**Нижегородский государственный университет им. Н.И.Лобачевского**

***Факультет Вычислительной математики и кибернетики***

# Операционные системы: *аспекты параллелизма*

## Процессы и потоки

Линёв А.В.

2007

# Тема обсуждения

- Программисты создают программы
- Пользователи запускают программы на исполнение
- Что представляет собой объект исполнения в ОС?

# Объект исполнения

- Объект, представляющий прикладную программу в состоянии выполнения, включает
  - Адресное пространство, выделенное для выполнения программы
  - Код выполняющейся программы
  - Данные выполняющейся программы
  - Стек и указатель на его вершину (stack pointer, SP)
  - Выделенные ресурсы ОС (открытые файлы, установленные сетевые соединения и т.д.)
  - Программный счетчик (instruction pointer, IP), указывающий на следующую выполняемую инструкцию
  - Текущие значения регистров общего назначения
- Объект исполнения представлен двумя понятиями: ***процесс*** и ***поток***

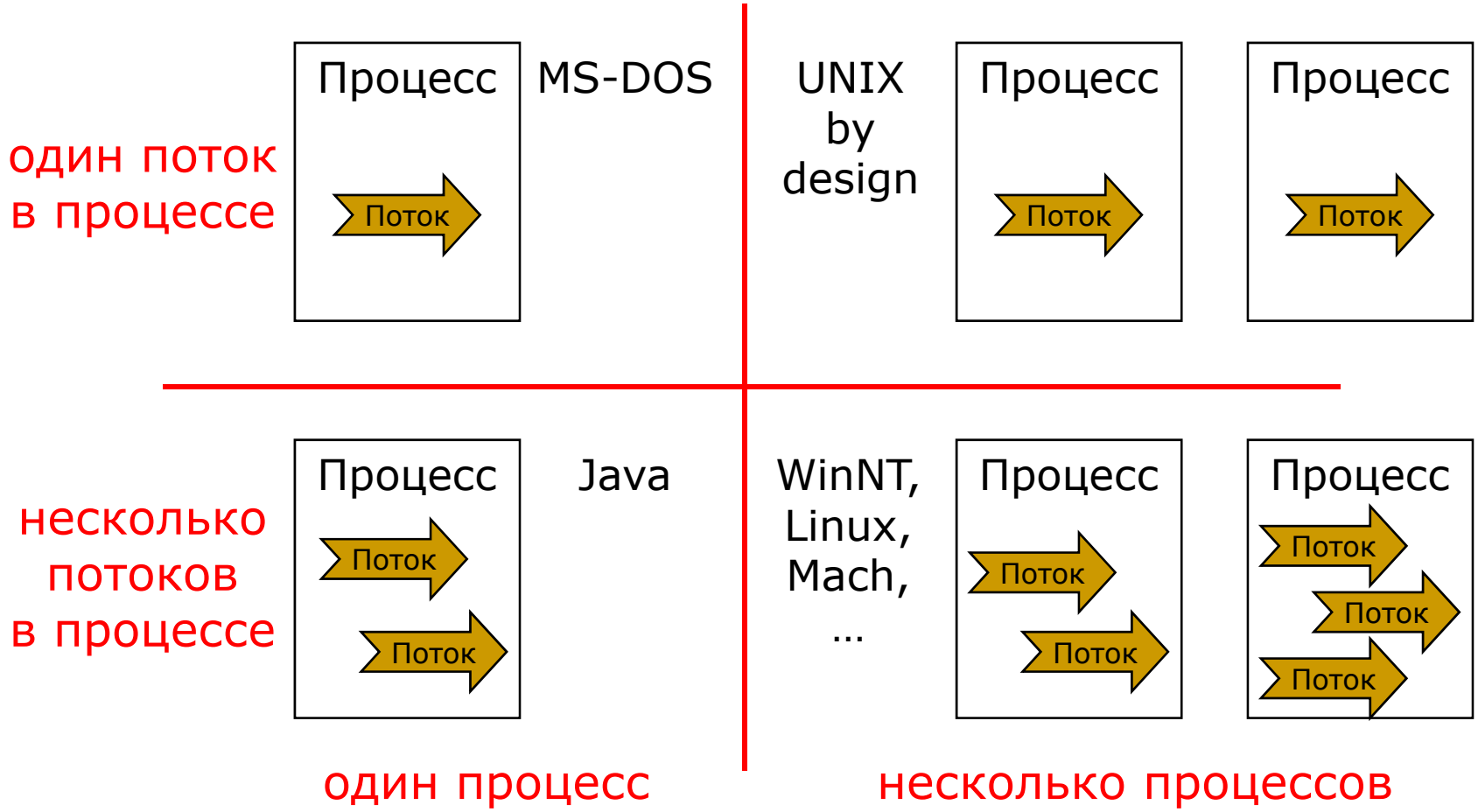
# Процесс

- **Процесс** - абстракция, представляющая программу во время ее выполнения
- Процессу ОС выделяет ресурсы, необходимые для выполнения программы, например:
  - адресное пространство процесса содержит его программный код, данные и стек (или стеки)
  - файлы используются процессом для чтения входных данных и записи выходных
  - устройства ввода-вывода используются в соответствии с их назначением
- Процесс – пассивный объект – владелец ресурсов, контейнер для выполнения ***потоков***

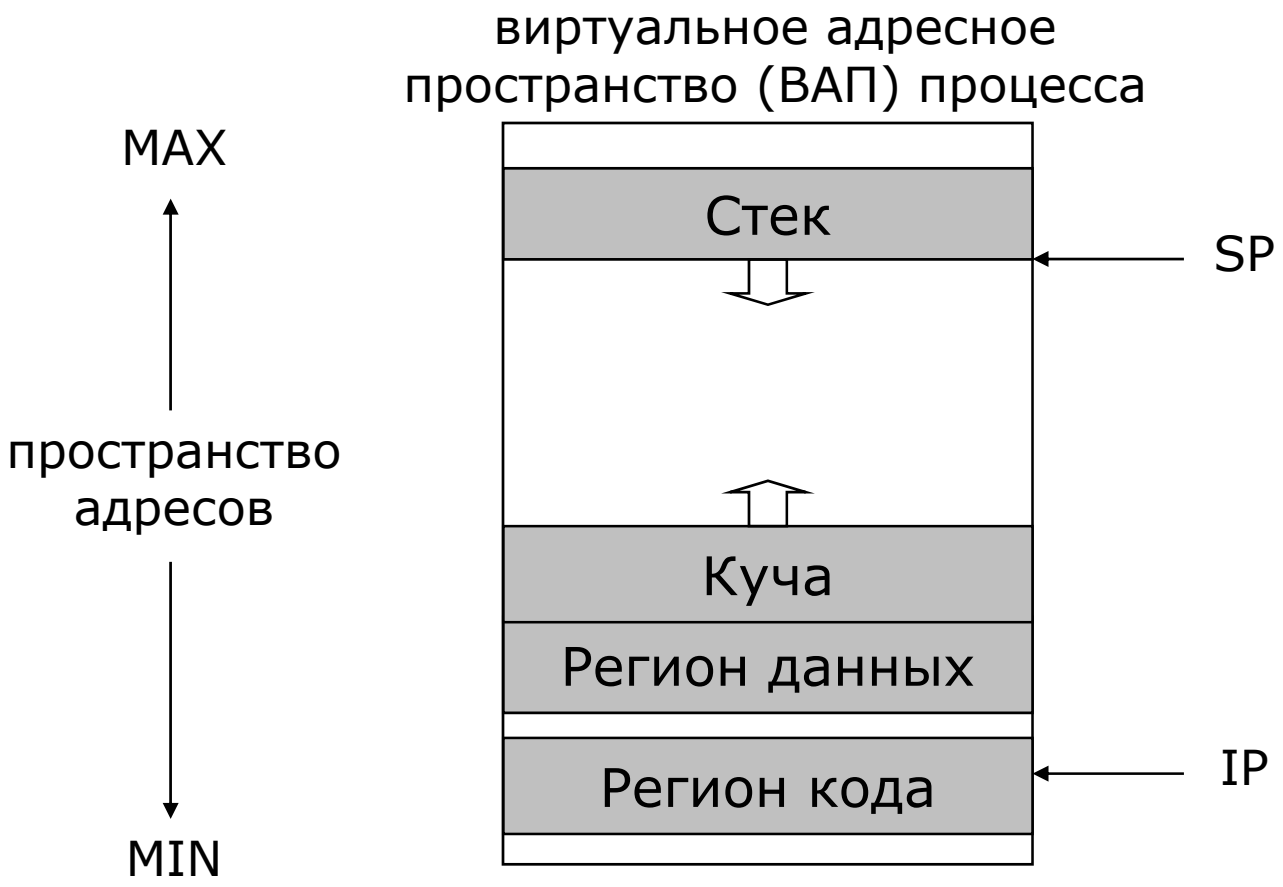
# Поток / нить (thread)

- **Поток** - абстракция, представляющая последовательное выполнение команд программы, развертывающееся во времени
- Процесс может иметь несколько потоков
  - потоки совместно используют
    - глобальные и статические переменные (располагаются в регионе данных)
    - динамически распределяемую память (кучу)
    - системные ресурсы, выделенные процессу
  - каждый поток имеет свои собственные
    - программный счетчик (IP)
    - значения регистров
    - локальные переменные (т.е. свой собственный стек)
- Процесс - совокупность взаимодействующих потоков и выделенных ему ресурсов

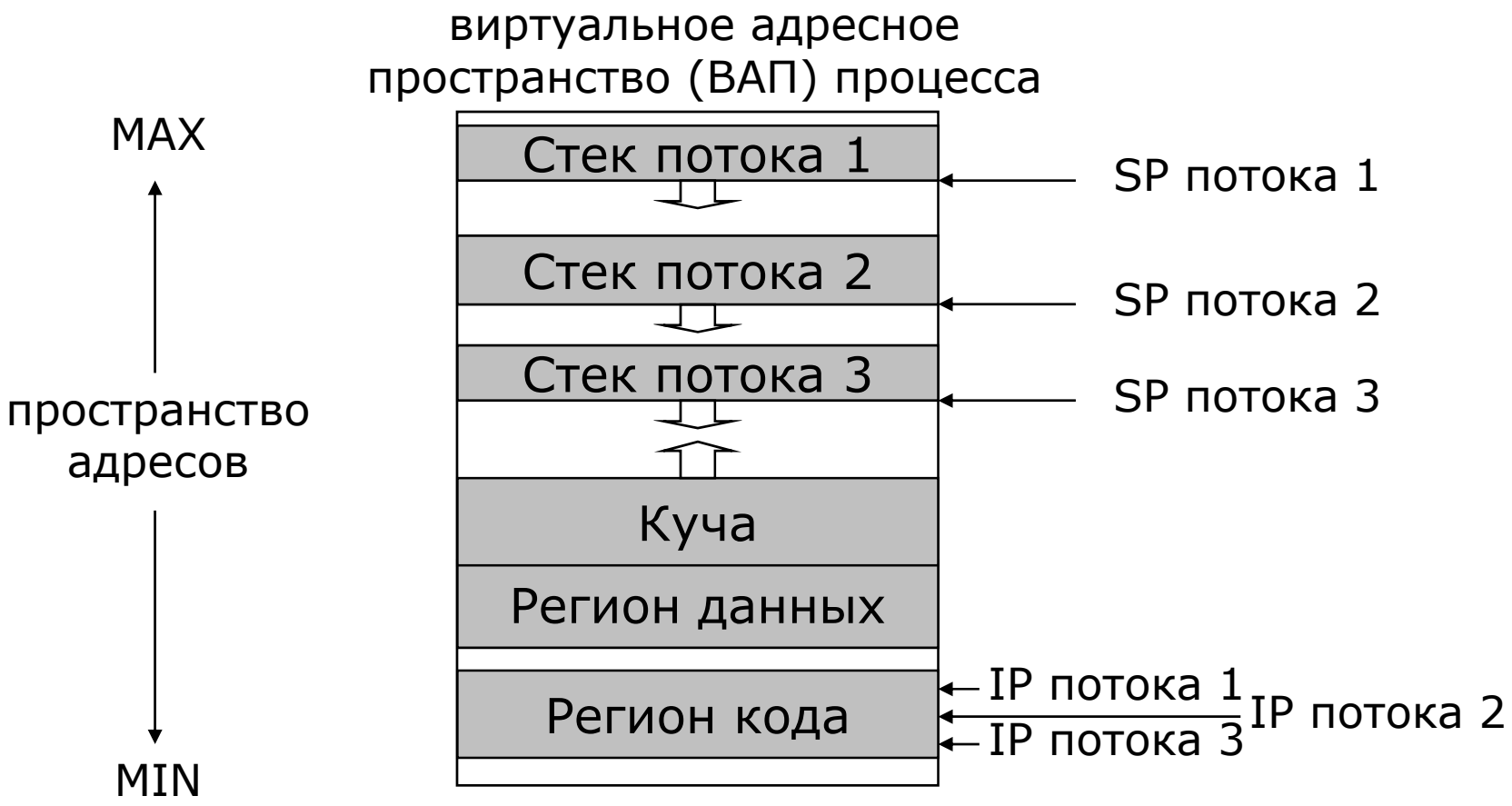
# Процессы и потоки



# Процесс с одним потоком



# Процесс с несколькими потоками



# Зачем нужны процессы с несколькими потоками?

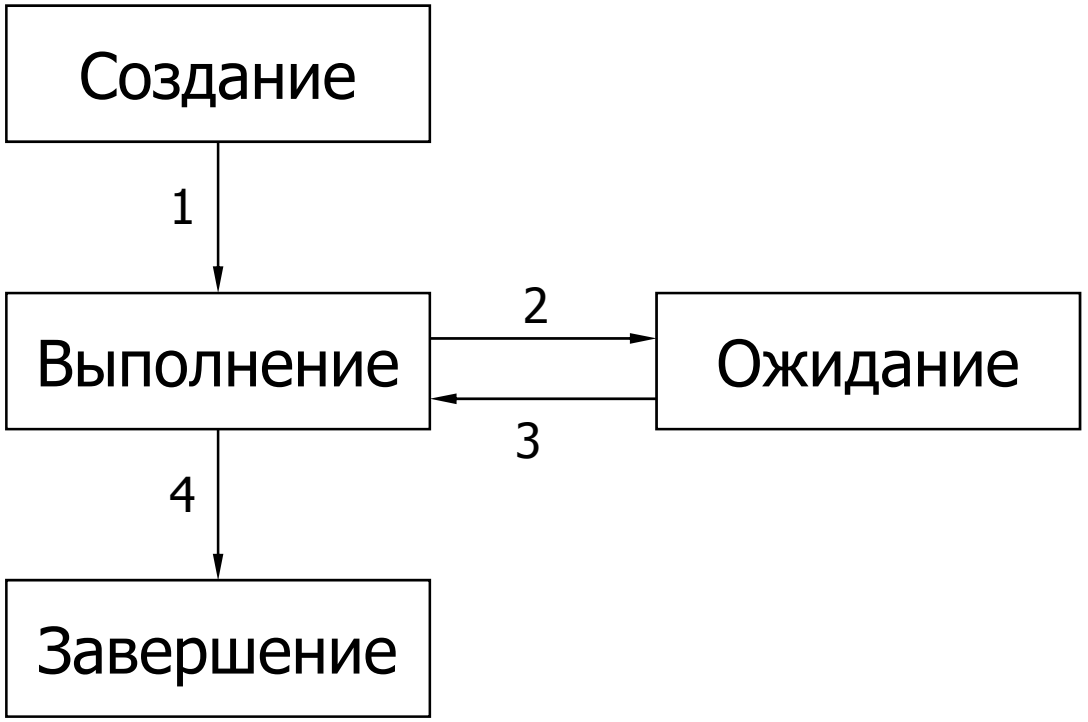
- Для параллельной обработки нескольких однотипных запросов (сетевые сервисы)
- Для разделения исполнительных активностей, параллельно решающих различные задачи
  - обеспечение пользовательского интерфейса
  - математические вычисления
  - фоновая печать
  - …
- Для создания параллельных программ, эффективно использующих аппаратные ресурсы (например, несколько аппаратно имеющихся ЦП)
- Для улучшения структуры программы

# Многопоточная программа vs. Взаимодействующие процессы

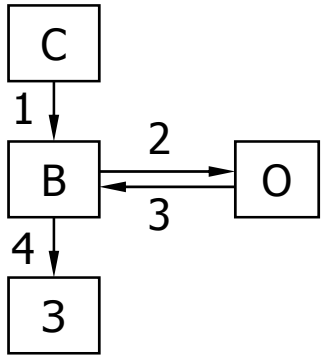
- При использовании потоков
  - экономятся ресурсы (все потоки одного процесса пользуются одним набором ресурсов)
  - экономится время (операции создания/уничтожения потока намного менее затратны, чем операции создания/уничтожения процесса)
  - взаимодействие между потоками одного процесса более удобно и эффективно по причине использования общей памяти

# Состояния потока

# Состояния потока в однозадачной ОС…

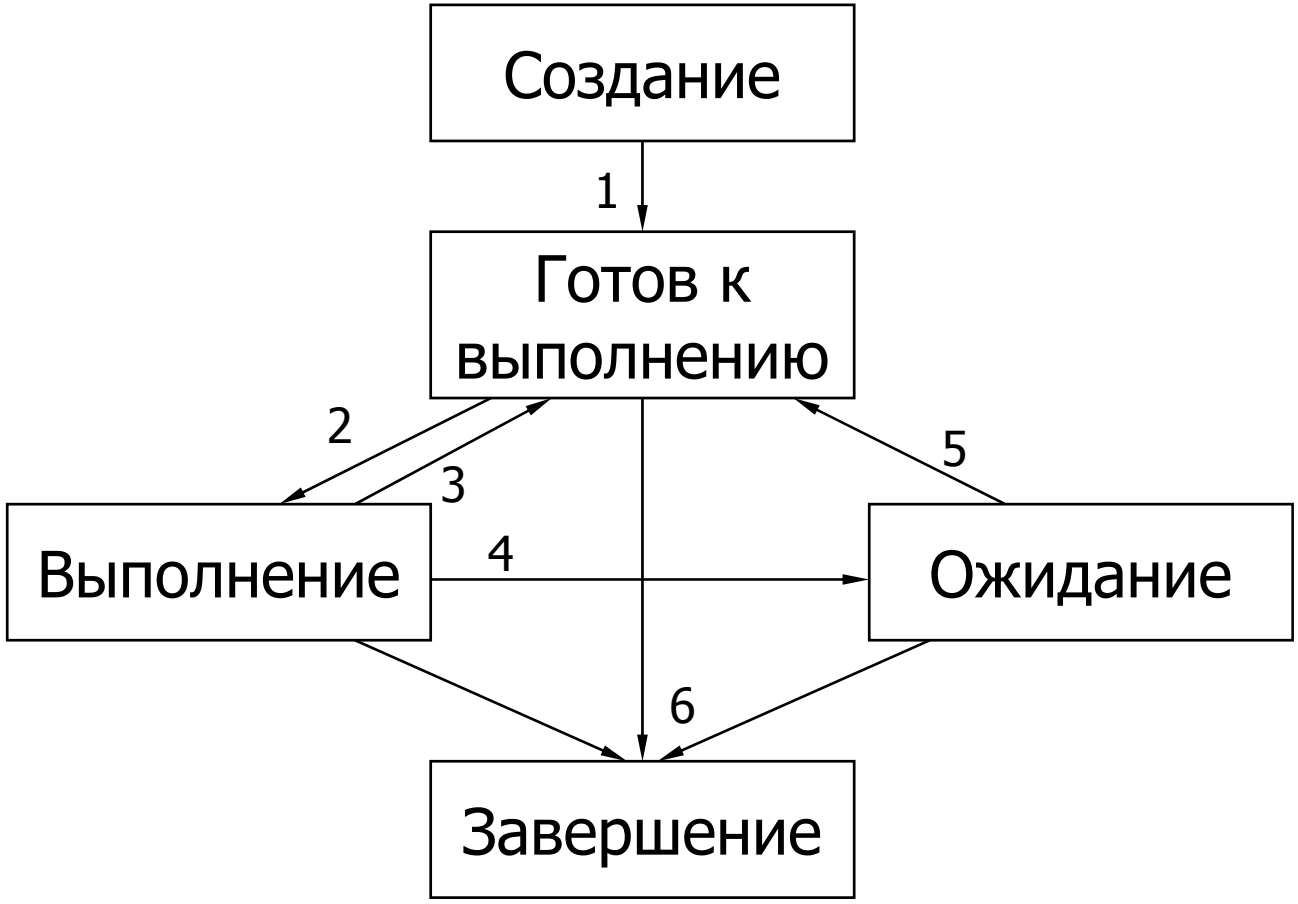


# Состояния потока в однозадачной ОС




- (2) - посредством выполнения системного вызова, подразумевающего ожидание наступления какого-либо события, например, нажатия клавиши или истечения 10 с
- (3) - при наступлении ожидаемого события происходит возврат из системного вызова и возвращение потока в состояние выполнения

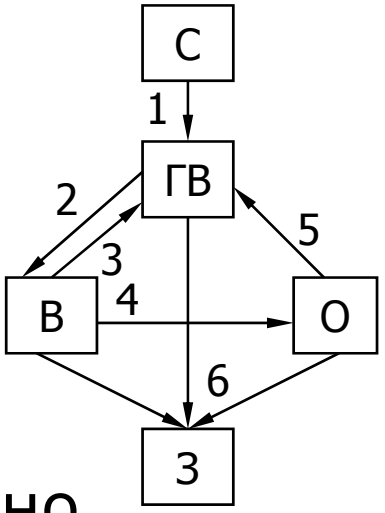
# Состояния потока в многозадачной ОС…



# Состояния потока в многозадачной ОС…

- **Выполнение** – состояние работающего потока – обладающего всеми необходимыми ресурсами, в том числе возможностью использования ЦП
- **Готов к выполнению** – поток обладает всеми необходимыми для выполнения ресурсами за исключением ресурса "время ЦП"
- **Ожидание** (сон, блокировка) – выполнение потока заблокировано до наступления некоторого внешнего события (например, поступления входных данных или освобождения ресурса)

# Состояния потока в многозадачной ОС




- (2,3) - осуществляются ядром операционной системы (**планировщиком**)
- (4) - продолжение работы невозможно
  - для продолжения работы требуется наступление какого-либо события
  - поток затребовал недоступный в данный момент ресурс
  - поток переводится в состояние ожидания ядром операционной системы во время обработки системного вызова
  - поток заблокирован внешним по отношению к нему вызовом
- (5) - производится ядром ОС в момент выполнения условия ожидания

# Управление процессами/потоками

# Управление процессами/потоками

- Структуры, описывающие процессы и потоки в ядре ОС
- Операции над процессами и потоками
- Распределение ресурсов между процессами/потоками
  - время центрального процессора (выделяется потокам)
  - оперативная память (выделяется процессам)
  - другие ресурсы (как правило, выделяются процессам)
- Интерфейс прикладных программ (частные реализации)

# Контекст процесса

- Контекст – множество информации, полностью описывающее состояние объекта (в частности, достаточное для восстановления объекта в случае его удаления)
- Контекст процесса включает
  - Множество информации, используемое операционной системой для управления ресурсом типа «процесс»
  - Адресное пространство процесса
  - Структуру и содержимое пользовательской части адресного пространства процесса
  - Множество ресурсов, используемых процессом или принадлежащих процессу, и их состояния

# Контекст потока

- Контекст потока включает
  - Множество информации, используемое операционной системой для управления ресурсом типа «поток»
  - Множество ресурсов, используемых потоком или принадлежащих потоку и их состояния
  - Аппаратный контекст исполнения потока

# Аппаратный контекст потока

- Состояние процессора с точки зрения предоставляемых потоку прав его использования в конкретной ОС (обычно представляется множеством доступным потоку регистров процессора и их текущими значениями)
- Состояние других устройств в случае, если управление ими осуществляется непосредственно на уровне команд программы, а не через интерфейс доступа к устройствам через выполнение системных вызовов ОС

# Переключение контекста

- Переключение контекста происходит при переходе к исполнению другого потока (возможно, другого процесса)
- При переключении контекста необходимо
  - сохранить контекст вытесняемого потока
  - если поток, выбранный на исполнение, принадлежит другому процессу
    - сохранить контекст процесса – владельца вытесняемого потока
    - загрузить контекст процесса – владельца потока, выбранного на исполнение
  - загрузить контекст потока, выбранного на исполнение
- Для описания процессов и потоков (в том числе для хранения их контекстов) в ядре ОС вводятся специальные структуры – дескрипторы процесса и потока

# Дескриптор процесса

- Идентификатор процесса
- Групповые параметры процесса
- Параметры, используемые в процессе определения приоритета процесса при конкуренции за какой-либо ресурс
- Состояние процесса
- Статистические данные
- Описание адресного пространства процесса
- Контекст ввода-вывода
- Контекст безопасности
- Текущие системные параметры выполнения
- Код завершения процесса

В Linux дескриптор процесса – структура task_struct (include/linux/sched.h), содержит около 100 полей!

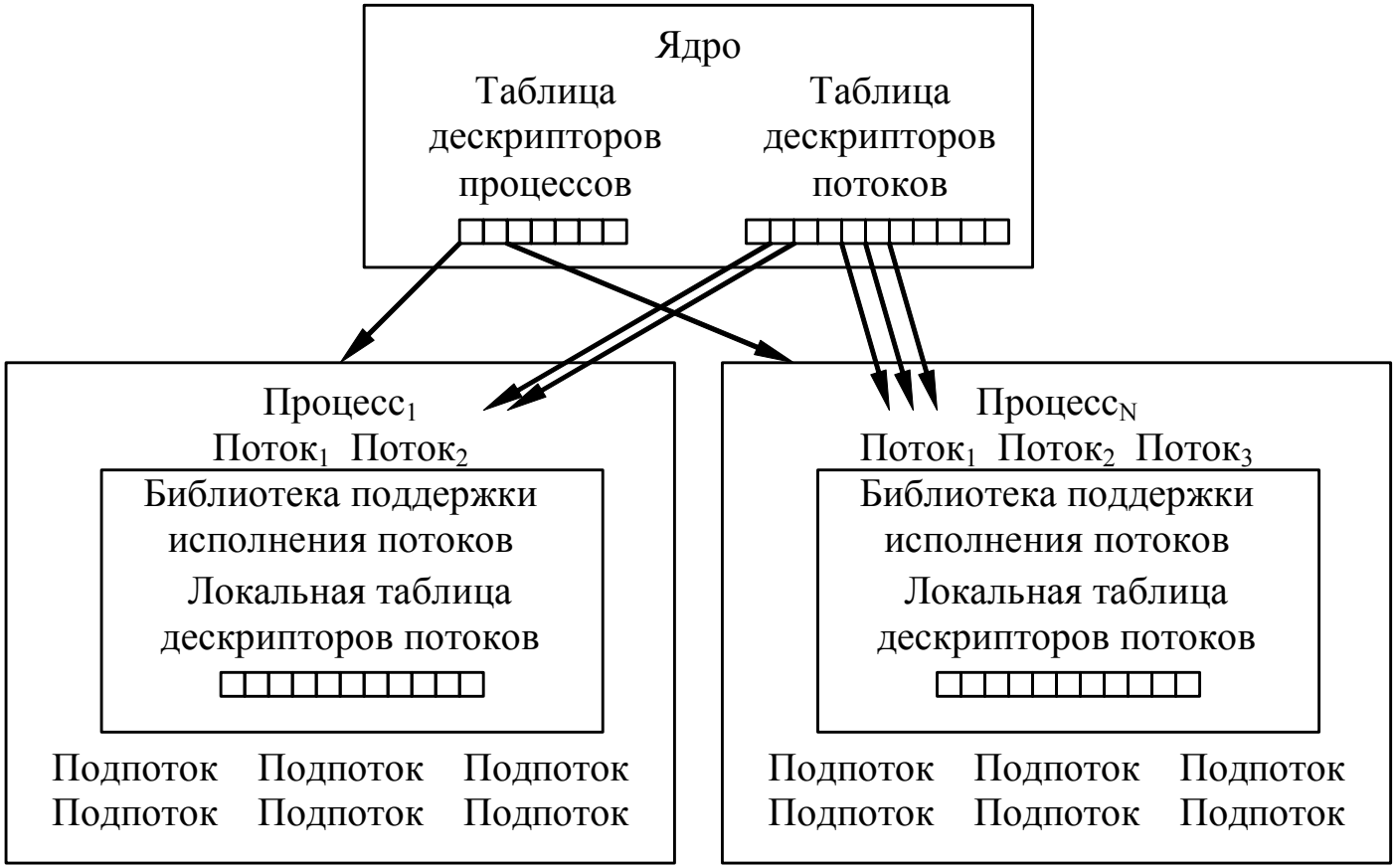
# Дескриптор потока

- Идентификатор потока
- Идентификатор процесса – владельца потока
- Параметры, используемые в процессе определения приоритета потока при конкуренции за какой-либо ресурс
- Статистические данные потока
- Аппаратный контекст выполнения потока (программный счетчик, стек и указатель на его вершину, значения регистров)
- Код завершения потока

# Кто управляет процессами и потоками?

- За управление процессами всегда отвечает ядро ОС
- Ядро всегда предоставляет каждому процессу один поток, но не всегда поддерживает многопоточность
  - потоки, непосредственно управляемые ядром ОС, называются потоками ядра
- Можно реализовать многопоточность в библиотеке пользовательского уровня!

# Потоки ядра vs. пользовательские потоки…



# Потоки ядра vs. пользовательские потоки

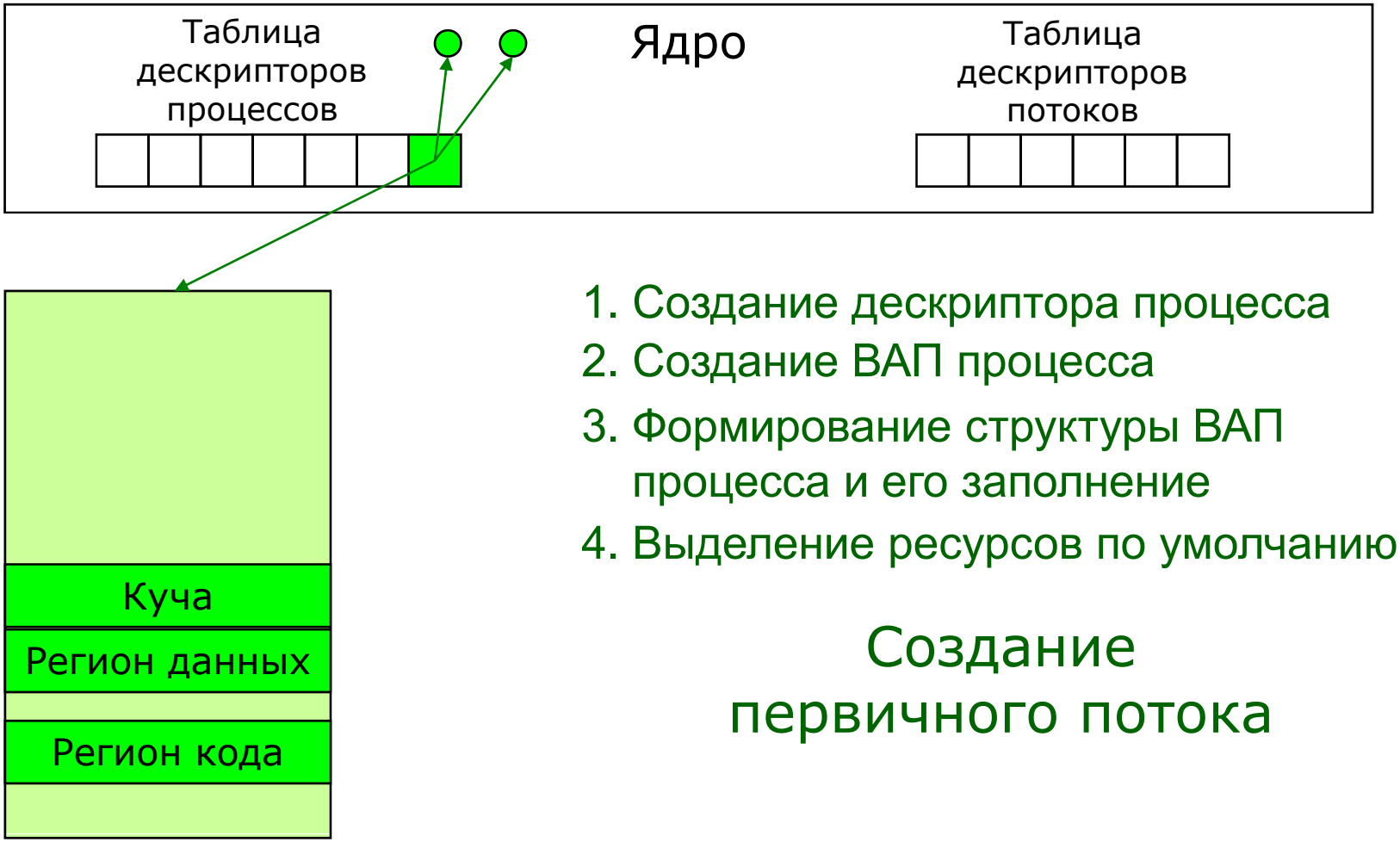
- При использовании пользовательских потоков
  - (+) Операции над потоками выполняются без выполнения системных вызовов (в 10-100 раз быстрее, чем при использовании потоков ядра)
  - (+) Можно реализовать собственный алгоритм планирования
  - (-) Ядро ничего не знает о потоках пользовательского уровня и распределяет время ЦП независимо от их количества в процессе
  - (-) Если будет выполнен блокирующий системный вызов (например, вызвана синхронная операция ввода-вывода), в состояние ожидания переводится поток ядра, использовавшийся для обеспечения выполнения нескольких (или всех) пользовательских потоков. Соответственно, выполнение всех этих пользовательских потоков будет заблокировано.

# Операции над процессами и потоками

# Создание процесса

- Создать дескриптор процесса и поместить его в таблицу процессов
- Проинициализировать значения полей общего назначения дескриптора процесса
- Создать виртуальное адресное пространство (ВАП) процесса и сформировать его структуру
- Заполнить необходимыми данными ВАП процесса (разместить в нем код, данные и т.д.)
- Выделить процессу ресурсы, которые он может использовать сразу после создания
- Оповестить подсистемы, принимающие участие в управлении процессами, о создании нового процесса
- Создать первичный поток процесса

# Создание процесса



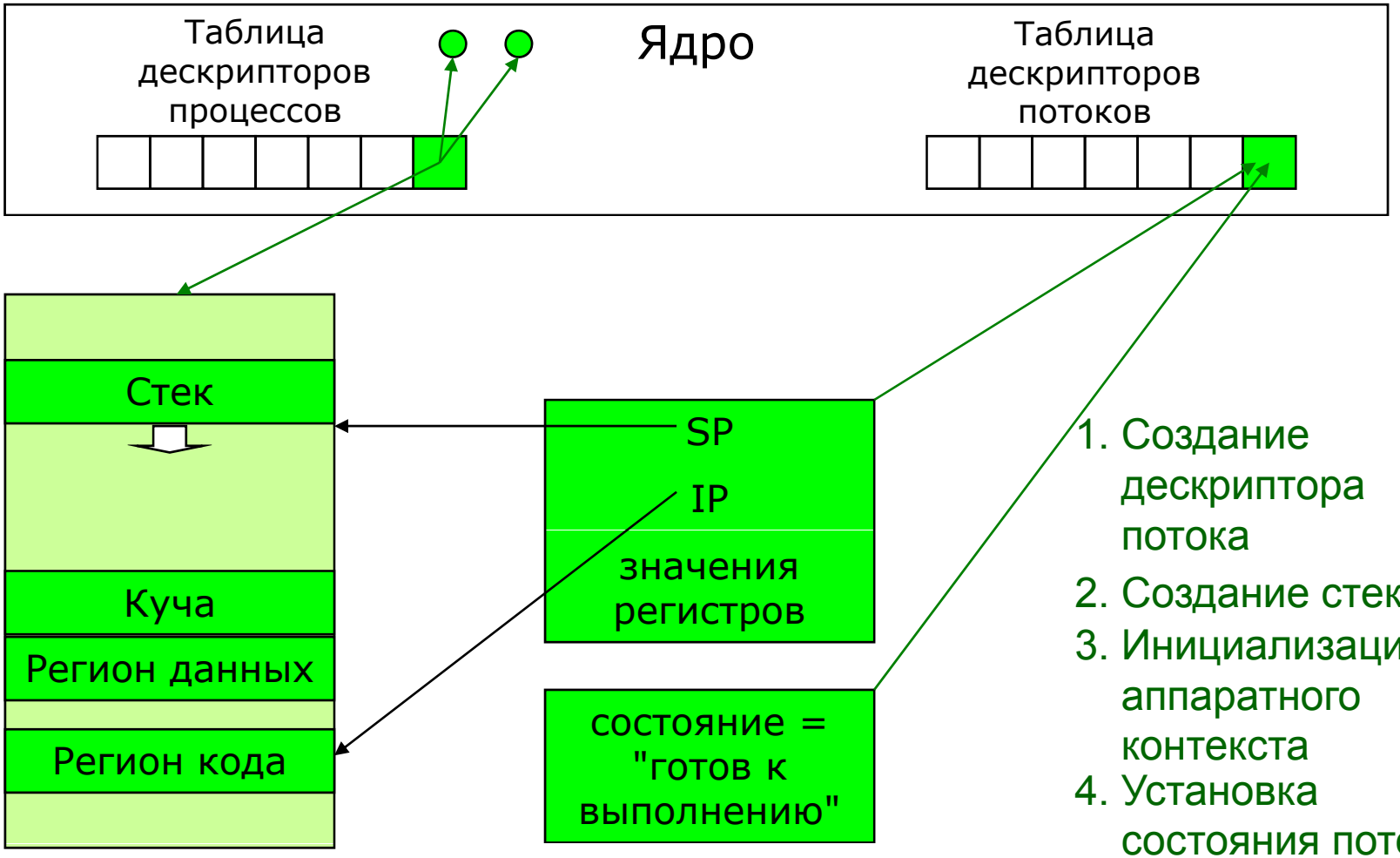

1. Создание дескриптора процесса
2. Создание ВАП процесса
3. Формирование структуры ВАП процесса и его заполнение
4. Выделение ресурсов по умолчанию

Создание первичного потока

# Создание потока

- Создать дескриптор потока и поместить его в таблицу потоков
- Проинициализировать значения полей общего назначения дескриптора потока
- Создать области данных, необходимые для функционирования потока в данной аппаратной архитектуре
- Инициализировать поле дескриптора «аппаратный контекст выполнения потока»
- Оповестить подсистемы, принимающие участие в управлении потоками, о создании нового потока
- Перевести поток в состояние «готов к выполнению»

# Создание потока




1. Создание дескриптора потока
2. Создание стека
3. Инициализация аппаратного контекста
4. Установка состояния потока

# Завершение потока

- Сохранить статистические данные потока и код возврата в его дескрипторе
- Перевести все ресурсы, принадлежащие потоку, в непротиворечивое и стабильное состояние
- Освободить все ресурсы, принадлежавшие потоку или использовавшиеся потоком
- Оповестить подсистемы, принимающие участие в управлении потоками, о завершении потока
- Установить состояние потока в значение «завершен»
- Если данный поток является последним активным потоком в процессе – завершить процесс

После выполнения всех действий остается дескриптор потока, содержащий его код возврата и статистические данные; момент уничтожения дескриптора зависит от реализации

# Завершение процесса

- Завершить выполнение всех потоков процесса
- Сохранить статистические данные процесса и код возврата в его дескрипторе
- Перевести все ресурсы, принадлежащие процессу, в непротиворечивое и стабильное состояние
- Освободить все ресурсы, принадлежавшие процессу или использовавшиеся процессом
- Освободить ВАП и уничтожить его
- Оповестить подсистемы, принимающие участие в управлении процессами, о завершении процесса
- Установить состояние процесса в значение «завершен»

После выполнения всех действий остается дескриптор процесса, содержащий его код возврата и статистические данные; момент уничтожения дескриптора зависит от реализации

# Создание/завершение процесса – Win32

```
BOOL CreateProcess(
    LPCTSTR lpszImageName,
    LPCTSTR lpszCommandLine,
    LPSECURITY_ATTRIBUTES lpsaProcess,
    LPSECURITY_ATTRIBUTES lpsaThread,
    BOOL fInheritHandles,
    DWORD fdwCreate,
    LPVOID lpvEnvironment,
    LPTSTR lpszCurDir,
    LPSTARTUPINFO lpsiStartInfo,
    LPPROCESS_INFORMATION lppiProcInfo);
```

```
VOID ExitProcess(UINT fuExitCode);
```

- Описание функций можно найти в MSDN и в работе [2]

# Создание/завершение процесса – UNIX…

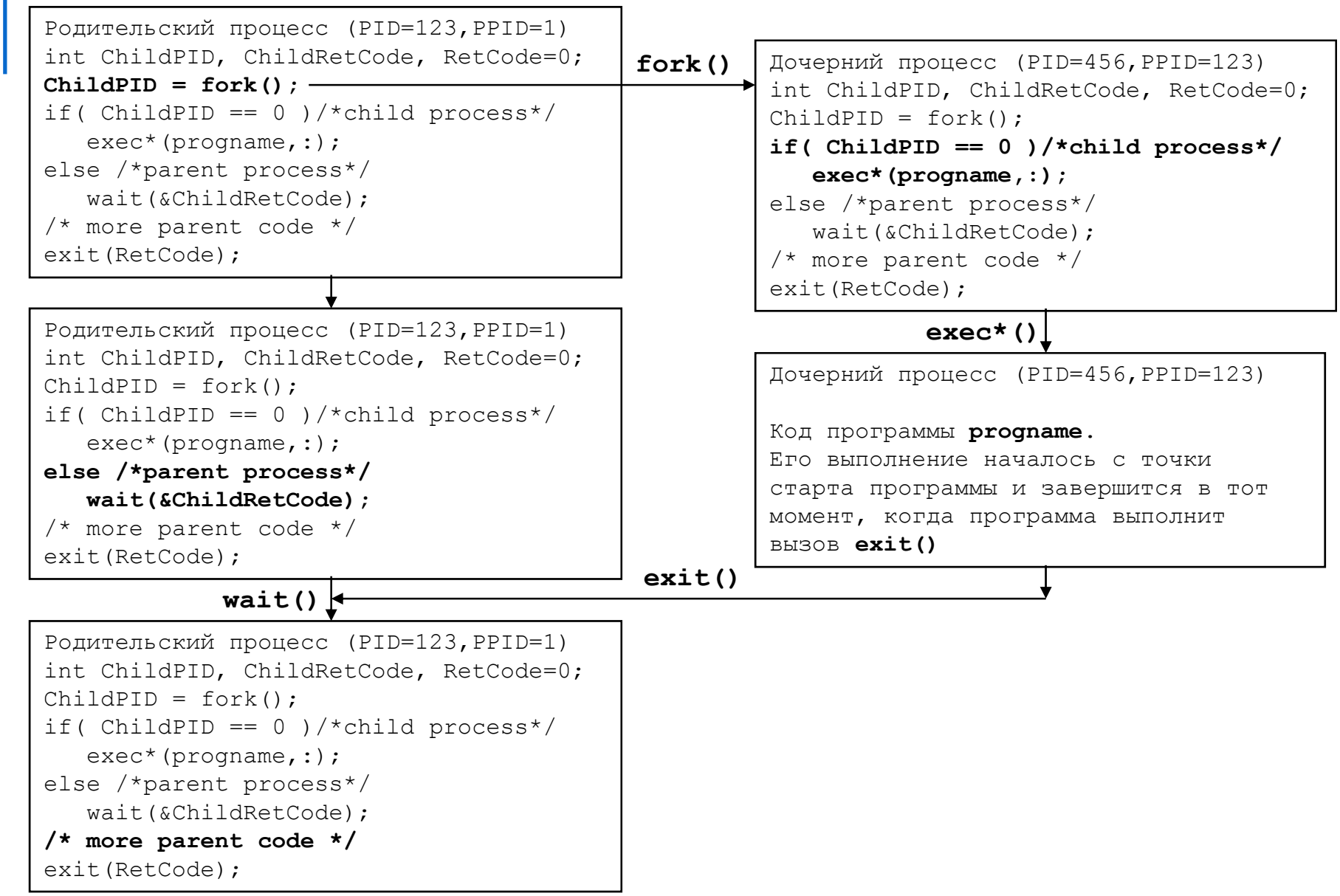
- Создание копии процесса
  int fork(void);
- Использование ресурсов процесса для выполнения указанной программы
  int exec*(char *path, char* argv[], char **env);
- Завершение процесса
  void exit(int status);
- Ожидание завершения процесса-потомка
  int wait(int *status);

- Описание функций можно найти в документации UNIX (man или info) и в работе [3]

# Создание/завершение процесса – UNIX

```
/* Программа, создающая процесс-потомок и
запускающая в потомке другую программу */

int ChildPID, ChildRetCode, RetCode=0;
ChildPID = fork();
if( ChildPID == 0 )/*child process*/
  exec*(progname,…);
else                          /*parent process*/
  wait(&ChildRetCode);
/* more parent code */
exit(RetCode);
```


Родительский процесс (PID=123,PPID=1)
int ChildPID, ChildRetCode, RetCode=0;
ChildPID = fork();
if( ChildPID == 0 )/*child process*/
exec*(progname,:);
else /*parent process*/
wait(&ChildRetCode);
/* more parent code */
exit(RetCode);
fork()
Дочерний процесс (PID=456,PPID=123)
int ChildPID, ChildRetCode, RetCode=0;
ChildPID = fork();
if( ChildPID == 0 )/*child process*/
exec*(progname,:);
else /*parent process*/
wait(&ChildRetCode);
/* more parent code */
exit(RetCode);
exec*()
Дочерний процесс (PID=456,PPID=123)
Код программы progname.
Его выполнение началось с точки старта программы и завершится в тот момент, когда программа выполнит вызов exit()
exit()
wait()
Родительский процесс (PID=123,PPID=1)
int ChildPID, ChildRetCode, RetCode=0;
ChildPID = fork();
if( ChildPID == 0 )/*child process*/
exec*(progname,:);
else /*parent process*/
wait(&ChildRetCode);
/* more parent code */
exit(RetCode);
Родительский процесс (PID=123,PPID=1)
int ChildPID, ChildRetCode, RetCode=0;
ChildPID = fork();
if( ChildPID == 0 )/*child process*/
exec*(progname,:);
else /*parent process*/
wait(&ChildRetCode);
/* more parent code */
exit(RetCode);

# Заключение

- **Процессы** и **потоки** – объекты, представляющие программы во время их выполнения
- Процесс – пассивный объект, владелец ресурсов
- Поток – активный объект
- С точки зрения ОС, процессы и потоки – специфичные типы ресурсов, требующие специального управления

# Вопросы для обсуждения

- Какая схема запуска новых программ кажется вам предпочтительней: в стиле UNIX или в стиле Windows?
- Приведенный в лекции список действий при создании и завершения процессов/потоков содержит только часть шагов. Какие действия вы бы добавили?

# Задание для самостоятельной работы

- Напишите программу, циклически выполняющую следующие действия:
  - ввод командной строки
  - выделение из командной строки имени программы и аргументов
  - запуск программы  и ожидание ее завершения
  - программа завершается, если введена командная строка "exit"

# Литература


1. Таненбаум Э. Современные операционные системы. 2-е изд. – СПб.: Питер, 2002.
2. Рихтер Дж. Windows для профессионалов (Создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows). 4-е изд. – М.: Русская Редакция; пер. с англ. – СПб.: Питер, 2001.
3. Робачевский А.М. Операционная система UNIX. – СПб.: BHV - Санкт-Петербург, 1998.

# Тема следующей лекции – Планирование

- Понятие планирования
- Критерии оценки алгоритмов планирования
- Некоторые алгоритмы планирования