# Управление процессами

## Операционные системы 2011/12

Татьяна Романова

10 сентября 2011 г.

# План на сегодня

- Модель процесса.
- Контекст процесса.
- Детали реализации.
- Права доступа.
- Создание и завершение процессов.
- Диаграмма состояний.
- Синхронизация.

# Материалы

- Э. Таненбаум, Современные операционные системы, глава 2 (до потоков).
- Ю. Вахалия, UNIX изнутри, глава 2.
- Р. Стивенс, С. Паго, UNIX. Профессиональное программирование, главы 7-8.
- CS162, лекция 3.

# Модель процесса

# Задачи и средства операционной системы

## Задачи:

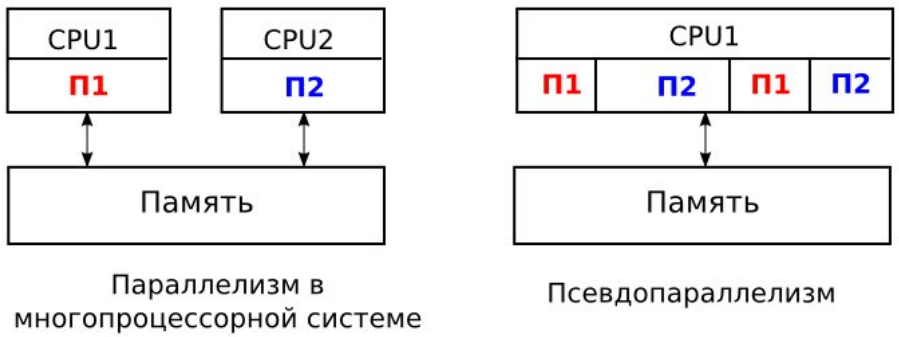
- удобный интерфейс к оборудованию;
- управление ресурсами — возможна одновременная (параллельная) работа нескольких процессов.

## Средства:

- трансляция адресов;
- два режима выполнения: режим ядра и режим задачи.

# ОС как виртуальная машина

Большинство современных систем — многозадачные.



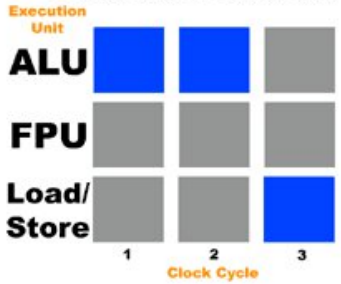

ОС обеспечивает свойства виртуальной машины: каждый процесс «думает», что он является единственной выполняемой задачей

- имеет полный доступ к устройствам;
- может выполяться неограниченное время, не прерываясь.

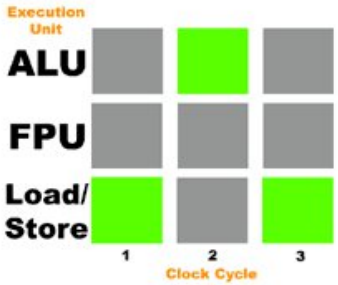
# Гиперпоточность (Hyper-threading)

- Технология Intel (Simultanios Multithreading)
- Один реальный CPU — два виртуальных.
- Регистры дублируются, вычислительные ресурсы — нет.
- Когда один поток приостанавливается (кэш-промах, ошибка в предсказании ветвления, ожидание результатов), другой работает.
- Нужна поддержка многопоточности в ОС.
- Производительность зависит от харатера вычислений.

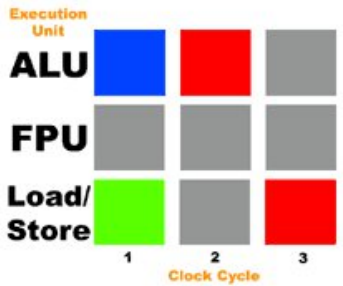
Два сложения, сохранение результата




Загрузка, сложение, сохранение




Конфликты при использовании гиперпоточности



# Понятие процесса

Процесс — последовательность инструкций, выполняющаяся в собственном адресном пространстве. Отслеживает текущую инструкцию при помощи регистра, называемого счетчиком команд (PC, program counter).

Свойства:

- работа в течение некоторого кванта времени;
- виртуальное адресное пространство;
- блокировка в ожидании ресурса;
- неопределенное время выполения.

# Процесс =? программа

Немного философии:

- Процесс больше программы:
  - программа — набор инструкций, часть адресного пространства процесса
  - редактирование в emacs лекций и кода: программа одна, процессы разные.

# Процесс =? программа

Немного философии:

- Процесс больше программы:
  - программа — набор инструкций, часть адресного пространства процесса
  - редактирование в emacs лекций и кода: программа одна, процессы разные.
- Программа больше процесса:
  - взаимодействующие процессы внутри одной программы
  - gcc запускает cpp, cc, as и ld.

# Полномочия пользователя

- У каждого пользователя в системе есть идентификатор пользователя и идентификатор группы (UID, GID).
- Для суперпользователя UID=0, GID=1.
- Каждый процесс запускается каким-то пользователем и получает те полномочия, которые доступны пользователю.
- У процесса два типа идентификаторов — реальные (определяющие, от кого запущен процесс на самом деле) и эффективные (влияют на права доступа к файлам).
- Программы с set-uid-bit'ом: при вызове exec ядро заменяет эффективные UID и GID на UID и GID владельца файла.
- Менять реальные и эффективные индетификаторы с помощью системных вызовов setuid и setgid может только суперпользователь.

# Реализация процессов

# Защита процессов друг от друга

Блок управления памятью (MMU):

- набор регистров, в которых хранятся адреса таблиц трансляции адресов работающего процесса
- переписывать регистры можно только в режиме ядра
- при переключении процессов меняются и адреса таблиц

# Что нужно сохранить при переключении?

Аппаратный контекст (или блок управления процессом, PCB)

- Программный счетчик (program counter, PC)
- Указатель стека (stack pointer, SP)
- Слово состояния процесса (processor status word, PSW), хранит инфорацию о текущем и предыдущем режимах выполнения, биты переполнения, переноса, данные о приоритетах прерываний
- Регистры MMU
- Регистры общего назначения

Переключение контекста — сохранение PCB одного процесса и загрузка в регистры данных из PCB другого.
Переключение контекста происходит при переключении между процессами, системных вызовах, вызовах обработчиков прерываний.

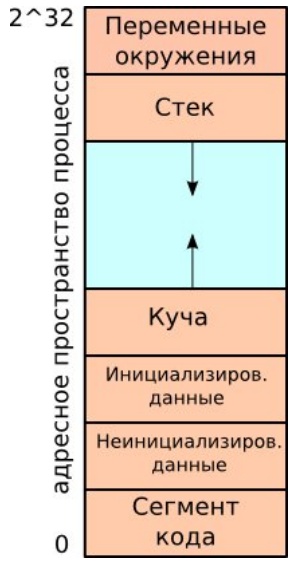
# Контекст выполнения


- В системе работает одна копия ядра.
- Контекст ядра защищен от доступа в режиме задачи, доступен только через системные вызовы.
- При системном вызове PCB сохраняется в стеке ядра.
- Данные о процессе (user-area) и стек ядра — в контексте процесса, но доступны только в режиме ядра.
- Обработка прерываний не имеет отношения к процессу, выполняется в системном контексте.

# Описание процесса

Информация, необходимая для описания процесса:

- адресное пространство и переменные окружения
- управляющая информация (область u, структура proc)
- полномочия
- аппаратный контекст (PCB)



# Управляющая информация

Необходимая только в момент выполения (область u):

- UID, GID — права доступа
- блок для сохранения PCB
- обработчики сигналов
- таблица дескрипторов открытых файлов
- таблица трансляции адресов
- стек ядра

Всегда хранящаяся в таблице процессов ядра (структура proc):

- идентификаторы процесса и сеанса
- текущее состояние процесса
- приоритеты планирования, положение в очереди планировщика
- маски сигналов
- информация об иерархии процессов (родительский, дочерние)

Пользовательский процесс не может менять управляющую информацию.

# Управление процессами

# Создание процесса (fork)

- Дочерний процесс при старте является точной копией родительского.
- Отличие — в возращаемом значении fork-а
- Последовательность действий при создании процесса:
  - новая запись в таблице процессов, часть полей копируется (UID, маски сигналов), часть обнуляется (время испльзования CPU), часть изменяется (PID, PPID).
  - размещение в памяти таблиц трансляции адресов
  - дублирование данных, стека; соответствующая модификация таблиц трансляции адресов
  - инициализация аппаратного контекста (копирование регистров)
  - поместить новый процесс в очередь планировщика
  - установить результат возврата вызова fork: 0 для дочернего, новый PID для родительского

# Оптимизация вызова fork

- Копирование при записи: области памяти доступны только для чтения, при попытке изменения ядро делает копию страницы.
- vfork: дочерний процесс выполятестся в адресном пространстве родителя, родительский ждет, когда дочерний вызовет exec или exit.
- rfork (FreeBSD, в оригинале Plan9), clone (Linux) — разделяемые и копируемые ресурсы управляются набором параметров.

# Пример

```
#include <unistd.h>
#include <stdio.h>
int a = 1;
char buf[] = "unbuffered\n";
int main()
{
   write(1, buf, sizeof(buf) - 1);
   printf("buffered\n");
   int b = 10;
   if (fork() == 0)
     ++a, ++b;
   else
     sleep(2);
   printf("a=%i, b=%i\n", a, b);
}
```

# Запуск новой программы (exec)

exec освобождает (или разблокирует, в случае vfork-а) адресное пространство текущего процесса и загружает туда содержимое новой программы.
Последовательность действий:

- разбирает путь к файлу, проверяет тип файла и права доступа.
- возможно, заменяет UID и GID
- освобождает старое адресное пространство
- выделяет новые таблицы трансляции адресов, устанавливает новое адресное пространство
- инициализирует аппаратный контекст, PC получает значение точки входа

# Связка fork-exec

Преимущества:

- fork может быть полезен сам по себе
- Между вызовами fork и exec можно сделать много вещей:
  - перенаправление ввода/вывода
  - изменение прав процесса (UID, GID)
  - сброс обработчико сигналов

Недостатки:

- Один вызов CreateProcess на первый взгляд выглядит логичнее
- При плохой реализации — лишнее копирование.

# Завершение процесса (exit)

Функция exit может быть вызвана явно (из кода программы) или неявно (при завершении по сигналу).
Последовательность действий:

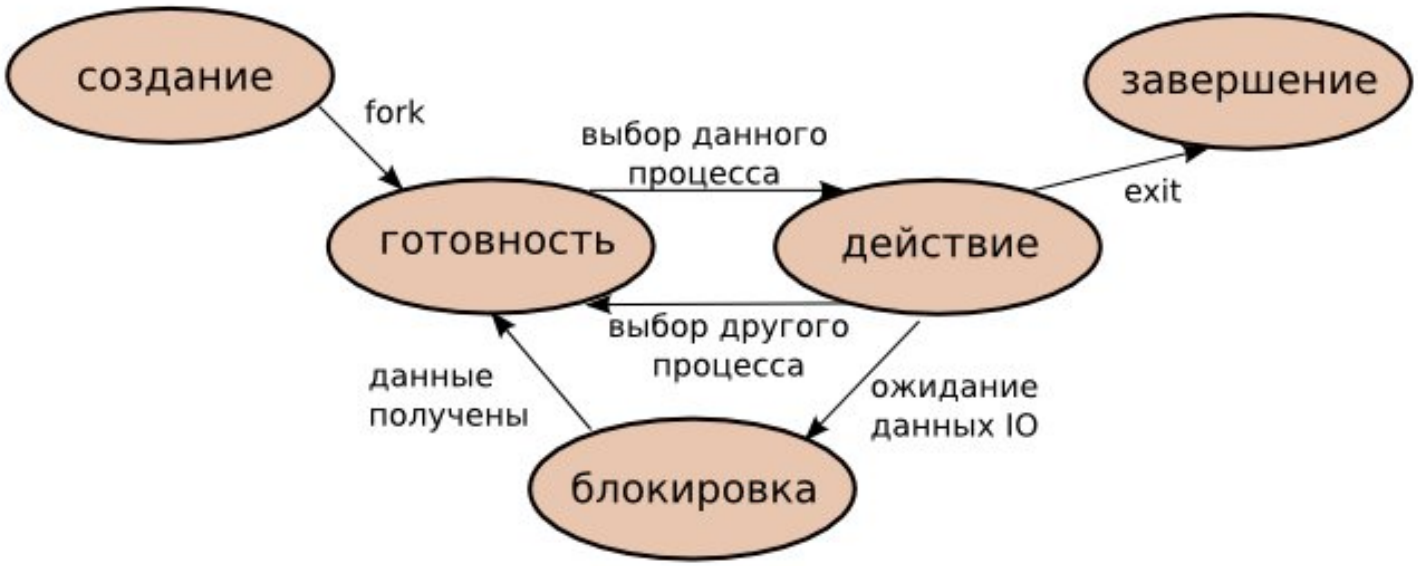
- закрывает открытые файлы
- сохраняет данные об использованных ресурсах и статус выхода в таблице процессов
- изменяет состояние процесса на «зомби»
- устанавливает процесс init в качетсве родителя для потомков завершенного процесса
- освобождает адресное пространство
- посылает родителю сигнал SIGCHLD (возможно, будит родительский процесс)
- запускает переключение на следующий процесс

# Ожидание завершения (wait)

- wait блокирует родителя пока один из потомков не завершит работу (если на момент вызова нет уже завершившихся).
- waitpid ждет завершения конкретного потомка, параметр WNOHANG делает waitpid неблокирующим

Обе функции записывают статус выхода дочернего процесса в переменную и освобождают структуру proc.

# Диаграмма состояний

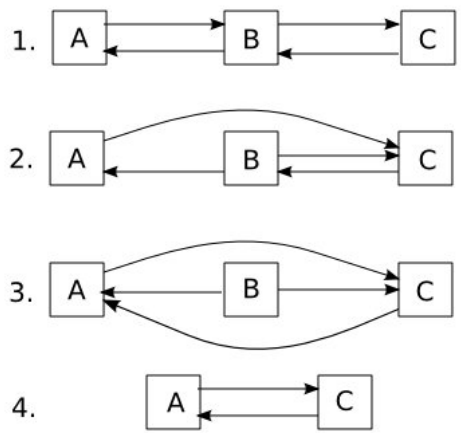


# Проблемы синхронизации

Результат работы с совместно используемыми данными зависит от порядка выполения процессов, который в общем случае неопределен.

Общие данные для процессов — находящиеся в структурах ядра.

Пример: удаление из списка, переключение на шаге 2 $\Longrightarrow$ ошибка




Решение: в режиме ядра один процесс не может быть вытеснен другим.

# Проблемы синхронизации

Некоторые проблемы остаются:

- структуры ядра, занятые заблокированным процессом (буфер чтения);
- асинхронные прерывания;
- работа на многопроцессорных системах.

# Заключение

# Итоги лекции

- В операционной системе задачи выполняются процессами.
- В многозадачных ОС работает несколько процессов параллельно.
- Процесс отдает процессор при блокировке, по таймеру, при возникновении прерываний.
- При смене выполняемых процессов необходимо переключение контекста.
- Управление процессом осуществляется системными вызовами: fork, exec, wait, exit.
- Вопросы межпроцессного взаимодействия и синхронизации очень важны!