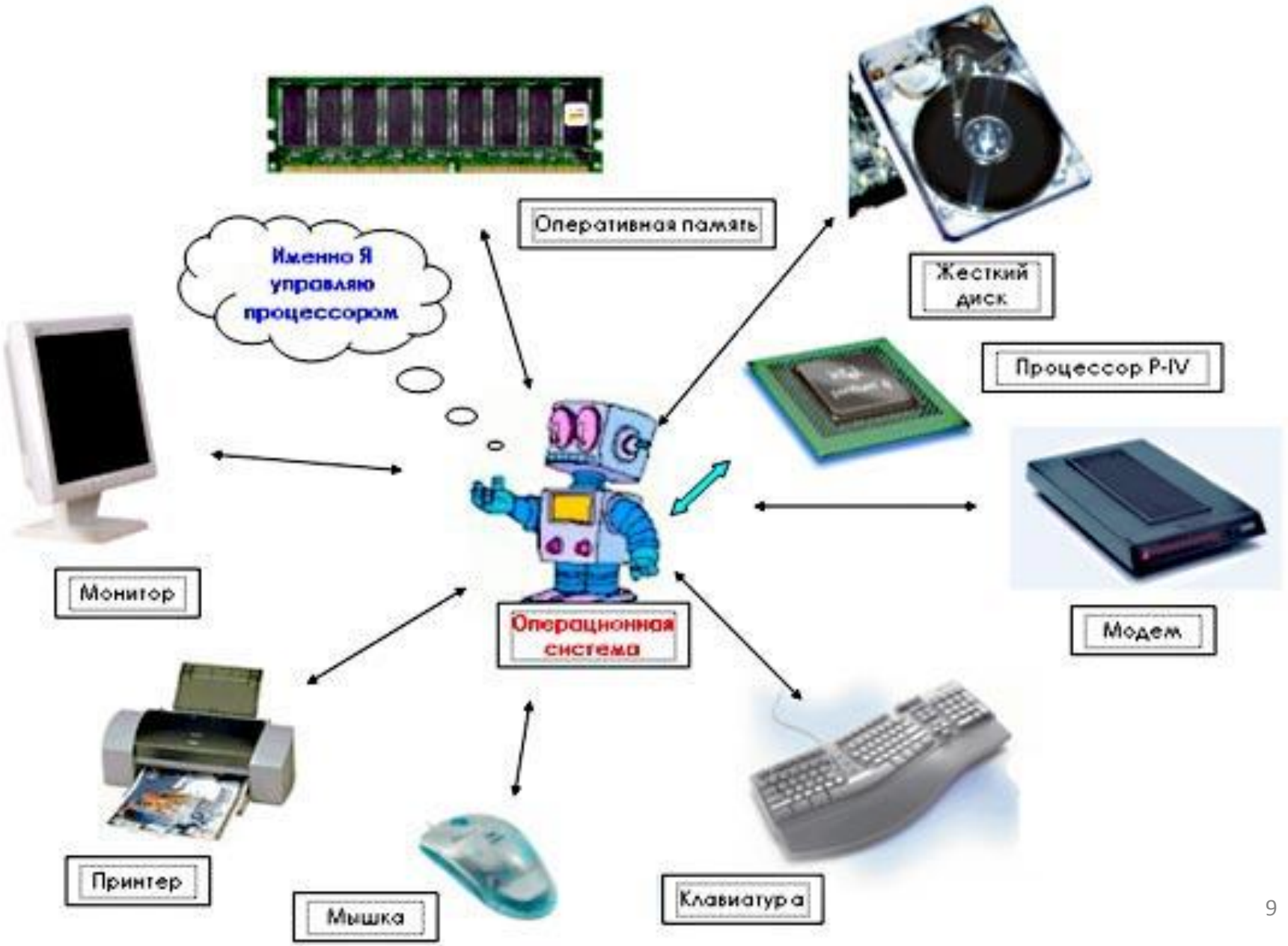
- Запущен процесс, ожидающий входящей электронной почты;
- Другой процесс — антивирусная программа выполняет периодическую проверку доступности определений новых вирусов;
- Запущены процесс, инициированный пользователем, сброс пользовательских фотографий на USB-накопитель;
- Одновременно пользователь через браузер просматривает Интернет.

Управление процессами



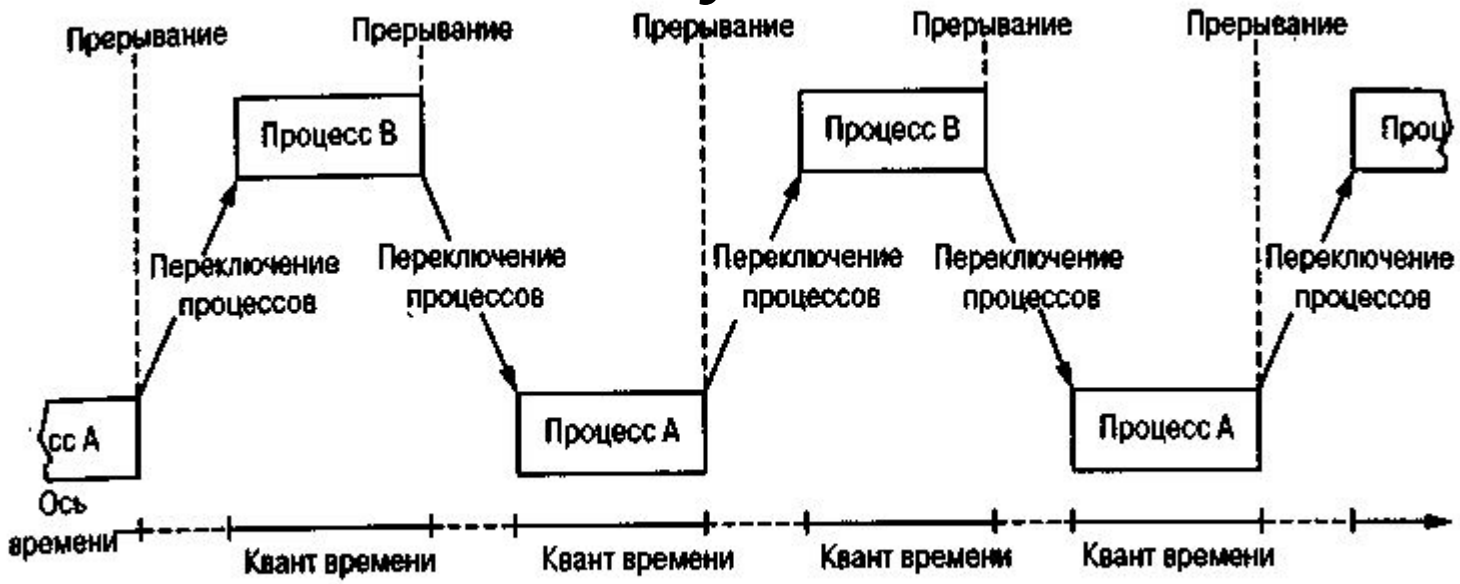
Всей этой работой нужно управлять, и для этого требуется многозадачная система, поддерживающая работу нескольких процессов.

Управление процессами



Управление процессами

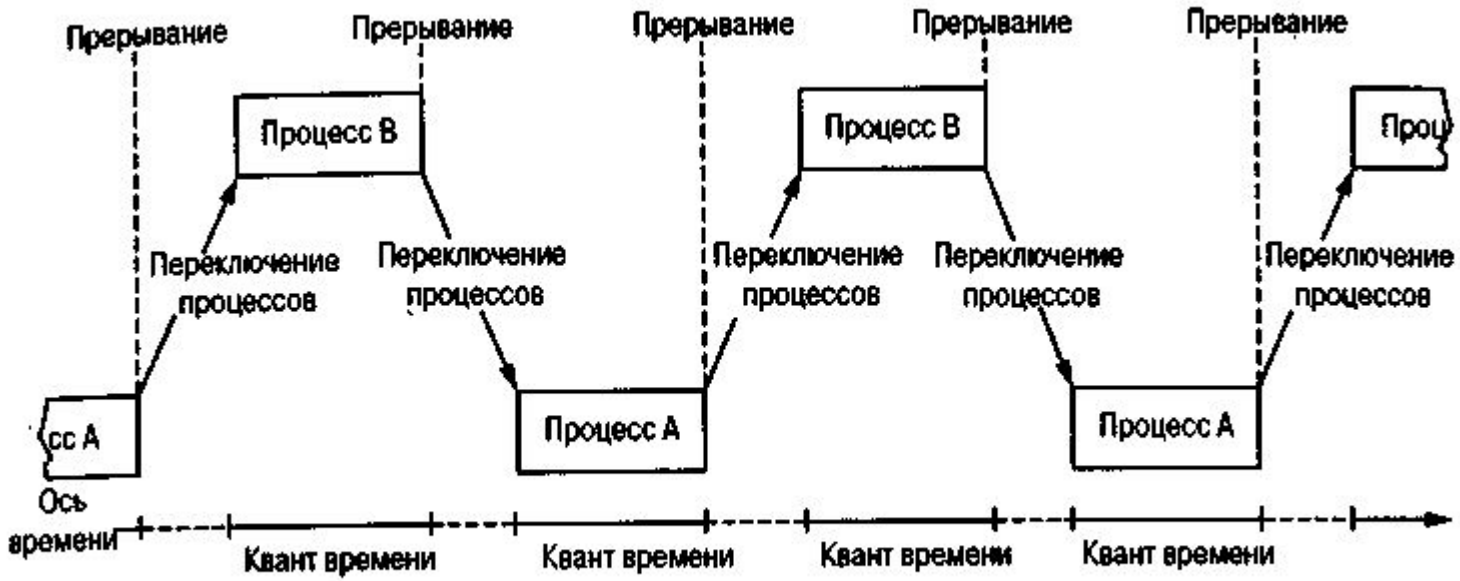
В любой многозадачной системе центральный процессор быстро переключается между процессами, предоставляя каждому из них десятки или сотни *миллисекунд*.



Разделение времени между процессами А и В

Управление процессами

При этом хотя в каждый конкретный момент времени центральный процессор работает только с одним процессом, в течение *1 секунды* он может успеть поработать с несколькими из них, создавая иллюзию параллельной работы

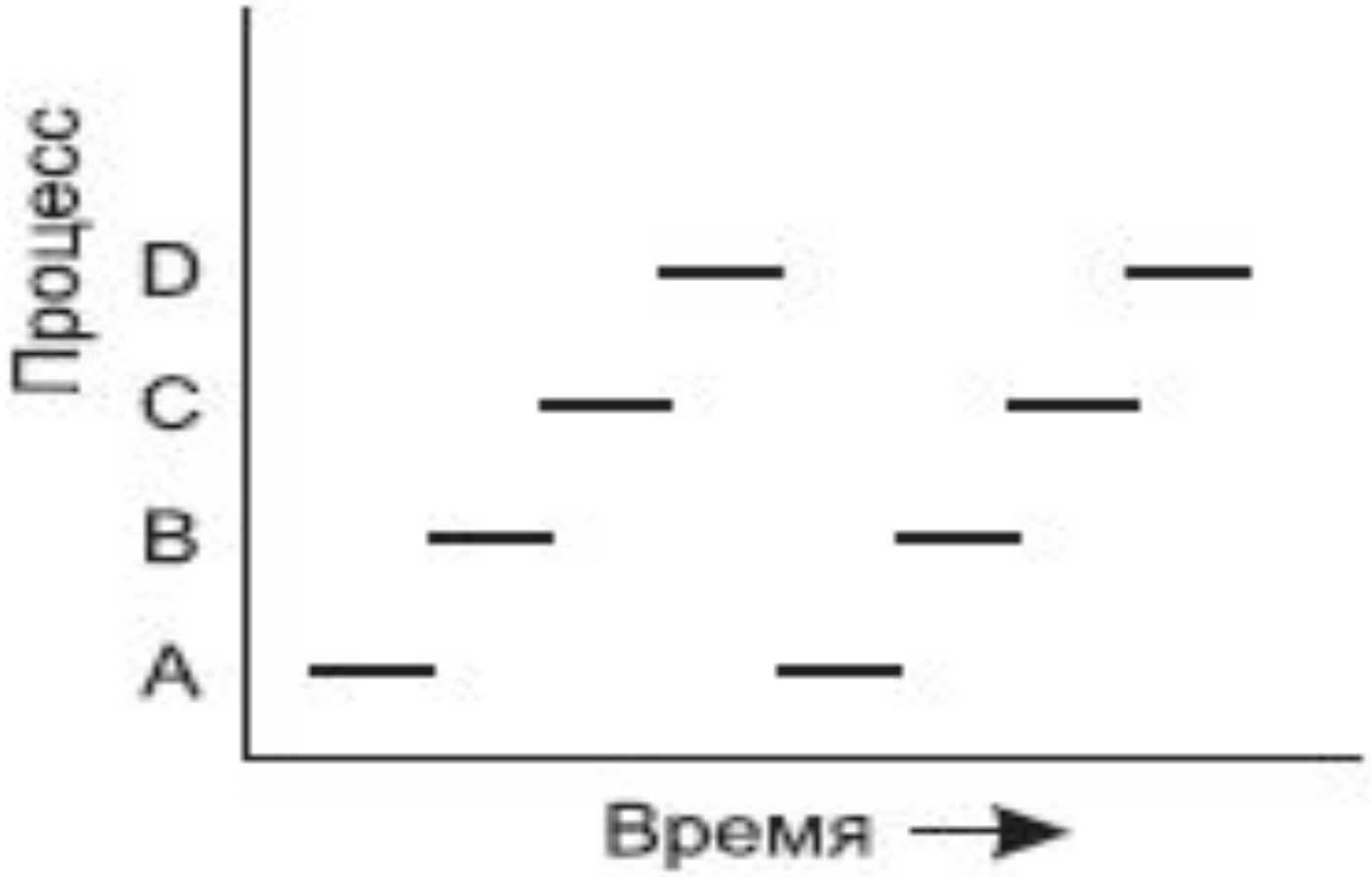


Разделение времени между процессами А и В

Пример по аналогии.

Представим себе программиста, решившего заняться кулинарией и испечь пирог на день рождения дочери. У него есть рецепт пирога, а на кухне есть все ингредиенты: мука, яйца, сахар, и т. д. В данной аналогии рецепт — это программа (то есть алгоритм, выраженный в некой удобной форме записи), программист — это центральный процессор, а ингредиенты пирога — это входные данные. Процесс — это действия, состоящие из чтения рецепта нашим кулинаром, выбора ингредиентов и выпечки пирога. Теперь представим, что на кухню вбегает сын программиста и кричит, что его ужалила пчела. Программист записывает, на каком месте рецепта он остановился (сохраняется состояние текущего процесса), достает книгу советов по оказанию первой помощи и приступает к выполнению изложенных в ней инструкций. Перед нами процессор, переключенный с одного процесса (выпечки) на другой процесс, имеющий более высокую степень приоритета (оказание медицинской помощи), и у каждого из процессов есть своя программа (рецепт против справочника по оказанию первой помощи). После извлечения пчелиного жала программист возвращается к пирогу, продолжая выполнять действия с того места, на котором остановился. Ключевая идея здесь в том, что процесс — это своего рода действия. У него есть программа, входные и выходные данные и состояние.

Управление процессами



Процесс – это

Выполняемый код в ОП

Область данных в ОП

Ресурсы

- Файлы
- Другие программы
- Прочие ресурсы

Служебная информация

ОС ведет некоторый список процессов

- **Таблица процессов** – на каждый процесс приходится одна запись в этой таблице;
- Для каждого процесса нужна некая структура данных, которая содержала бы в себе все то, что относится к процессу (некий контекст). Для этого в ОС в используется **Блок управления процессом (Process Control Block – PCB)** – который описывает свой процесс, которому он принадлежит и его текущее состояние

Блок управления процессом (Process Control Block – PCB).

Данный блок постоянного размера для любого процесса в ОС.

Содержит всю информацию, необходимую для выполнения над процессом любых действий (приостановки, последующего восстановления процесса, выгрузки на диск и загрузки с диска):

- Идентификаторы процесса или Номер процесса (так называемый PID — Process IDentificator)
- Информация о пользователе
- Состояние процесса
- Приоритет которым обладает данные процесс (информация для планировщика)
- Привилегии – доступ к памяти, допустимые инструкции
- Информация о виртуальной памяти, присвоенной процессу
- Статистическая информация и ограничения (ограничения по времени выполнения, статистика о затраченном времени ЦП)
- Вв/выв – владение ресурсами, открытые файлы, выделенные устройства.

Управление процессами

Диспетчер – отправляет процессы на выполнение, выделяет время ЦП и переключает ЦП с одного процесса на другой.

В любой момент времени любой процесс может находиться в каком-либо состоянии:

- Выполнение
- Готовность к выполнению
- Ожидание освобождения ресурса
- Ожидание завершения операции вв/выв.

- На протяжении существования процесса его выполнение может быть многократно прервано и продолжено.

Фоновые процессы

Во время работы операционной системы создаются, как правило, несколько процессов. Некоторые из них представляют собой высокоприоритетные процессы, то есть процессы, взаимодействующие с пользователями и выполняющие для них определенную работу. Остальные являются ***фоновыми процессами***, не связанными с конкретными пользователями, но выполняющими ряд специфических функций.

Фоновые процессы

Например, фоновый процесс, который может быть создан для приема входящих сообщений *электронной почты*, основную часть времени проводит в спящем режиме, активизируясь только по мере появления писем.

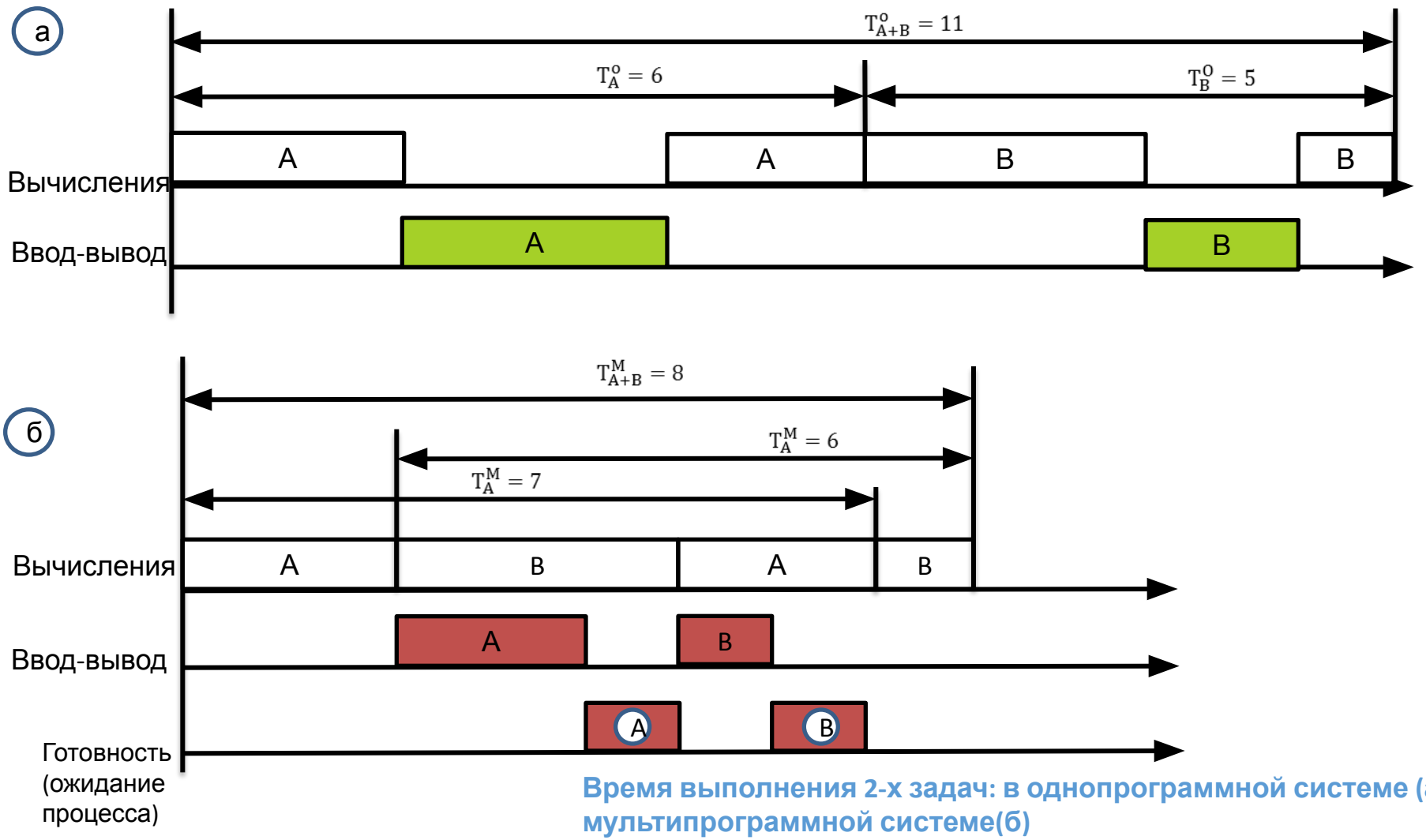
Другой фоновый процесс, который может быть создан для *приема входящих запросов* на веб-страницы, размещенные на машине, просыпается при поступлении запроса с целью его обслуживания.

Фоновые процессы

Фоновые процессы, предназначенные для обработки какой-либо активной деятельности, связанной, например, с электронной почтой, веб-страницами, новостями, выводом информации на печать и т. д., называются **демонами**. Обычно у больших систем насчитываются десятки демонов.

Создание процессов

- 1. Инициализация системы.
- 2. Выполнение работающим процессом системного вызова, предназначенного для создания процесса.
- 3. Запрос пользователя на создание нового процесса..



Время выполнения 2-х задач: в однопрограммной системе (а), в мультипрограммной системе(б)

Состояния процесса

1. Процесс заблокирован в ожидании ввода
2. Диспетчер выбрал другой процесс
3. Диспетчер выбрал данный процесс
4. Входные данные стали доступны

Состояния процесса

- ***выполнение*** - активное состояние процесса;
- ***ожидание*** - пассивное состояние процесса, процесс заблокирован;
- ***готовность*** - также пассивное состояние процесса.

Состояния процесса

Выполнение - активное состояние процесса, во время которого процесс обладает всеми необходимыми ресурсами и выполняется процессором.

Состояния процесса

Ожидание - процесс заблокирован, он не может выполняться по своим внутренним причинам, он ждет осуществления некоторого события, например: завершения операции ввода-вывода, получения сообщения от другого процесса, освобождения какого-либо необходимого ему ресурса.

Состояния процесса

Готовность - процесс заблокирован в связи с внешними по отношению к нему обстоятельствами: процесс имеет все требуемые для него ресурсы, он готов выполняться, однако процессор занят выполнением другого процесса.

Состояния процесса

В состоянии «*выполнение*» в однопроцессорной системе может находиться только один процесс. А в состояний «*ожидание*» и «*готовность*» - может одновременно находиться несколько процессов и эти процессы образуют очереди соответственно ожидающих и готовых процессов.

Спулер

В операционных системах совместно работающие процессы могут использовать какое-нибудь общее хранилище данных, доступное каждому из них по чтению и по записи. Это общее хранилище может размещаться в оперативной памяти. Чтобы посмотреть, как взаимодействие процессов осуществляется на практике, давайте рассмотрим простой общеизвестный пример — спулер печати. Когда процессу необходимо распечатать какой-нибудь файл, он помещает имя этого файла в специальный **каталог спулера**. Другой процесс под названием **демон принтера** периодически проверяет наличие файлов для печати и в том случае, если такие файлы имеются, распечатывает их и удаляет их имена из каталога.

Планирование процессов

1. определение момента времени для смены выполняемого процесса;
2. выбор процесса на выполнение из очереди готовых процессов;
3. переключение контекстов "старого" и "нового" процессов (аппаратно).

Планирование процессов

Исходя из **трех** основных состояний процесса **«готов», «выполнение» «заблокирован»** планировщик должен знать, какой процесс находится в каком состоянии. Все усложняется, если ЦП содержит несколько вычислительных ядер.

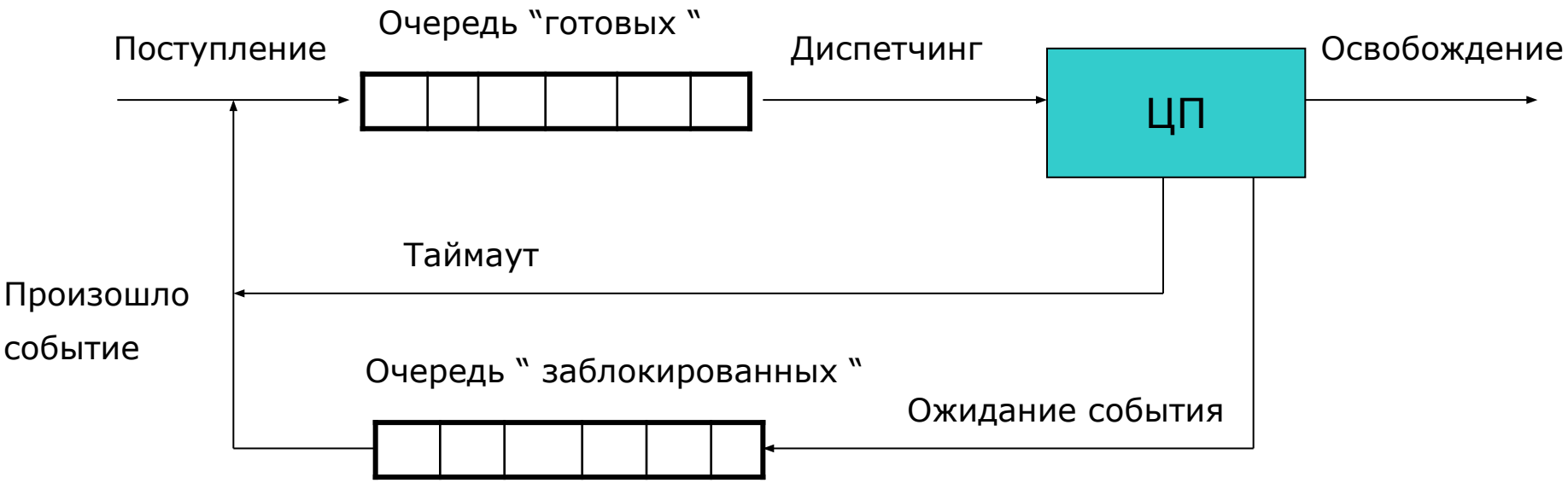
Поэтому в ОС вводятся различные **очереди (списки)** для планирования процессов.

Исходя из трех состояний процесса вводятся 3 очереди:

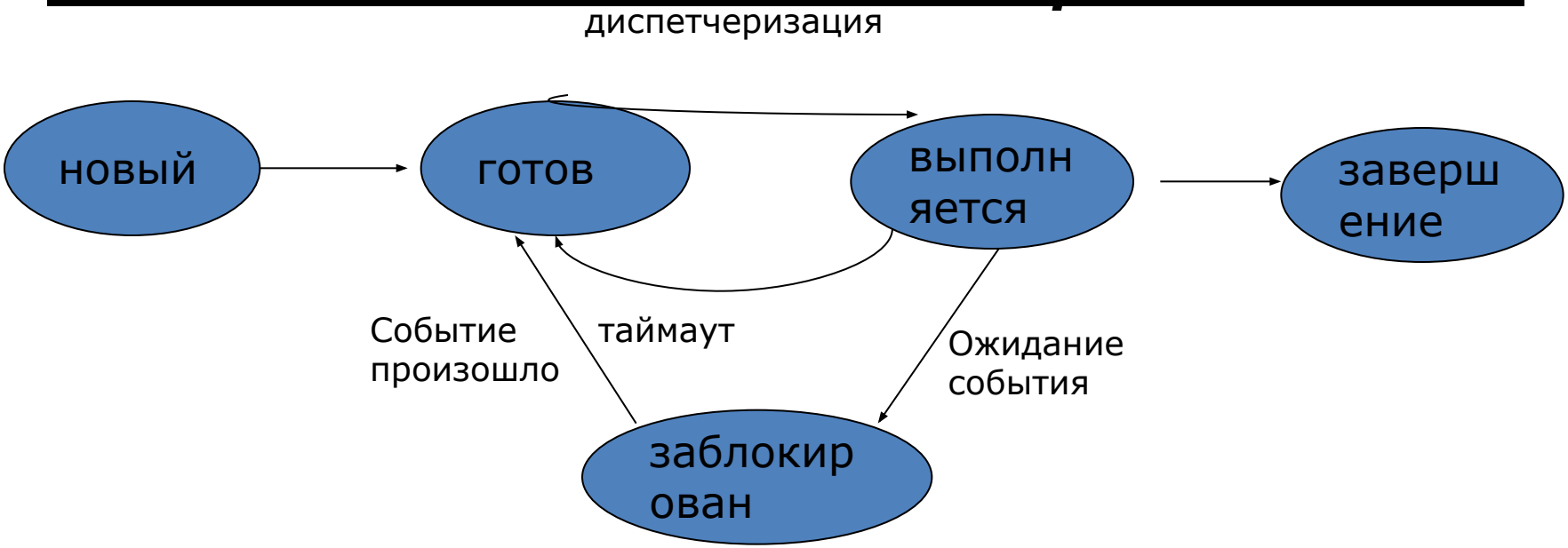
1. **Очередь задач**: множество всех процессов, которые есть в системе
2. **Очередь готовых**: множество всех процессов, готовых для выполнения, им можно в любой момент дать квант процессорного времени и они будут выполняться.
3. **Очередь ожидающих**: множество всех заблокированных процессов.

В процессе жизненного цикла процессы перемещаются между этими очередями.

Управление процессами



Модель состояния процесса 2



Модель состояния процесса №2

Новая модель, состоящая из **5 состояний**, очень близка к сегодняшним ОС

1. «**Новый**» – процесс создан, но он еще не помещен ОС в пул выполняемых процессов. Создана структура PCB, но процесс еще не загружен в память (т. е. создан PCB и пустое адресное пространство)
2. Если новый процесс принимается ОС, если соблюдается, все права доступа, то процесс помещается в состояние «**Готовность**»: процесс полностью готов к выполнению, т.е. может получить управление и непосредственно начать работать. Все загружено в память, инициализированы данные, стек, куча.
3. «**Выполнение**» - процесс исполняется.
4. «**Блокировка**» - процесс ожидает внешнего события (может быть вв/выв).
5. «**Завершение**» - процесс удаляется из пула выполненных процессов, он закончил работу. Процесс помечается как «завершенный». Диспетчер будет выполнять работу по очистке процесса. На данном этапе проходит работа по освобождению памяти, закрытию ресурсов процесса (вв/выв, файлов…).

Задачи динамического планирования, т.е. наиболее эффективного распределения ресурсов, возникающие практически при каждом событии, называются диспетчеризацией.

Долгосрочный планировщик решает, какой из процессов, находящихся во входной очереди, должен быть переведен в очередь готовых к выполнению процессов в случае освобождения ресурсов памяти.

Краткосрочный планировщик решает, какая из задач, находящихся в очереди готовых к выполнению, должна быть передана на выполнение.

Основные стратегии планирования:

- по возможности заканчивать вычисления в том же порядке, в котором он были начаты;
- отдавать предпочтение более коротким задачам;
- предоставлять всем пользователям одинаковые услуги, в том числе и одинаковое время ожидания.

В соответствии с алгоритмами, основанными на квантовании, смена активного процесса происходит, если:

- процесс завершился и покинул систему,
- произошла ошибка,
- процесс перешел в состояние ***ожидание***,
- исчерпан квант процессорного времени, отведенный данному процессу.

ОС выполняет следующие функции:

- определяет момент снятия с выполнения текущей задачи;
- сохраняет контекст текущей задачи в дескрипторе задачи;
- выбирает из очереди готовых к выполнению задач следующую;
- загружает контекст выбранной задачи;
- запускает выбранную задачу на исполнение.

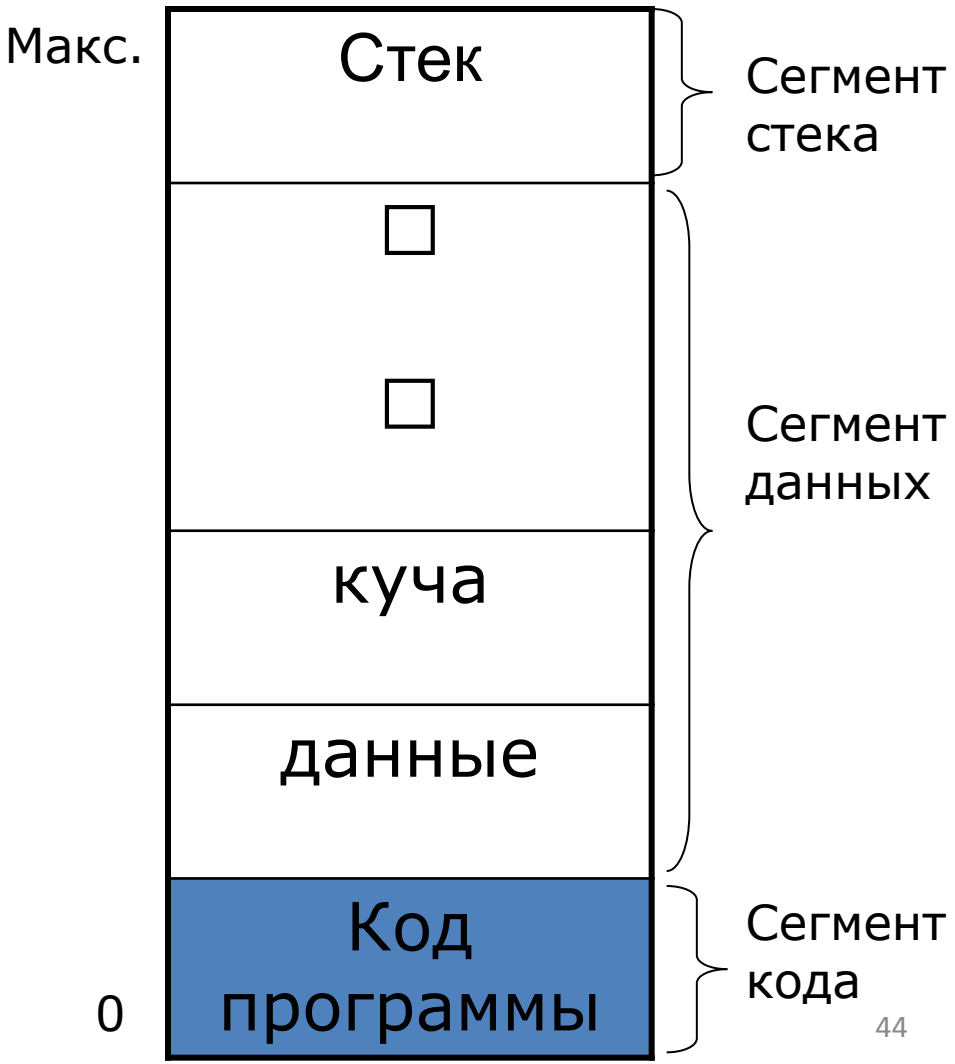
Управление процессами

Операционная система поддерживает обособленность процессов: у каждого процесса имеется свое виртуальное адресное пространство, каждому процессу назначаются свои ресурсы - файлы, окна, семафоры и т.д.

Обособленность нужна для защиты одного процесса от другого, поскольку они, совместно используя все ресурсы машины, конкурируют с друг другом. В общем случае процессы принадлежат разным пользователям, разделяющим один компьютер.

Процесс (физическое представление)

- Внизу 0 адрес, сверху максимальный. На максимуме расположен стек, затем куча, которые растут в противоположных направлениях, данные и код программы.
- Важно понимать, что каждый процесс обладает своим адресным пространством. На схеме виртуальное адресное пространство начинается в нуля и заканчивается неким максимумом, которое состоит из сегментов : **кода, данных и стека.**



Процесс (физическое представление)

При запуске программы (например MS Word) в ОС происходит следующее:

1. **Выделяется место в памяти**.

Каждый процесс выполняется в собственном виртуальном адресном пространстве, которое состоит:

1. Сегмента стека –используется для вызовов функций и системных вызовов

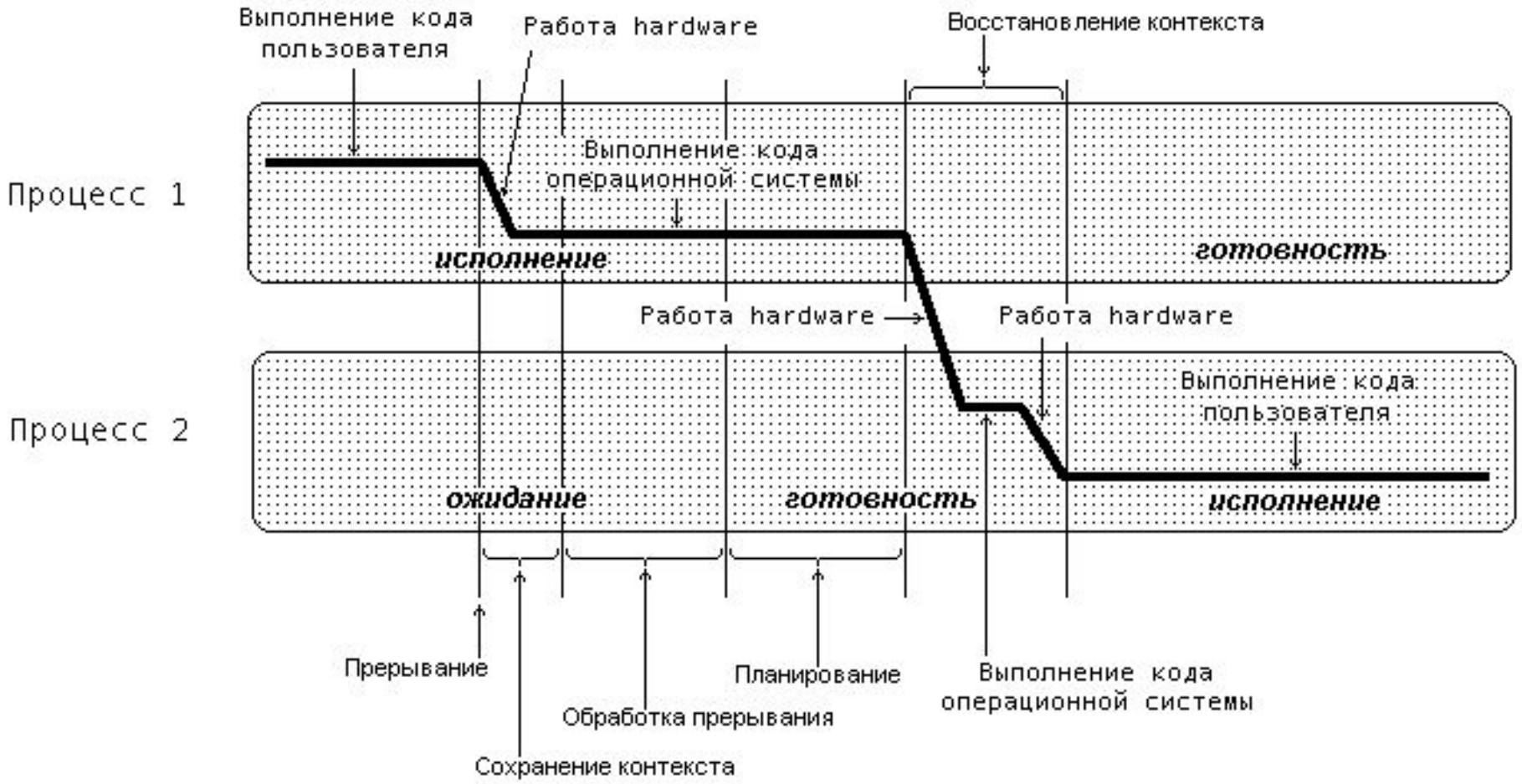
2. Сегмента данных – переменные статические и динамические, выделяемые из кучи(все что нужно для работы)

3. Сегмента кода – код программы, обычно предоставляется доступ «только для чтения»

Запуск одной и той же программы несколько раз порождает новые процессы, у каждой из которых свое виртуальное адресное пространство и окружение. Т.е. эта схема будет у каждого запущенного процесса.

При мультизадачности повышается пропускная способность системы, но отдельный процесс никогда не может быть выполнен быстрее, чем если бы он выполнялся в однопрограммном режиме (всякое разделение ресурсов замедляет работу одного из участников за счет дополнительных затрат времени на ожидание освобождения ресурса).

Управление процессами

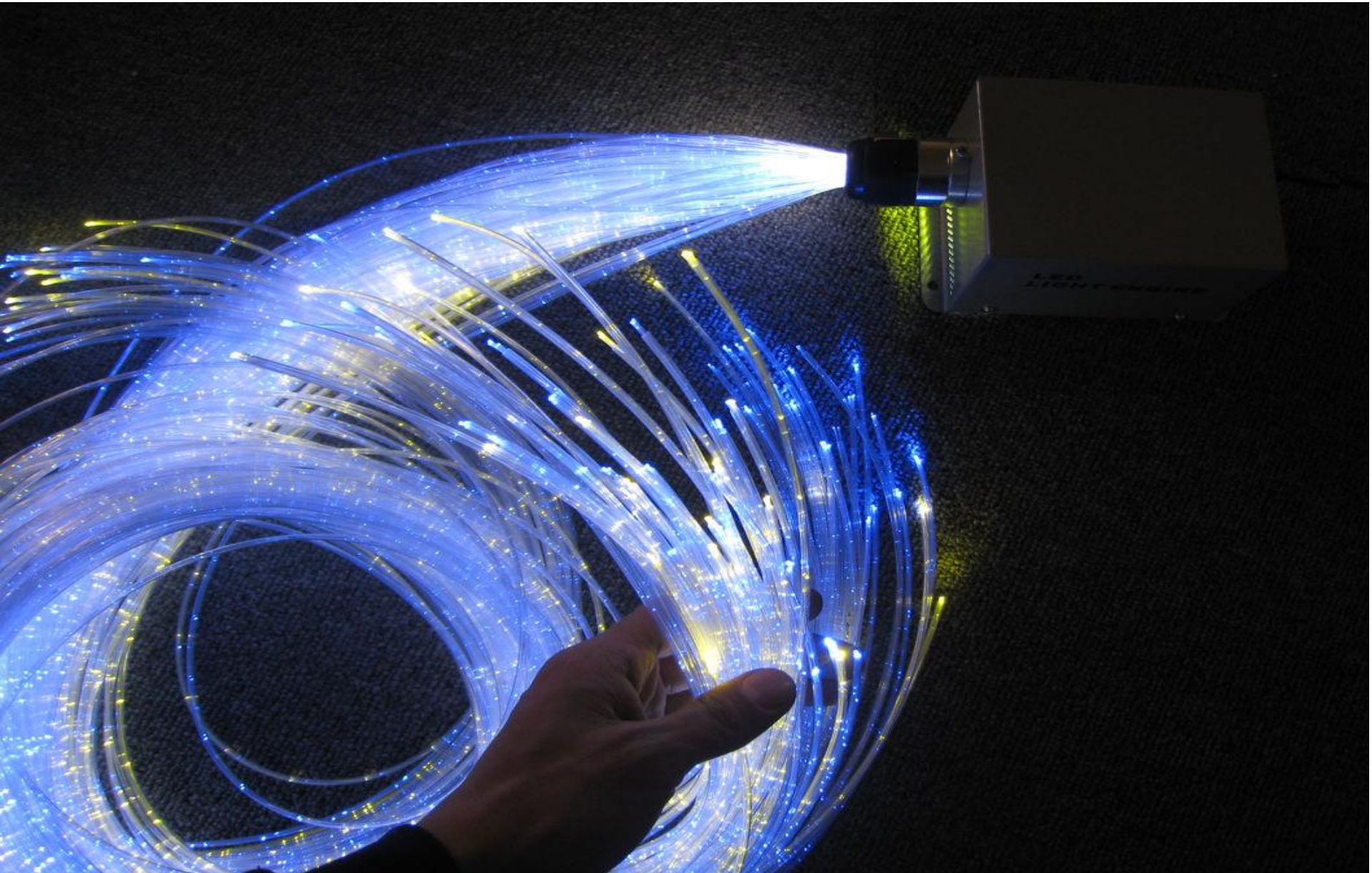


Этапы создания процесса

Чтобы создать процесс надо:

1) Присвоить уникальный идентификатор новому процессу
2) Выделить ему место в памяти (для программы, данных и стека) – физически в памяти выделяются некоторые страницы (создается образ процесса на диске)
3) Инициализировать PCB (блок управления процессом)
4) Добавить процесс в очередь «готовых» к выполнению.

Потоки



ПОТОКИ (НИТИ)

Процесс состоит как минимум из:

- **1) Адресное пространство** (набор инструкций – код программы, данные)
- **2) Состояние процесса выполнения**
  Характеризуется состоянием регистров ЦП:
  - Счетчик команд (регистр IP)
  - Указатель стека (SP)
  - Др.регистры ЦП
- **3) Множество ресурсов ОС**, которыми процесс владеет в данное время (открытые файлы, сетевые соединения) .

Все это находится в одном понятии процесса. Но не всегда это хорошо. Для трех несвязанных между собой процессов надо выделять 3 различные области.

Приходит на помощь понятие **ПОТОК.**

ПОТОКИ (НИТИ)

Потоки нужны для двух вещей – для параллелизма и одновременности.

Параллелизм – это физически одновременное выполнение для достижения наибольшей производительности (например, между двумя ядрами)

Одновременность – логическое и/или физическое одновременное выполнение (есть один ЦП, на нем одновременно выполняется несколько программ – многозадачная ОС).

Потоки нужны в обоих случаях для эффективного использования. В самом простом варианте, чтобы достичь параллелизма – использование множества процессов – программы изолированы друг от друга в разных процессах, поэтому параллелизм есть.

Потоки – другой способ достичь параллелизма. Потоки работают внутри одного процесса. Все потоки процесса имеют одно адресное пространство и те же ресурсы ОС. У каждого потока есть свой стек и свое состояние ЦП.

Параллелизм

Примеры:

Веб-сервер, который для каждого пользовательского процесса создает новый процесс, т.е. должен обслуживать несколько запросов параллельно.

Ожидая данных по запросу клиента из БД сервер в это же время мог бы загрузить данные с диска для другого клиента и обработать запрос третьего клиента.

Веб-браузер – в момент обращения к веб- странице, он мог бы параллельно загружать данные из различных источников.

Некоторая вычислительная программа использующая физический параллелизм – например, когда нужно обработать большой массив данных.

Параллелизм

В каждом из этих примеров параллелизма есть **общее:**

- Один код
- Доступ к одним данным
- Один уровень доступа
- Одно множество ресурсов.

разное:

- Стэк и указатель стэка (регистр SP)
- Счетчик инструкций (регистр IP), указывающий на следующую инструкцию в коде программы
- Множество регистров ЦП

Параллелизм

Как достичь параллелизма?

Используя знания о процессах, можно:

- Можно породить сразу несколько процессов
- Заставить каждый из них создавая свое адресное пространство копировать в него одну и ту же информацию

Неэффективно: Затраты на PCB, таблицы страниц, создание ОС структур данных, копирование адресного пространства, синхронизировать доступ.

Решение – ввести понятие **Потока**

Отделить понятие процесса (адресного пространства, ресурсов ОС) от **минимальной нити, потока управления**, т.е. состояния стека и регистров ЦП.

Иногда такое состояние называют «легким» процессом или **потоком**.

Процессы и потоки

Большинство современных ОС поддерживает два объекта:

- **Процесс**, который определяет адресное пространство и общие атрибуты процесса.
- **Поток**, который определяет последовательный поток выполнения в рамках процесса.

Поток привязывается к одному **процессу** (одному адресному пространству)

- Может быть много потоков в одном адресном пространстве
- Легкий доступ к общим данным
- Создание потоков занимает очень мало времени

ПОТОКИ стали единицей планирования ОС

Процессы – всего лишь контейнер, в котором выполняются **потоки**.

Процесс – это непосредственно контейнер, а **поток** – это нити выполнения, которые у него есть внутри.

Многопоточность

Многопоточность полезна для :

- обработки одновременных событий
- построение параллельных программ.

Поддержка многопоточности – разделение понятие процесса от минимального потока управления.

- Для параллельного потока выполнения не нужно создавать новые процессы.
- Работает быстрее, меньше требования к памяти.

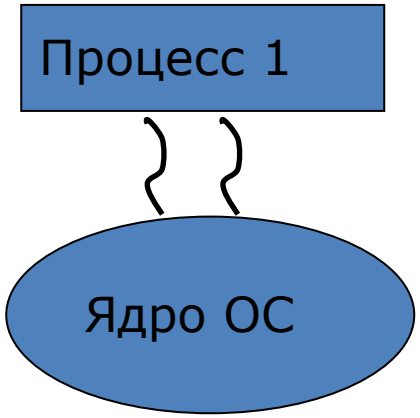
Без потоков: «Процесс»= адресное пространство + ресурсы ОС+ подразумевался единственный поток

С потоками: «Процесс»= адресное пространство + ресурсы ОС+ все потоки принадлежащие процессу

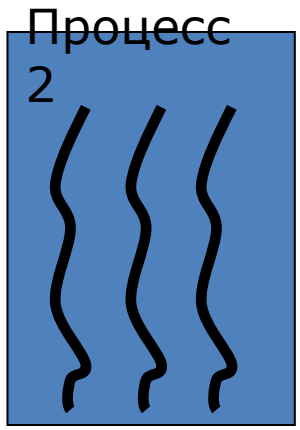
Какие бывают потоки?

Делают двумя способами:

1) **На уровне ядра** (есть функции ядра для создания нового потока)
   - выделяется стек выполнения внутри адресного пространства процесса
   - создает и инициализирует блок управления процессом
2) **На уровне пользователя**
   -



Ядро ОС управляет потоками



Есть спец. *Библиотека*, которая управляет потоками вне ядра самостоятельно

Отличие потоков от процессов

Отличия *потоков* от традиционных *процессов* многозадачной операционной системы:

- *процессы*, как правило, независимы, тогда как *потоки* существуют как составные элементы процессов
- *процессы* несут значительно больше информации о состоянии, тогда как несколько *потоков* внутри процесса совместно используют информацию о состоянии, а также память и другие вычислительные ресурсы
- *процессы* имеют отдельные адресные пространства, тогда как *потоки* совместно используют их адресное пространство
- процессы взаимодействуют только через предоставляемые системой механизмы связей между процессами
- переключение контекста между потоками в одном процессе, как правило, быстрее, чем переключение контекста между процессами.
- Такие системы, как Windows и OS/2, как говорят, имеют «дешёвые» потоки и «дорогие» процессы. В других операционных системах разница между потоками и процессами не так велика, за исключением расходов на переключение адресного пространства.

Стратегии обслуживания процессов

Дисциплина FCFS (first come – first served) реализует стратегию обслуживания «**по возможности заканчивать вычисления в порядке их появления**». Задачи обслуживаются в порядке очереди, т.е. в порядке их появления. Задачи, приостановленные для ожидания какого-либо ресурса, после перехода в состояние готовности становятся в эту очередь перед задачами, которые еще не выполнялись.

Образуются две очереди:

- новые задачи;
- ранее выполнявшиеся, но попавшие в состояние ожидания.

Дисциплина FCFS не требует внешнего вмешательства в ход вычислений и перераспределения процессорного времени. По классу диспетчеризации (вытесняющие и не вытесняющие) дисциплина FCFS относится к не вытесняющим.

Дисциплина обслуживания SJN (shortest job next) требует, чтобы пользователи указывали предположительное время выполнения. Диспетчер задач сравнивал указанное время с реальным временем выполнения и, если время выполнения превышало указанное, то помещал это задание в конец очереди.

Дисциплина обслуживания SRT (shortest remaining time), основана на том, что выбираемое на исполнение задание требует меньше всего времени для своего завершения.

Перечисленные три дисциплины обслуживания могут использоваться для пакетных режимов работы, когда не важно время отклика.

Для интерактивной работы надо обеспечить приемлемое время реакции системы и равенство в обслуживании, если система мультитерминальная. Интерактивные задания должны иметь преимущество перед фоновыми. Эти условия решены в дисциплине RR (round robin – круговая, карусельная)

Дисциплина обслуживания RR предполагает, что каждая задача получает процессорное время порциями (**квантами**). После окончания выделенного кванта времени задача снимается с исполнения и на выполнение выбирается следующая задача.

Величина кванта времени выбирается как компромисс между приемлемым временем реакции системы на запросы пользователей и накладными расходами на частоту смены контекста задач.

Диспетчеризация без

перераспределения процессорного времени, **не вытесняющая многозадачность**, – это такой способ диспетчеризации процессов, при котором активный процесс выполняется до тех пор, пока он сам, по своей инициативе, не отдаст управление диспетчеру задач для выбора из очереди другого, готового к исполнению процесса. Дисциплины обслуживания **FCFS, SJN,**

При **не вытесняющей многозадачности** механизм разделения процессорного времени распределен между ОС и прикладной программой. Диспетчер задач формирует очереди и выбирает задачу на исполнение.

Диспетчеризация с

перераспределением процессорного времени между задачами, **вытесняющая многозадачность**, – это такой способ, при котором решение о переключении процессора с выполнения одного процесса на выполнение другого процесса принимается диспетчером задач, а не самой активной задачей. Дисциплина **RR** и аналогичные ей относятся к вытесняющим.

Критерии для сравнения алгоритмов диспетчеризации:

- использование (загруженность) центрального процессора;
- пропускная способность – количество процессов, выполняющихся в единицу времени;

Критерии для сравнения алгоритмов диспетчеризации (продолжение):

- время оборота – интервал времени от момента появления процесса во входной очереди до момента его завершения (время ожидания во входной очереди + время ожидания в очереди готовых к выполнению процессов + время ожидания в очередях к оборудованию + время выполнения в процессоре + время ввода/вывода);

Критерии для сравнения алгоритмов диспетчеризации (продолжение):

- время ожидания – суммарное время нахождения процесса в очереди готовых к выполнению процессов;

- время отклика – время от момента попадания процесса во входную очередь до момента первого обращения к терминалу

Главные причины уменьшения производительности системы:

- накладные расходы на переключение процессора (переключения контекстов задач, перемещения страниц виртуальной памяти, обновление данных в кэш-области);

- переключение на другой процесс в тот момент, когда текущий процесс выполняет критическую секцию, а другие процессы активно ожидают входа в свою критическую секцию.

Методы повышения производительности:

- совместное планирование, все потоки одного приложения одновременно выбираются для выполнения процессорами и одновременно снимаются с них;
- находящиеся в критической секции задачи не прерываются, а активно ожидающие входа в критическую секцию задачи не выбираются до тех пор, пока вход в секцию не освободится;

Введение механизма динамических приоритетов позволяет реализовать быстрое выполнение коротких задач и гарантировать выполнение любых запросов. Эта дисциплина используется в ОС UNIX.

Каждый процесс имеет два атрибута приоритета, с учетом которого распределяется процессорное время между исполняющимися задачами:

- **p_cpu - текущий приоритет, на основе которого осуществляется планирование;**
- **p_ nice - заказанный относительный приоритет (nice number).**

p_nice назначается пользователем явно или формируется по умолчанию с помощью системы программирования.

p_cpu формируется диспетчером задач (планировщиком разделения времени) и называется системной составляющей или текущим приоритетом.

Текущий приоритет процесса варьируется в диапазоне от 0 (низкий приоритет) до 127 (высокий приоритет).

Процессы, выполняющиеся в режиме задачи, имеют более низкий приоритет (0 – 65), чем в режиме ядра (66 – 95, системный диапазон).

Приоритеты в диапазоне 96 – 127 относятся к процессам с фиксированным приоритетом.

Для принятия решения о выборе следующего запускаемого процесса планировщику необходима информация об использовании процессора.

Составляющая приоритета p_cpu уменьшается обработчиком прерываний по таймеру по каждому тику таймера. Когда процесс выполняется в режиме задачи, его текущий приоритет линейно уменьшается.

Каждую секунду процессор пересчитывает приоритеты процессов, готовых к выполнению, что приводит к перемещению процессов в более приоритетные очереди и повышает вероятность их последующего запуска.

Данный алгоритм планирования обеспечивает:

- интересы низкоприоритетных процессов, так как в результате длительного ожидания их приоритет и вероятность выполнения увеличиваются;
- более вероятный выбор интерактивных процессов по сравнению с вычислительными.

Управление процессами