Процессы и потоки

Процесс

Рассмотрим ключевые понятия, такие как **процессы и потоки.** Посмотрим, как все это исторически развивалось и к чему привело сейчас. Начнем с самого фундаментального понятия в ОС – процесс, на нем строится все остальное.

- **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**.**

Разберем из чего же состоит процесс.

Процесс

Процесс состоит из трех основных компонент:

- 1) Исполняемого кода
- 2) Ассоциируемых с ним данных, необходимых для выполнения этой программы
- 3) Контекста – информация для ОС, необходимая для управления процессом. Эта информация используется для переключения м/у процессами, для сохранения и восстановления состояния процесса.
  - Номер процесса
  - Регистры ЦП
  - Содержимое стека
- Т.О. Контекст – это основа для переключения процессов
- ОС ведет список всех процессов, находящихся в системе. Он может усложняться, а может быть самым простым.

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.

Как все этим управлять? Структура управления процессами в ОС

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). Все это хранится в ОС в определенном универсальном понятии **Блок управления процессом (Process Control Block – PCB).** Описывает свой процесс, которому он принадлежит и его текущее состояние
- **Образ процесса (Process Image) –** на предыдущей схеме – память, выделенная для процесса

Блок управления процессом (Process Control Block – PCB).

Данный блок постоянного размера для все процессов в ОС.

Содержит всю информацию, необходимую для осуществления над процессом любых действий – приостановки, последующего восстановления процесса, выгрузки на диск и загрузки с диска.

Идентификаторы процесса

- Номер процесса (так называемый PID — Process IDentificator)
- Информация о пользователе

Состояние процесса – регистры, указатели стека.

Состояние процесса

- Информация для планировщика – приоритет которым обладает данные процесс
- 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.

Диспетчеризация

- **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: как минимум это ожидание вв/выв, выполнение, готовность к выполнению, и еще можно придумать ряд дополнительных промежуточных состояний.

Модель состояний процесса

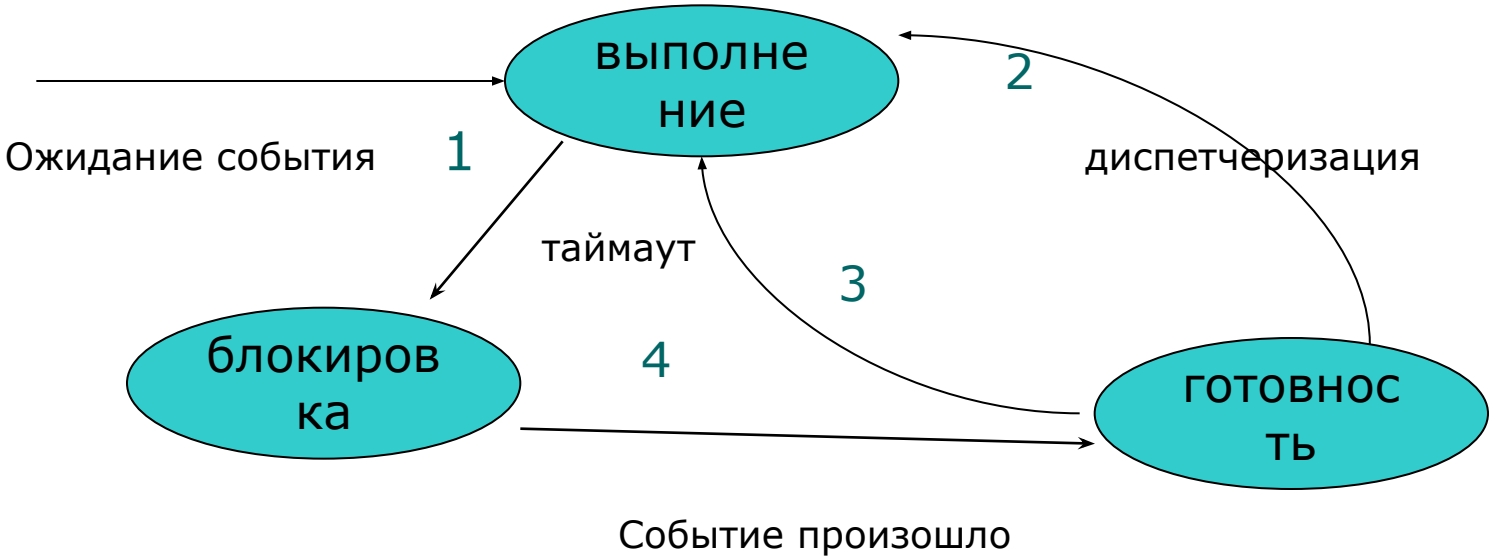
1. Процесс заблокирован для вв/выв
2. Диспетчер планирует другой процесс
3. Диспетчер планирует этот процесс
4. Вв/выв произошел, процесс возобновляет выполнение

Модель состояний процесса

Конечная цель любой ОС – выполнить какую-либо работу, задачу, ответить на запрос пользователя…

Можно выделить три основные состояния процесса

- Выполнения (исполняется на ЦП)
- Готовности (временно остановлен)
- Блокировки (ожидает внешнего события)



Модель состояния процесса

Любой процесс находится в нескольких состояниях, в самом простом варианте можно выделить 3 состояния:

1) Предположим, что любой процесс начинает свою “жизнь” с состояния **«Выполнение»**.

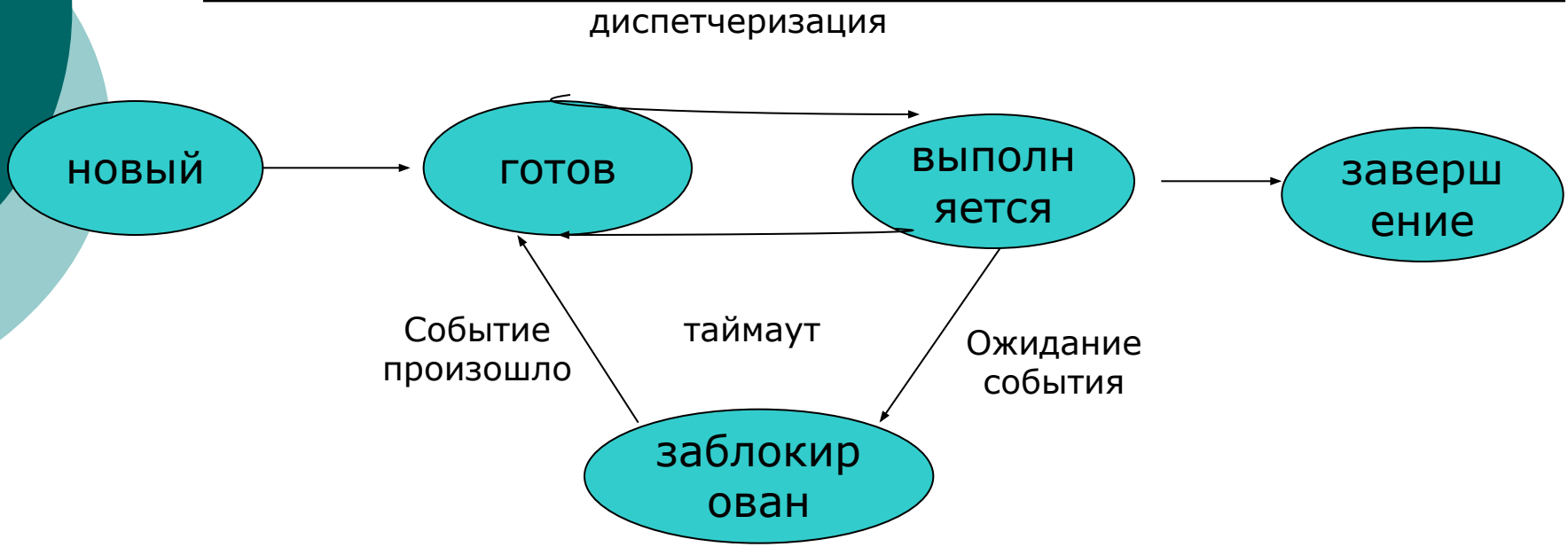
2) Если был сделан запрос на вв/выв, он осуществляется с некоторой задержкой (вв/выв работает медленней чем ЦП), значит появляется некоторое ожидание события(т.е. ожидание вв/выв). Процесс в это время находится в заблокированном состоянии **«Блокировка».** Ему диспетчер не выделяет времени ЦП, ибо это бесполезно, процесс не может делать какую-либо работу пока не выполнится вв/выв.

3) Далее, когда произошел вв/выв процесс уже может что-то обработать, его состояние изменяется в состояние **«Готовность»** (процесс должен показать диспетчеру о своей готовности работать)**.** Диспетчер при очередном переключении м/у процессами видит, что есть процесс в состоянии «Готовность» и переключает его в состоянии «**Выполнение**». В этом состоянии «Выполнение», диспетчер передает процессу квант времени ЦП – начинается непосредственное выполнение. Этот круг замкнутый.

Еще одно состояние, когда процесс попадает в «Готовность» - когда квант процессорного времени, отведенный на выполнение истекает, то происходит так называемый **таймаут,** процесс из состояния **«Выполнение»** переходит в **«Готовность»** и диспетчер в это время передает управление другому процессу.

Диспетчеру системы нужно знать, что происходит с общими событиями: с вв/выв, синхронизацией, др.

Модель состояния процесса 2



Новая модель состоит из 5 состояний, эта модель 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 все соблюдается, все права доступа, то процесс помещается в состояние «**Готовность**»: процесс полностью готов к выполнению

Модель состояния процесса №2

Новая модель состоит из **5 состояний**, эта модель 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 все 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 (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…).

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 м/у этими очередями.

Очередь задач Windows

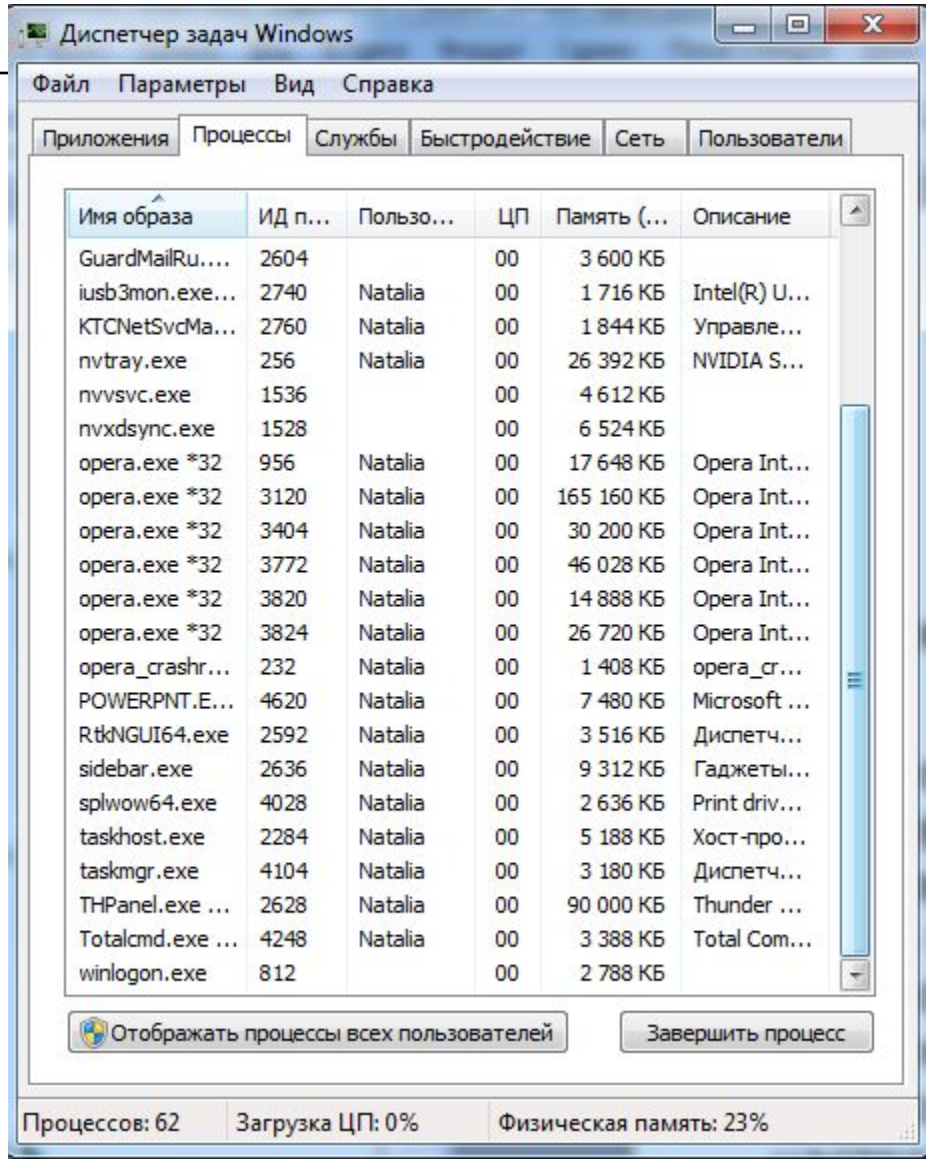
Показывает множество всех процессов в ОС.

Исторически эта задача называется TackMenager – менеждер задач. Он отображает все процессы, которые есть в ОС, правда состояний процесса (готовность, ожидание, выполнение) мы не видим. Видим только общий список., можно настроить видимость колонок списка.

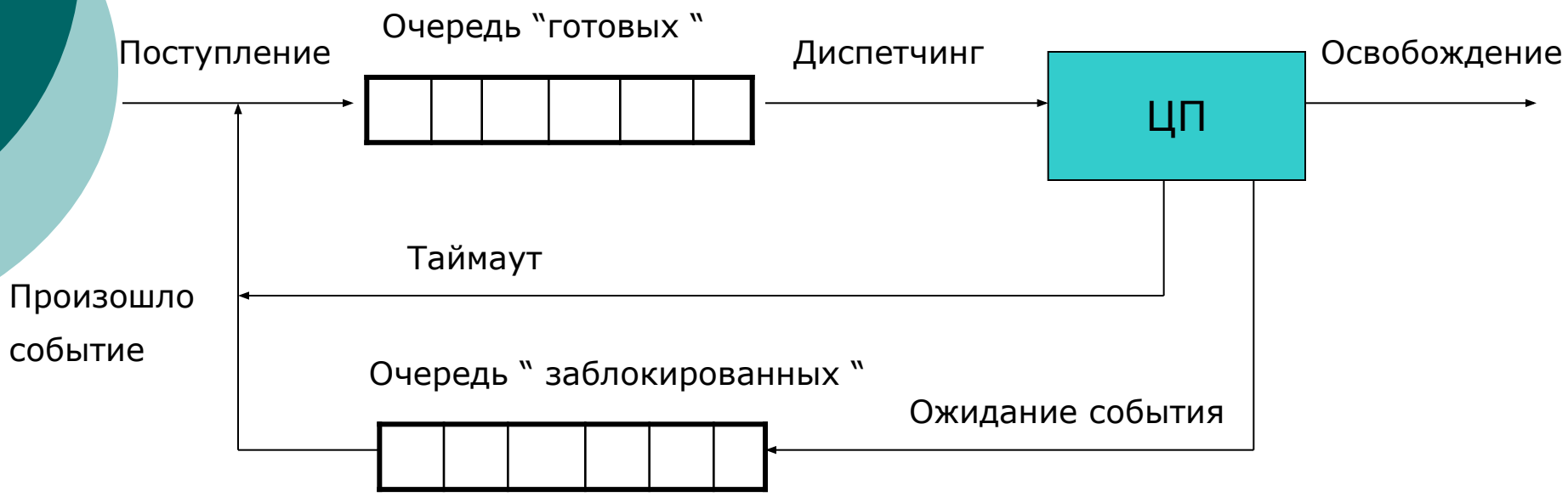
В Windows очень сложно определить, сколько памяти занимает процесс (много параметров).

Ядро в Windows представлено как процесс с идентификатором 4(PID), наименьший 0, но он занятый.

В Linux ядро не является процессом.



Управление процессами



Управление процессами

Используется одна очередь «готовых» и одна очередь «заблокированных» процессов

При этом есть следующие недостатки:

- При наступлении события, все ожидающие этого события процессы нужно переместить из «заблокированных» в очередь «готовых»
- Нужно смотреть в очереди все процессы, которые ожидают именно того события, которое произошло и после этого переместить их в очередь «готовых». После этого им будет дано диспетчером время ЦП и они будут выполняться.
- ОС нужно посмотреть все заблокированные процессы в очереди, чтобы выбрать правильный. Для этого придется все время перебирать очередь заблокированных по кругу.

Управление процессами

Событие –

1) Ожидание вв/выв
2) Синхронизация – когда процессу нужно получить доступ к какому-либо ресурсу, который используется другим процессом, ему приходится ждать.

Каким то образом данную схему необходимо улучшить. Наиболее логично – каждому событию в ОС должна быть своя очередь заблокированных по этому событию. Тогда, выбирая события видим все процессы, которые его ожидают, сразу же все переносим в очередь «готовых» и все работает быстро и замечательно.

Но на практике такое невозможно, сильно много будет очередей заблокированных процессов, поэтому применяются некоторые компромиссы - используется несколько очередей, отсортированных по типу ожидаемых событий.

РЕШЕНИЕ: использовать несколько очередей заблокированных процессов. Уменьшить нагрузку и не перебирать весь список заблокированных процессов.

Создание процесса

Самыми первыми создаются процессы в момент загрузки ОС

1) **Загрузка системы**

При инициализации системы создаются несколько исходных процессов

- В Unix – это процессы «демоны» Sched(pid0) init(pid1) – др. высокоуровневые (веб-сервер, емейл-сервер). Ядро – не процесс. Идентификаторы в Unix идут последовательно с приращением 1.
- В Windows NT ядро – это системный процесс System(Pid4), далее загружаются система управления подсистемами smss. Идентификаторы в Win идут с приращением 4, идентификатор 0 зарезервирован, системный процесс - 4.

Создание процесса

2) Текущий процесс порождает дочерний процесс

- Есть интересная особенность в построении ОС- так называемая иерархия процессов, исторически она существовала в ОС класса Unix. Там каждый процесс имеет строгое родство – есть родительский процесс и м.б. дочерний процесс.
  - Пример веб-сервер может порождать дочерний процесс для обработки нового запроса(это не есть хорошо, работает медленно)
  - В Unix процесс **init** отслеживает авторизации пользователя для того, чтобы запустить оболочку(нов.процессы)

3) Пользователь создает новый процесс

- Пользователь вызывает команду из текстовой оболочки или запускает новую программу из графической оболочки. Это создает новый процесс, родитель которого – оболочка ОС.

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.

Иерархия процессов (ОС Unix)

- Строгая иерархия между процессами: дочерний и родительский всегда взаимосвязаны
- Идентификатор командной строки (оболочка пользователя Shell) является родительской для всех процессов, которые пользователь запускает из командной строки.
  - Существует понятие **Группа процессов** – любой процесс может принадлежать какой – либо группе, посылаем сигнал группе и его выполняют все процессы в группе.
  - Если пользователь посылает сигнал (напр. SIGKILL на завершение) группе процессов, то сигнал доставляется каждому процессу из группы. Это достаточно удобно для работы с группами процессов.

Создание процесса (ОС Unix)

Процессы создаются через Fork() / exec()

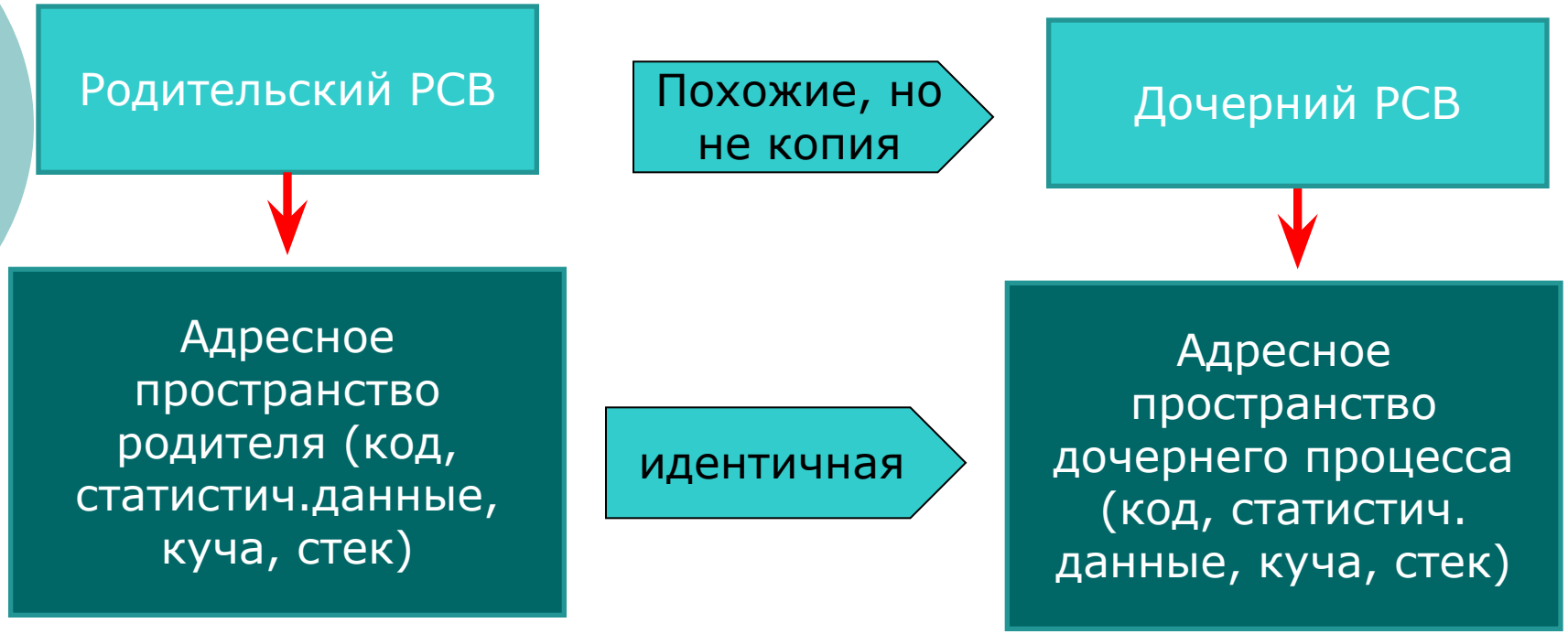
1) Начинается с fork(), он создает точный клон вызывающего процесса, так называемый «дочерний» процесс
2) Менеджер исполнения exec() заменяет образ процесса этого клона новой программой, которая должна быть выполнена.

Так сложилось исторически и другого способа породить процесс нет, поэтому существует иерархия, как основа основ.

После создания у родительского и дочернего процессов возникают собственные разные адресные пространства.

Некоторые ресурсы м.б. общими (например открытые файлы). Поэтому системный вызов fork() возвращается дважды – один раз в родительский процесс, другой раз во вновь созданный.

Создание процесса (ОС Unix) ключевой механизм



После вызова Fork() создается почти такой же PCB (блок упр-я), только будет другой идентификатор у процесса, адресное пространство аналогичное.
Идентификатор берется из таблицы(она есть у каждой ОС).
Недостаток Unix - Fork() работает очень медленно, надо вначале создать точную копию процесса, а затем уже ее заменить на новый процесс.

Создание процесса (ОС Unix) Linex

Как создать новую программу, а не еще одну копию старой?

Легко. Вначале fork(), потом exec().

Exec() не создает новый процесс, а заменяет данные текущего процесса новыми данными

У такой модели есть недостатки:

-Folk() очень медленный, нужно создавать полную копию всего, а потом ее заменять

Решение было найдено в функциях vfolk() copy-on-write

В Linux есть функции

Clone() – заменяет fork()/vfork()

Clone() обладает дополнительными опциями. Exec() в Linex не является системным вызовом

Execve() – единственный вызов, аналогичный по функционалу exec()

Создание процесса (ОС Windows)

В ОС Windows процессы создаются через системный вызов NtCreatProcess() – данный вызов имеет множество параметров, многие из которых «по умолчанию». Процесс порождается непосредственно по желанию, он не должен быть обязательной копией текущего, поэтому не соблюдается родство.

- **Хэндл процесса** – существительное (дословно переводится «ручка») Это сущность доступа к процессу – некоторое целое число, которое является индексом в определенной таблице и позволяет идентифицировать этот процесс – контроль доступа, права, наследование.
- Pid – это смещение (индекс) в таблице процессов

Иерархия процессов в ОС Windows

- Ее нет, все равны. Поэтому встает вопрос взаимодействия
- **Хэндл процесса** – когда новый процесс создается родительским, то родитель получает хэндл дочернего процесса и т.о. может им управлять
- Этот хендл может передаваться другим процессам (в отличии от Unix, где родительский процесс не может менять множество дочерних процессов).
- **Хэндл процесса –** идентификатор объекта процесса

Переключение между процессами (самый простой вариант)

При необходимости переключиться на другой процесс ОС выполняет «переключение контекста».

- Состояние старого процесса сохраняется в его PCB
- Состояние нового процесса восстанавливается в его PCB

Время на переключение контекста – накладные расходы ОС (чем меньше, тем быстрее работает)

Зависит от аппаратной реализации (т.е.от «железа компьютера», оптимизируется аппаратно).

Чем вызваны переключения?

События, вызывающие переключения контекста:

- Прерывания
- Исключения
- Системные вызовы

ПОТОКИ (НИТИ)

Процесс состоит как минимум из:

- **1) Адресного пространства** (набор инструкций –код программы, данные)
- **2) Состояние потока выполнения**
  Характеризуется состояние потока регистрами 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 это не есть хорошо. 3 несвязанных между собой процесса хорошо бы разделить на 3 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 для того и другого понятия 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 потоков 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
- Множество регистров ЦП

Параллелизм

Как достичь параллелизма?

Используя знания о процессах, можно**:**

- Выполнить Fork() для нескольких процессов (т.е. породить их сразу несколько)
- Заставить каждый из них отображать свое адресное пространство на одну и ту же физическую память

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

Решение – ввести понятие **ПОТОКИ**

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 адрес. пространству)

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

ПОТОКИ стали единицей планирования

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.

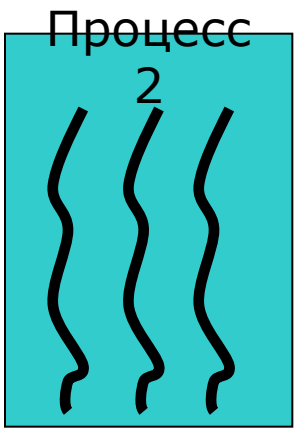
Раньше: «Процесс»= адресное пространство + ресурсы ОС+ подразумевался единственный поток

Раньше: «Процесс»= адресное пространство + ресурсы ОС+ все потоки принадлежат процессу

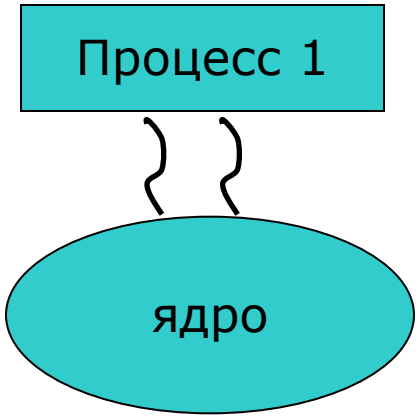
Какие бывают потоки?

Делают двумя способами:

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



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



Ядро управляет потоками

Потоки

В обеих вариантах есть + и –

1)вар. «-» нужен системный вызов при создании потока, а это время процессора. На уровне пользователя системного вызова не нужно, все управляется специальной библиотекой.

Библиотека Pthreads (положительное)

- Каждый поток представляется регистрами ЦП, стеком и небольшим блоком TCB
- Создание потока, переключение между потоками и синхронизация потоков выполняется без участия ядра
- Потоки уровня пользователя могут быть в 10-100 раз быстрее, чем потоки режима ядра

Потоки

- Множество потоков в одном адресном пространстве это хорошо
- Потоки режима ядра намного эффективнее процессов, но есть потери на системные вызовы
- Потоки режима пользователя имеют преимущества и недостатки:
  - высокая скорость и «дешевизна» создания
  - могут быть проблемы с вв/выв и блокировками, из-за того, что ядро не знает об этих потоках.

Поэтому пользовательские потоки не обрели большой популярности.

Возможно решить эти проблемы на уровне **планировщика.**

В ОС WinNT потоки реализованы на уровне ядра, что позволяет более тонко регулировать события.

Ввели fiber – вытесняющую многозадачность.

Отличие потоков от процессов

Потоки выполнения отличаются от традиционных процессов многозадачной операционной системы тем, что:

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

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.**

Управление процессами