СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

ЛЕКЦИЯ № 6

ПРЕПОДАВАТЕЛЬ: ХУСТОЧКА А.В.

РЕСУРСЫ ОПЕРАЦИОННОЙ СИСТЕМЫ

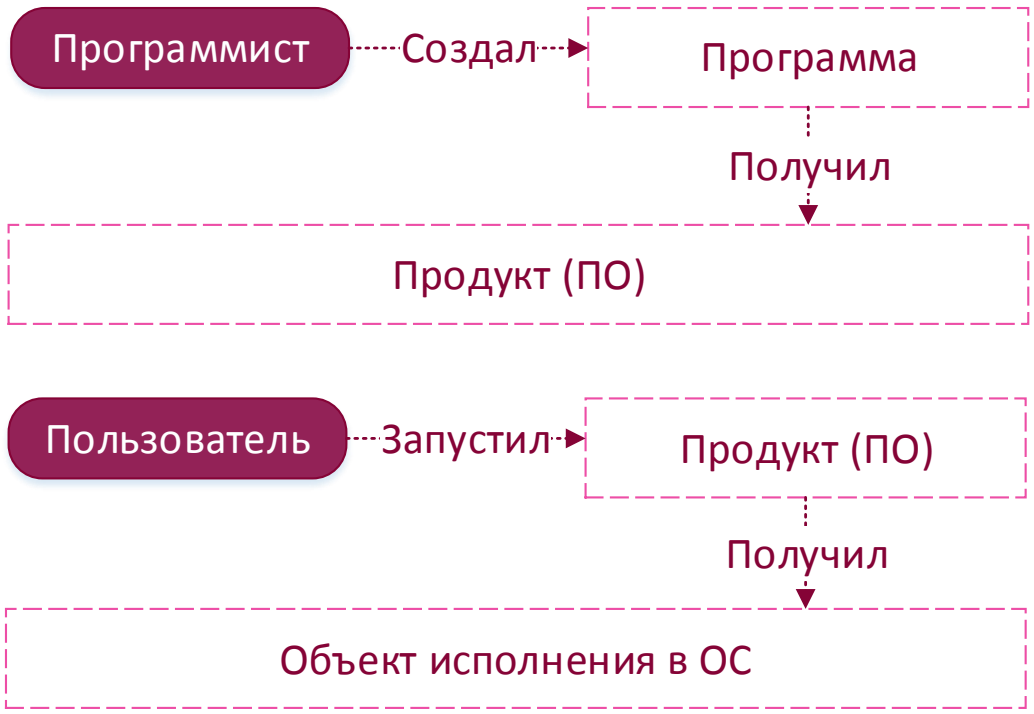
Ресурс - средство системы обработки данных, которое может быть выделено процессу обработки данных на определенный интервал времени.

Основные ресурсы:

- Процессорное время
- Память
- Программные модули

КЛАССИФИКАЦИИ РЕСУРСОВ

ОБЪЕКТ ИСПОЛНЕНИЯ В ОПЕРАЦИОННОЙ СИСТЕМЕ



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 на следующую выполняемую инструкцию
- Текущие значения регистров общего назначения

Объект исполнения представлен двумя понятиями: процесс и поток

ПРОЦЕСС

- Процесс – абстракция, представляющая программу во время её выполнения.
- Операционная система выделяет процессу необходимые ресурсы.
- Процесс – это пассивный объект – владелец ресурсов, некий контейнер для выполнения потоков.
- Процесс может иметь несколько потоков.
- Процесс – совокупность взаимодействующих потоков и выделенных ему ресурсов

ПОТОК

Поток – абстракция представляющая последовательное выполнение команд программы, развертывающееся во времени.

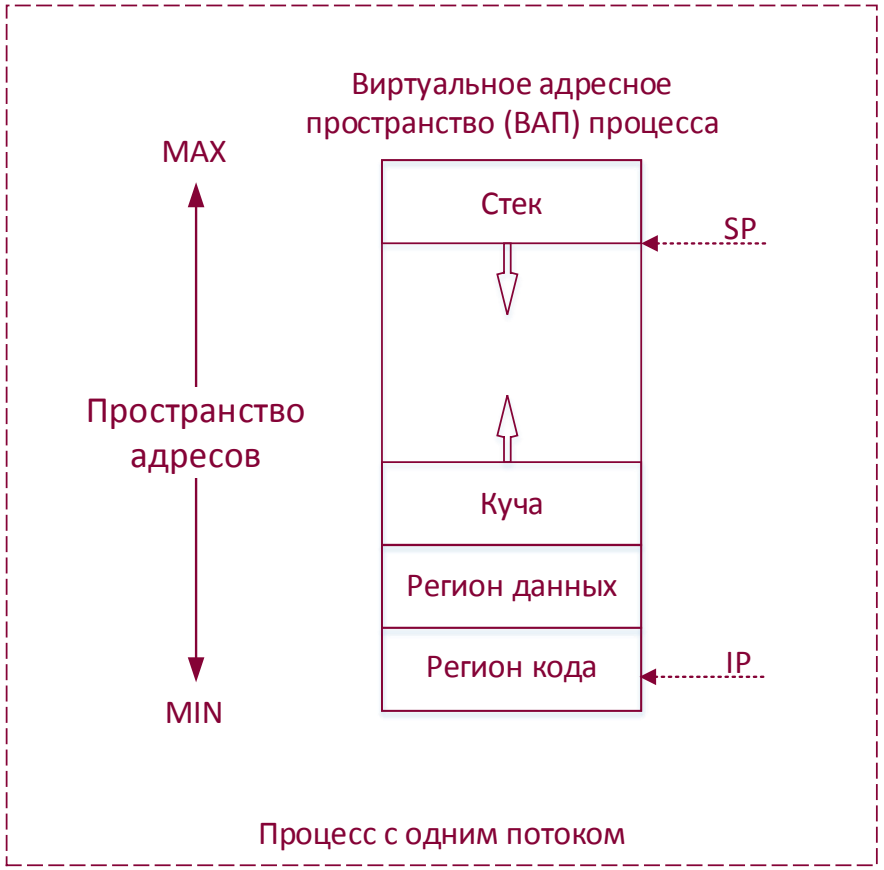
Потоки совместно используют:

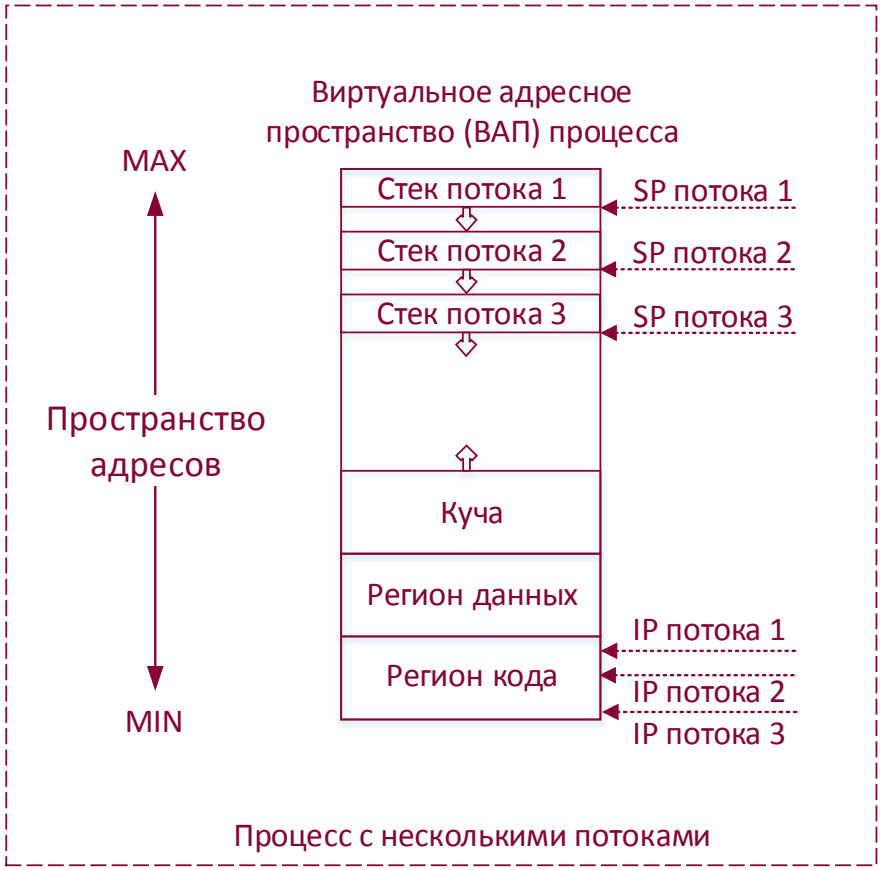
- глобальные и статические переменные (располагаются в регионе данных)
- динамически распределяемую память (кучу)
- системные ресурсы, выделенные процессу

Каждый поток имеет свои собственные:

- программный счетчик (Instruction Point, IP)
- значения регистров
- локальные переменные (т.е. свой собственный стек)

ВИРТУАЛЬНОЕ АДРЕСНОЕ ПРОСТРАНСТВО





ГДЕ ИСПОЛЬЗУЮТСЯ ПРОЦЕССЫ С НЕСКОЛЬКИМИ ПОТОКАМИ?

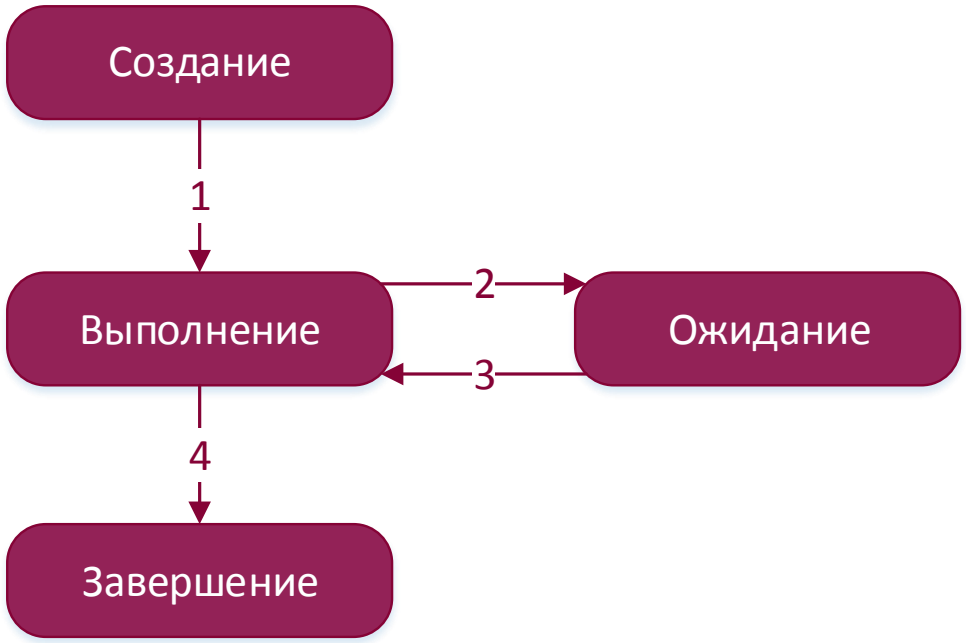
- Параллельная обработка однотипных запросов (сетевые сервисы)
- Разделение исполнительных активностей, параллельно решающих различные задачи:
  - Обеспечение UI
  - Математические вычисления
  - Фоновые задачи (например, печать документов)
- Реализация параллельных программ, эффективно использующих аппаратные ресурсы
- При улучшении 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
- Используется общая память: по этому взаимодействие между потоками одного процесса более удобно и эффективно

СОСТОЯНИЯ ПОТОКА В ОДНОЗАДАЧНОЙ ОС



- (2) – посредством выполнения системного вызова, подразумевающего ожидания наступления какого-либо события (таймер/нажатие клавиши)
- (3) – при наступлении ожидаемого события происходит возврат из системного вызова и возвращение потока в состояние выполнения

СОСТОЯНИЯ ПОТОКА В МНОГОЗАДАЧНОЙ ОС

- Выполнение – состояние работающего потока – обладающего всеми необходимыми ресурсами, в том числе возможностью использования центрального процессора
- Готов к выполнению – поток обладает всеми ресурсами для выполнения ресурсами, за исключением ресурса «время центрального процессора»
- Ожидание (сон/блокировка) – выполнение потока заблокировано до наступления некоторого внешнего события (поступления входных данных/ освобождение ресурса)

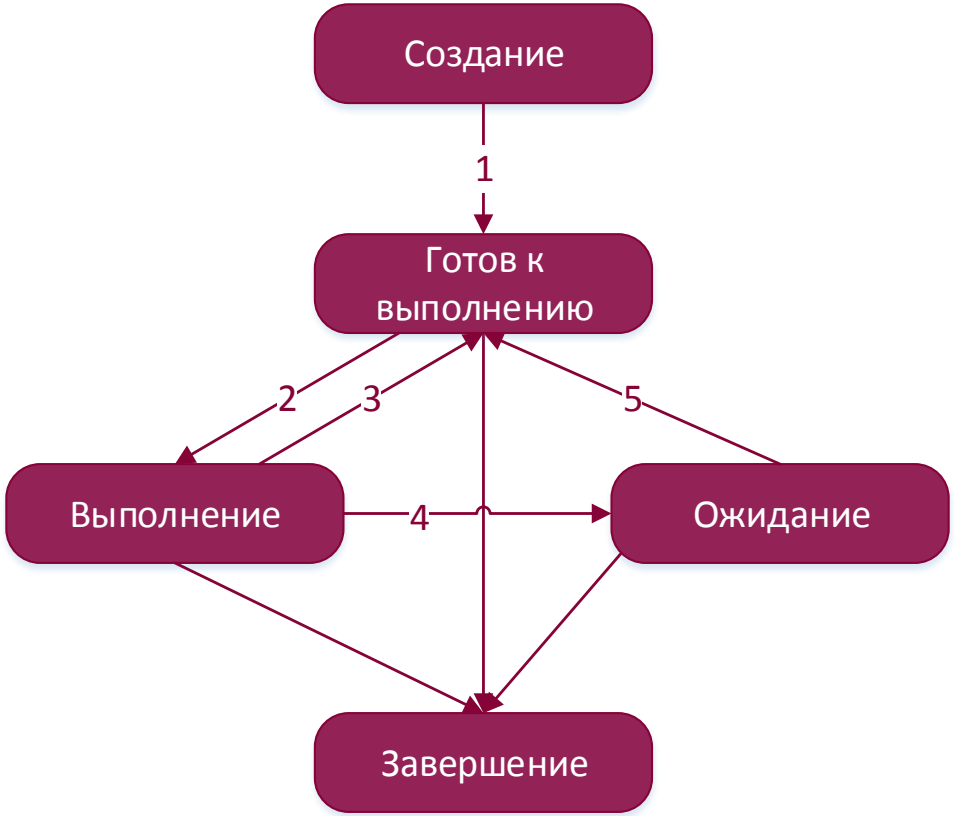
СОСТОЯНИЯ ПОТОКА В МНОГОЗАДАЧНОЙ ОС

(2,3) – осуществляется ядром ОС (планировщик)

(4) – продолжение работы невозможно:

- Потому что требуется наступление какого-либо события
- Поток затребовал недоступный в данный момент ресурс
- Поток переводится в состояние ожидания ядром ОС во время обработки системного вызова
- Поток заблокирован внешним, по отношению к нему, вызовом

(5) – производится ядром ОС в момент выполнения условия ожидания



КОНТЕКСТ ПРОЦЕССА

Контекст – множество информации, полностью описывающее состояние объекта (в частности, достаточное для восстановления объекта в случае его удаления)

Контекст включает:

- Множество информации, используемой 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, а также состояния этих ресурсов

КОНТЕКСТ ПОТОКА

Контекст потока включает:

- Множество информации, используемой операционной системой для управления ресурсом типа «поток»
- Множество ресурсов, используемых потоком или принадлежащих потоку, а также состояния этих ресурсов
- Аппаратный контекст исполнения потока

АППАРАТНЫЙ КОНТЕКСТ ПОТОКА

- Состояние процессора с точки зрения предоставляемых потоку прав его использования в конкретной ОС (как правило, предоставляется 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 в 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, то:
  - Сохранить контекст процесса – владельца вытесняемого потока
  - Загрузить контекст процесса – владельца 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

Примечание: в Linux дескриптор процесса описывается с помощью структуры task_struck (sched.h), которая 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 счётчик, стек и указатель на его вершину, значения регистров)
- Код завершения потока

КТО УПРАВЛЯЕТ ПРОЦЕССАМИ И ПОТОКАМИ?

- За управления процессами всегда отвечает ядро операционной системы
- Ядро всегда предоставляет каждому процессу один потока, но не всегда поддерживает многопоточность
  - Потоки, непосредственно управляемые ядром ОС, называются потоками ядра
- Многопоточноть можно реализовать в библиотеке в пространстве пользователя

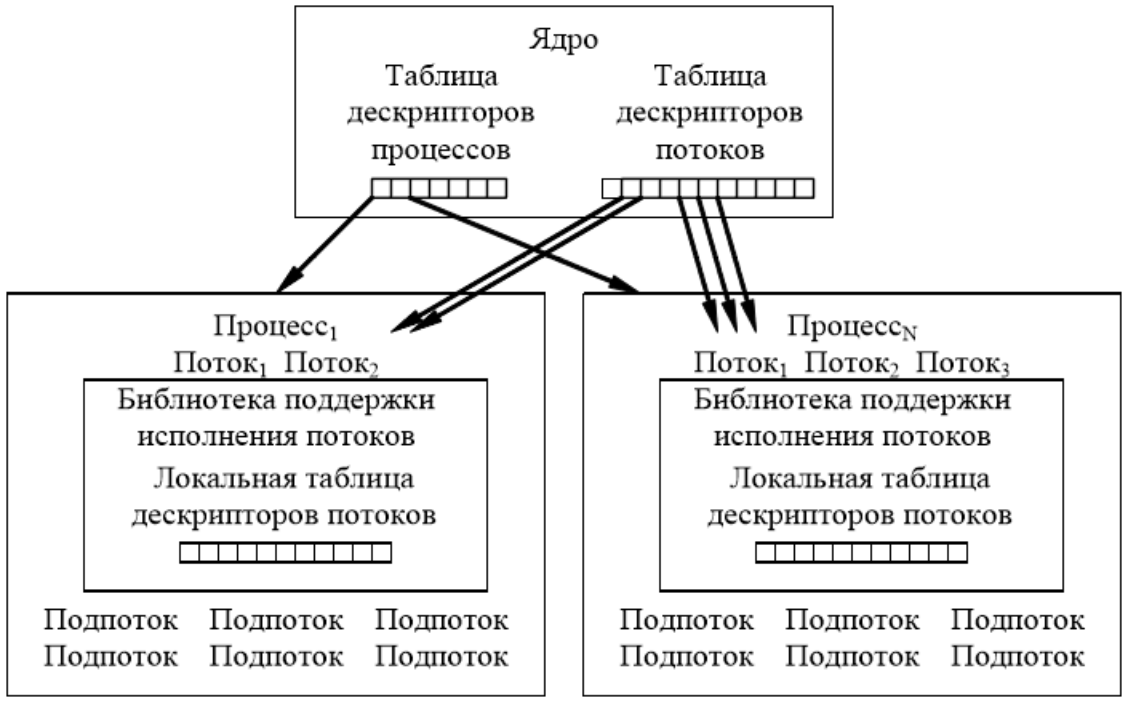
ПОТОК ЯДРА VS. ПОЛЬЗОВАТЕЛЬСКИЙ ПОТОК

Плюсы 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 потоков

Минусы при использовании пользовательских потоков:

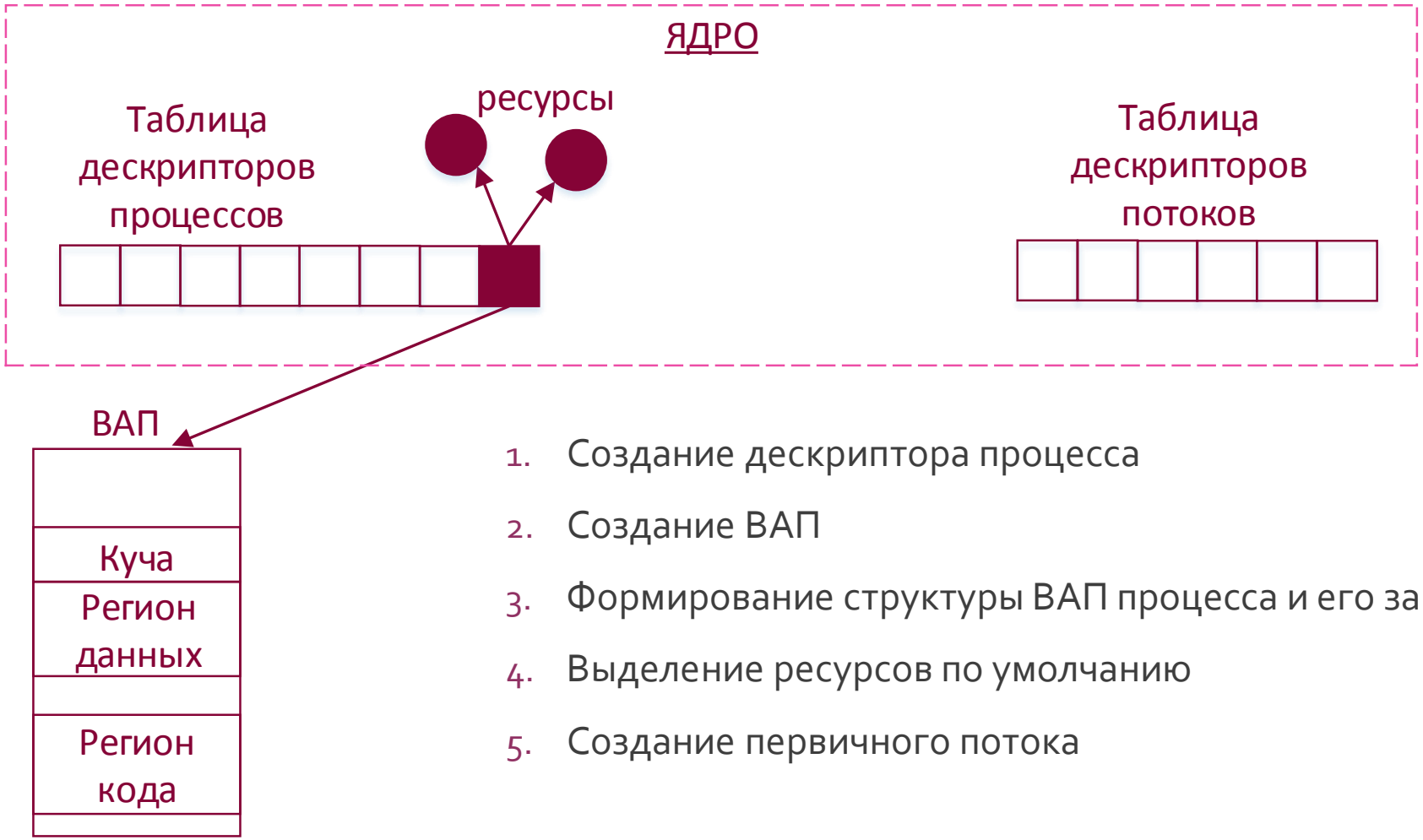
- Ядро ничего не знает о потоках пользовательского уровня и распределяет время центрального процессора независимо от их количества в процессе
- Если будет выполнен блокирующий системный вызов (например, вызвана синхронная операция ввода/вывода), в состояние ожидания переводится поток ядра, использовавшийся для обеспечения нескольких (или всех) пользовательских потоков. Соответственно, выполнение всех этих потоков будет заблокировано.



СОЗДАНИЕ ПРОЦЕССА

- Создать таблицу дескрипторов 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 нём код, данные итд)
- Выделить процессу ресурсы, которые он может использовать сразу после создания
- Оповестить подсистемы, принимающие участие в управление процессами, о создании нового процесса
- Создать первичный поток процесса

СОЗДАНИЕ ПРОЦЕССА

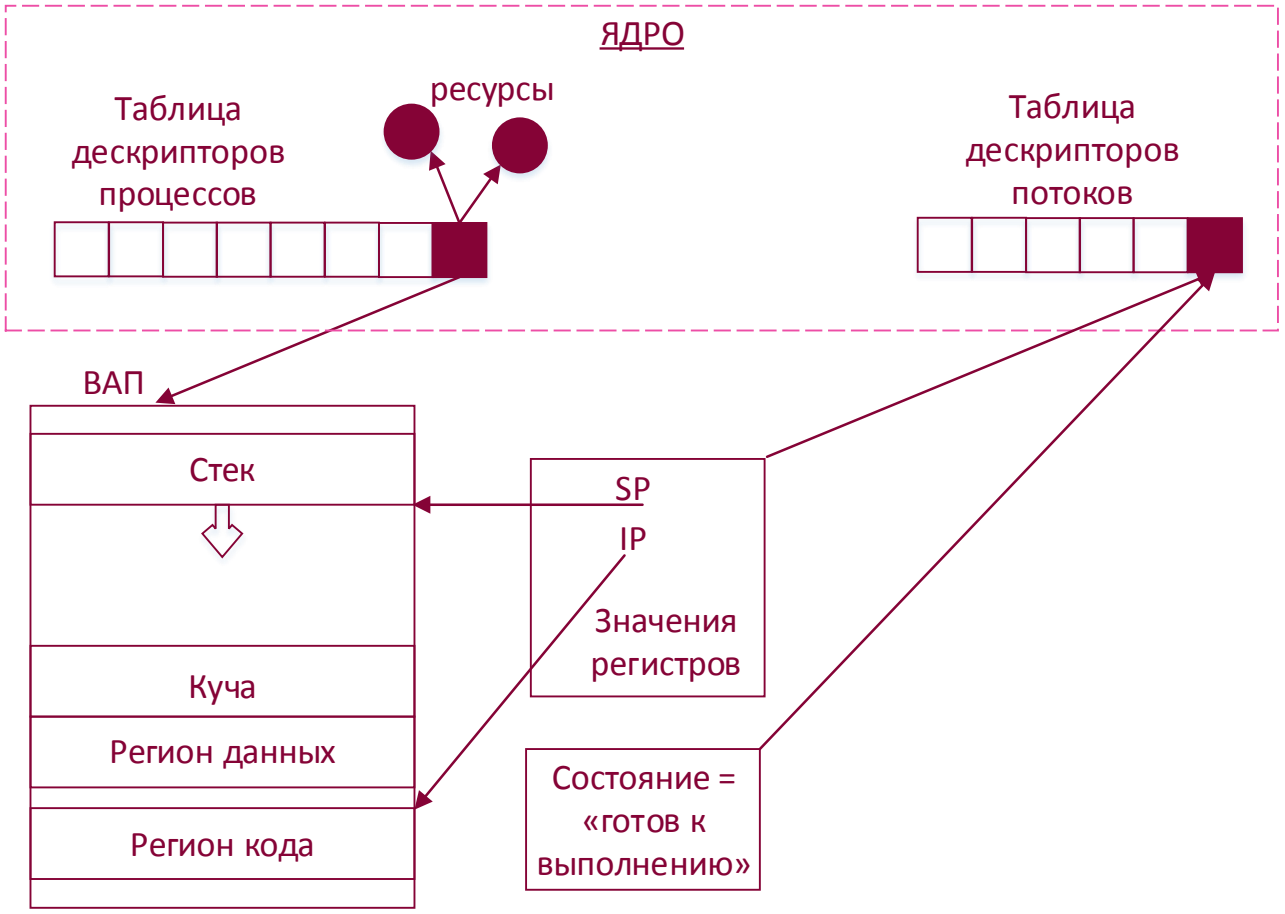


1. Создание дескриптора процесса
2. Создание ВАП
3. Формирование структуры ВАП процесса и его заполнение
4. Выделение ресурсов по умолчанию
5. Создание первичного потока

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 аппаратное архитектуре
- Инициализировать поле дескриптора «аппаратный контекст выполнения потока»
- Оповестить подсистемы, принимающие участие в управлении потоками, о создании нового потока
- Перевести поток в состоянии «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, принадлежащие потоку или использовавшиеся потоком
- Оповестить подсистемы, принимающие участие в управлении потоками, о завершении потока
- Установить состояние потока в значение «завершён»
- Если данный поток является последним активным потоком в процессе – завершить процесс

После выполнения всех действий остаётся 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 дескрипторе
- Перевести все ресурсы, принадлежащие процессу, в непротиворечивое и стабильное состояние
- Освободить все ресурсы, принадлежащие процессу или использовавшиеся процессом
- Освободить ВАП и уничтожить его
- Оповестить подсистемы, принимающие участие в управлении процессами, о завершении процесса
- Установит состоянии процесса в значение «завершён»

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

СОЗДАНИЕ / ЗАВЕРШЕНИЕ ПРОЦЕССА WINAPI

```
WINBASEAPI
BOOL
WINAPI
CreateProcessW(
    _In_opt_ LPCWSTR lpApplicationName,
    _Inout_opt_ LPWSTR lpCommandLine,
    _In_opt_ LPSECURITY_ATTRIBUTES lpProcessAttributes,
    _In_opt_ LPSECURITY_ATTRIBUTES lpThreadAttributes,
    _In_ BOOL bInheritHandles,
    _In_ DWORD dwCreationFlags,
    _In_opt_ LPVOID lpEnvironment,
    _In_opt_ LPCWSTR lpCurrentDirectory,
    _In_ LPSTARTUPINFOW lpStartupInfo,
    _Out_ LPPROCESS_INFORMATION lpProcessInformation
    );
```

```
VOID
WINAPI
ExitProcess(
    _In_ UINT uExitCode
    );
```

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);

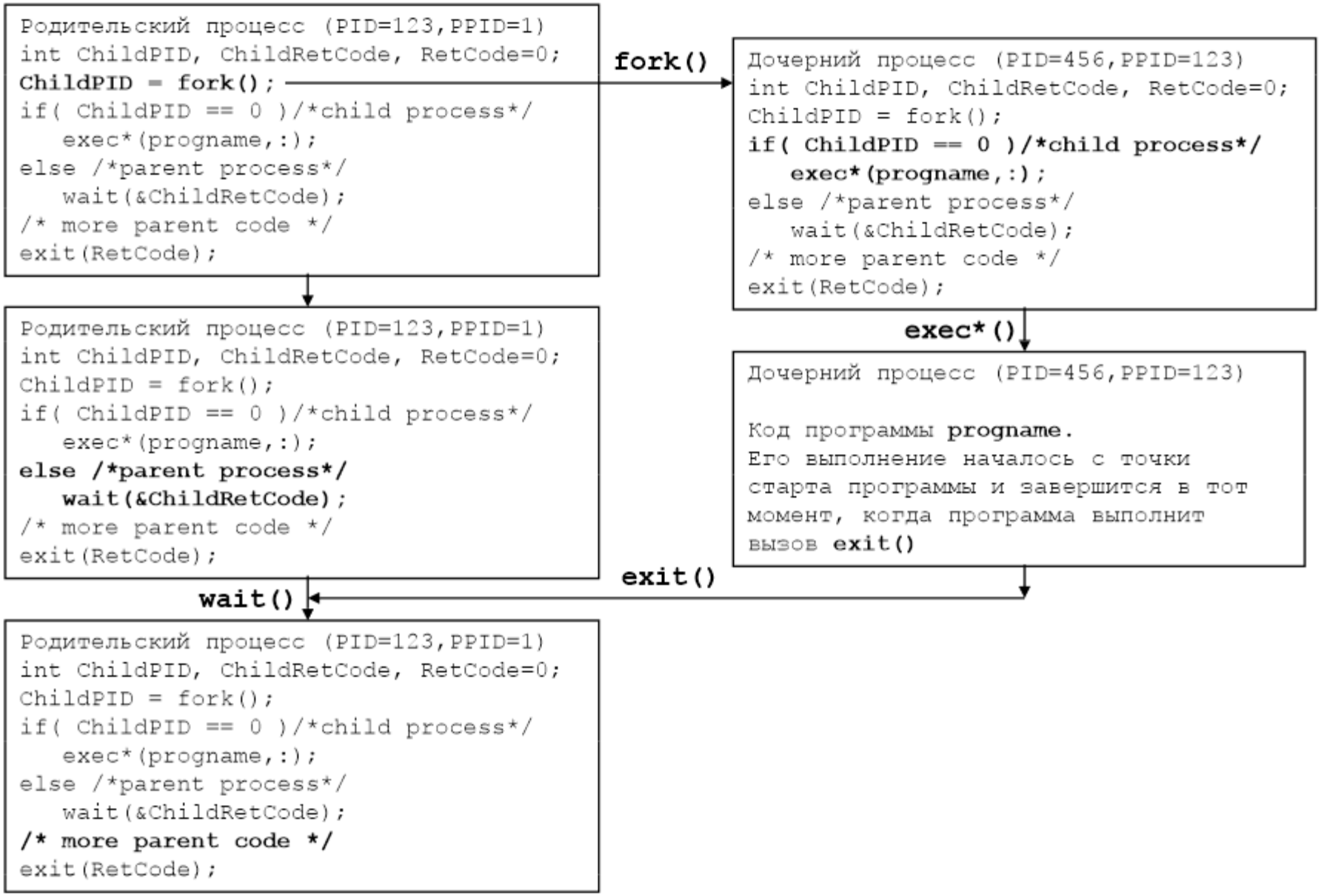
- Ожидание завершения процесса-потомка

  int wait(int *status);

Описание функций можно найти в Описание функций можно найти в документации UNIX (man или info)

ПРИМЕР ДЛЯ UNIX

```
/*Программа, создающая процесс-потомок и запускающая в потомке другую программу */
int ChildPID, ChildRetCode, RetCode = 0;
ChildPID = fork();
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);