Управление процессами

Определение процесса. Основные понятия

Процесс — совокупность машинных команд и данных, которая обрабатывается в рамках вычислительной системы и обладает правами на владение некоторым набором *ресурсов*.

Ресурсы могут принадлежать только одному процессу, либо ресурсы могут разделяться между процессами — **разделяемые ресурсы**.

Выделение ресурсов процессу

- предварительная декларация — до начала выполнения
- динамическое пополнение списка принадлежащих процессу ресурсов

Количество допустимых процессов в системе — *ресурс ВС*.

Жизненный цикл процесса

Основной из задач ОС является поддержание жизненного цикла процесса.

Типовые этапы обработки процесса в системе

Жизненный цикл процесса в системе:

- Образование (порождение) процесса
- Обработка (выполнение) процесса
- Ожидание (по тем или иным причинам) постановки на выполнение
- Завершение процесса

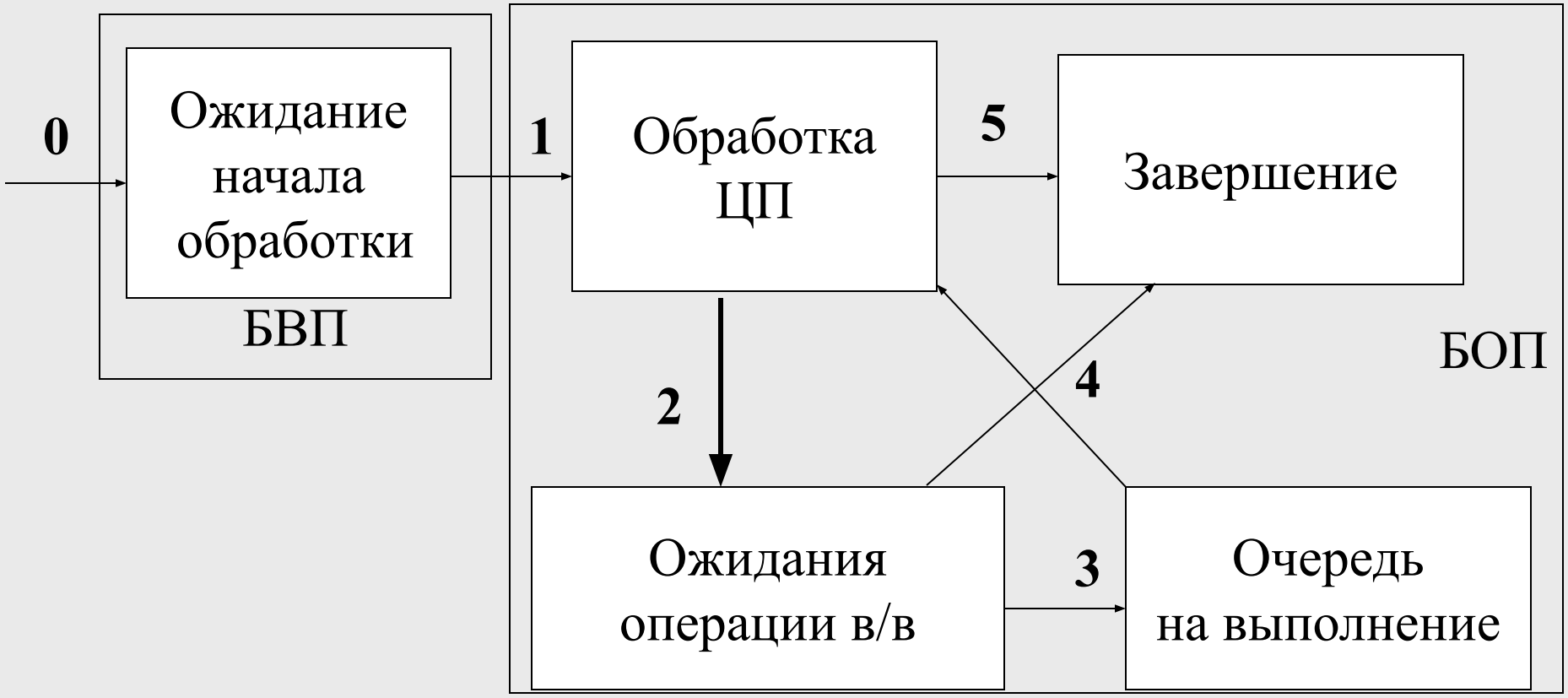
Модельная ОС

1. **Буфер ввода процессов** (**БВП**) — пространство, в котором размещаются и хранятся сформированные процессы с момента их образования, до момента начала выполнения.
2. **Буфер обрабатываемых процессов** (**БОП**) — буфер для размещения процессов, находящихся в системе в мультипрограммной обработке.

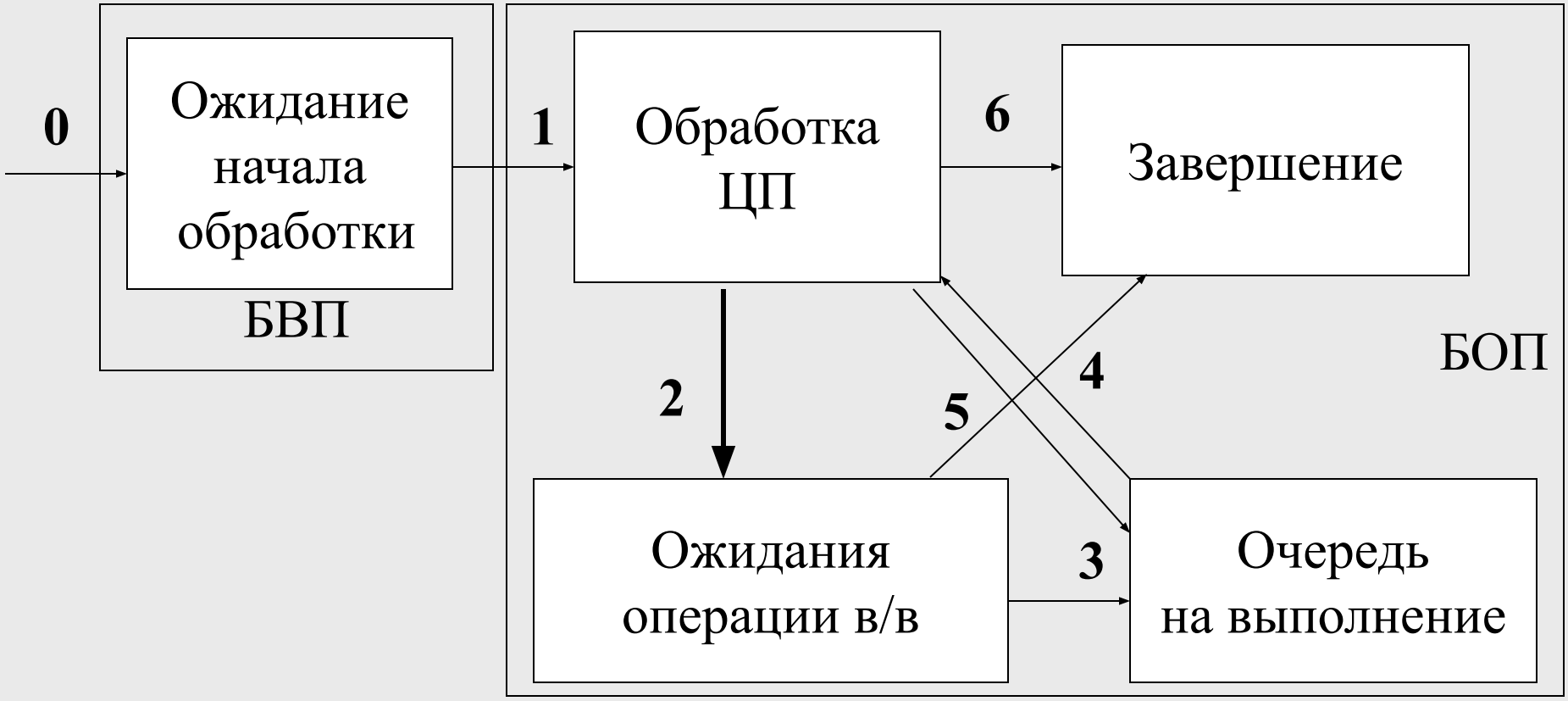
Модель пакетной однопроцессной системы

0. Поступление процесса в очередь на начало обработки ЦП (процесс попадает в БВП)
1. Начало обработки процесса на ЦП (из БВП в БОП)
2. Завершение выполнения процесса, освобождение системных ресурсов.

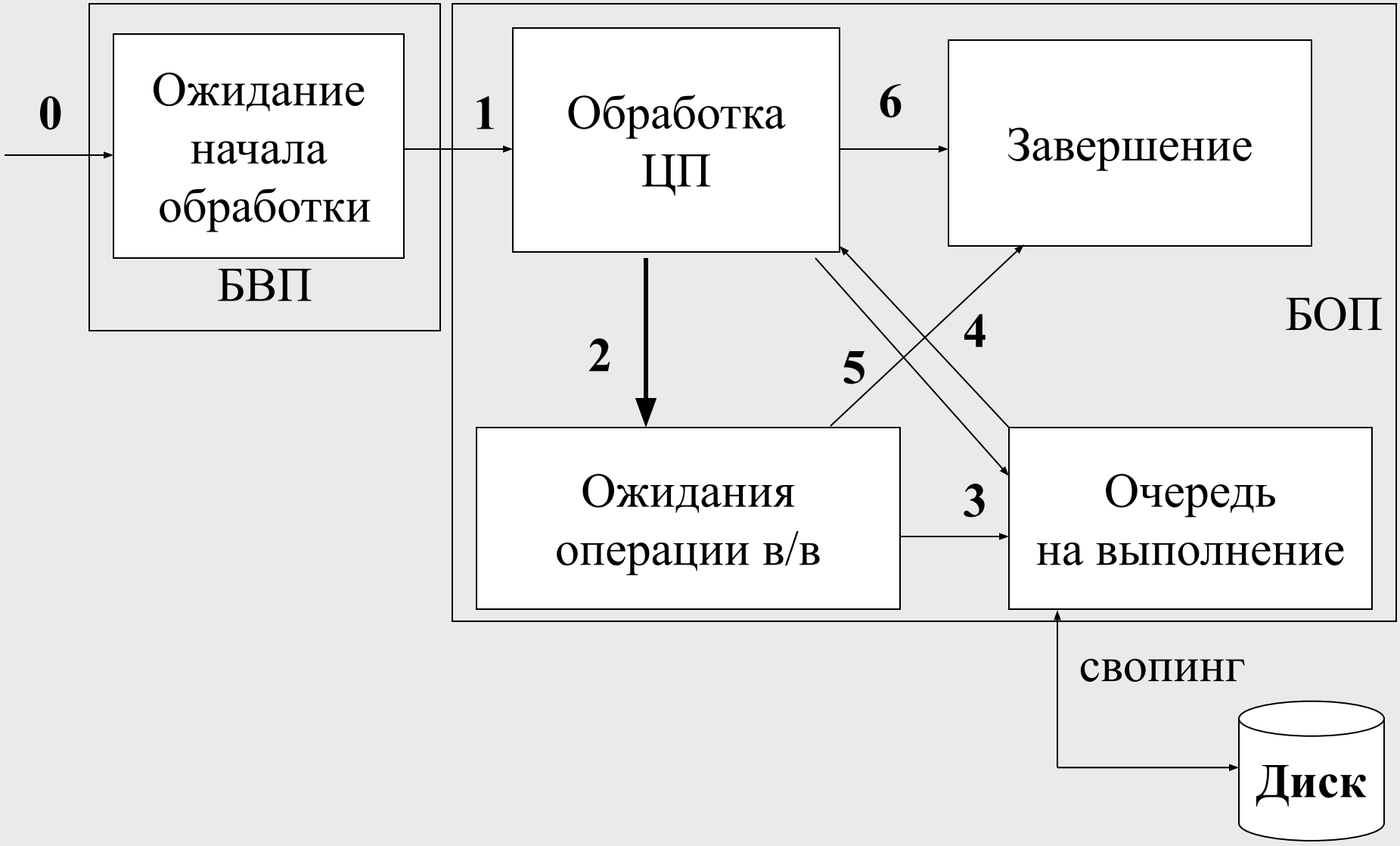
Модель пакетной мультипроцессной системы



Модель ОС с разделением времени



Модель ОС с разделением времени (модификация)



Типы процессов

- **«полновесные» процессы**
- **«легковесные» процессы**

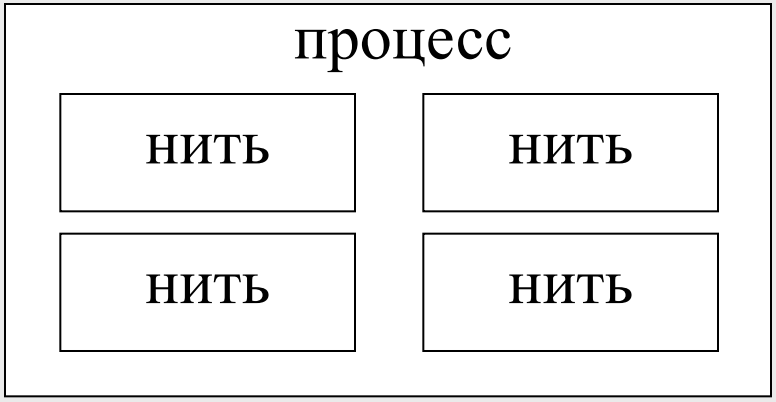
«Полновесные процессы» — процессы, выполняющиеся внутри защищенных участков оперативной памяти.

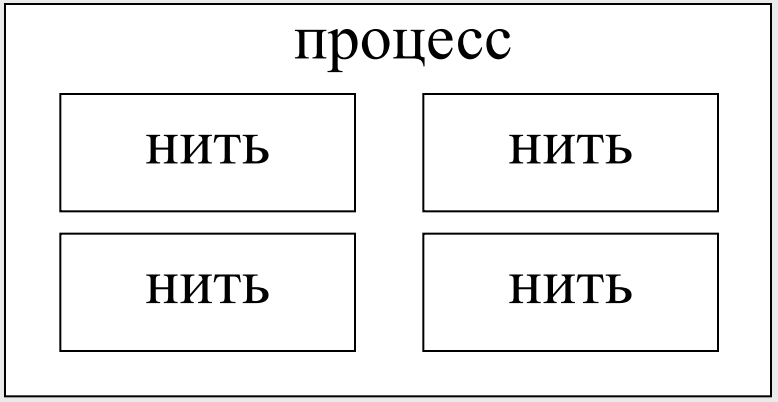
«Легковесные процессы» (нити) — работают в мультипрограммном режиме одновременно с активировавшей их задачей и используют ее виртуальное адресное пространство.

Типы процессов

Однонитевая организация процесса — «один процесс — одна нить»:

Многонитевая организация процесса:





Понятие «процесс»

Понятие «**процесс**» включает в себя следующее:

- исполняемый код
- собственное адресное пространство, которое представляет собой совокупность виртуальных адресов, которые может использовать процесс
- ресурсы системы, которые назначены процессу ОС
- хотя бы одну выполняемую нить

Контекст процесса

Контекст процесса — совокупность данных, характеризующих актуальное состояние процесса.

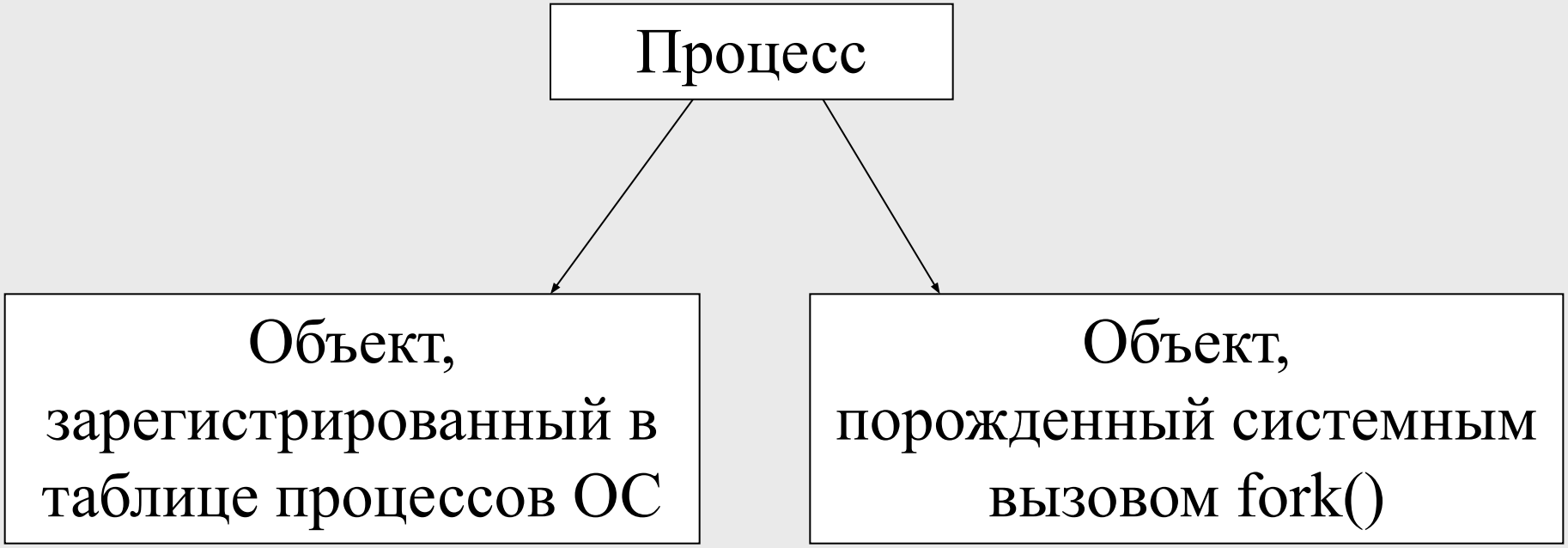
- Пользовательская составляющая — текущее состояние программы (совокупность машинных команд и данных, размещенных в ОЗУ)
- Системно-аппаратная составляющая

  · информация идентификационного характера (PID процесса, PID «родителя»…)

  · информация о содержимом регистров, настройках аппаратных интерфейсов, режимах работы процессора и т.п.

  · информация, необходимая для управления процессом (состояние процесса, приоритет).

Определение процесса Unix

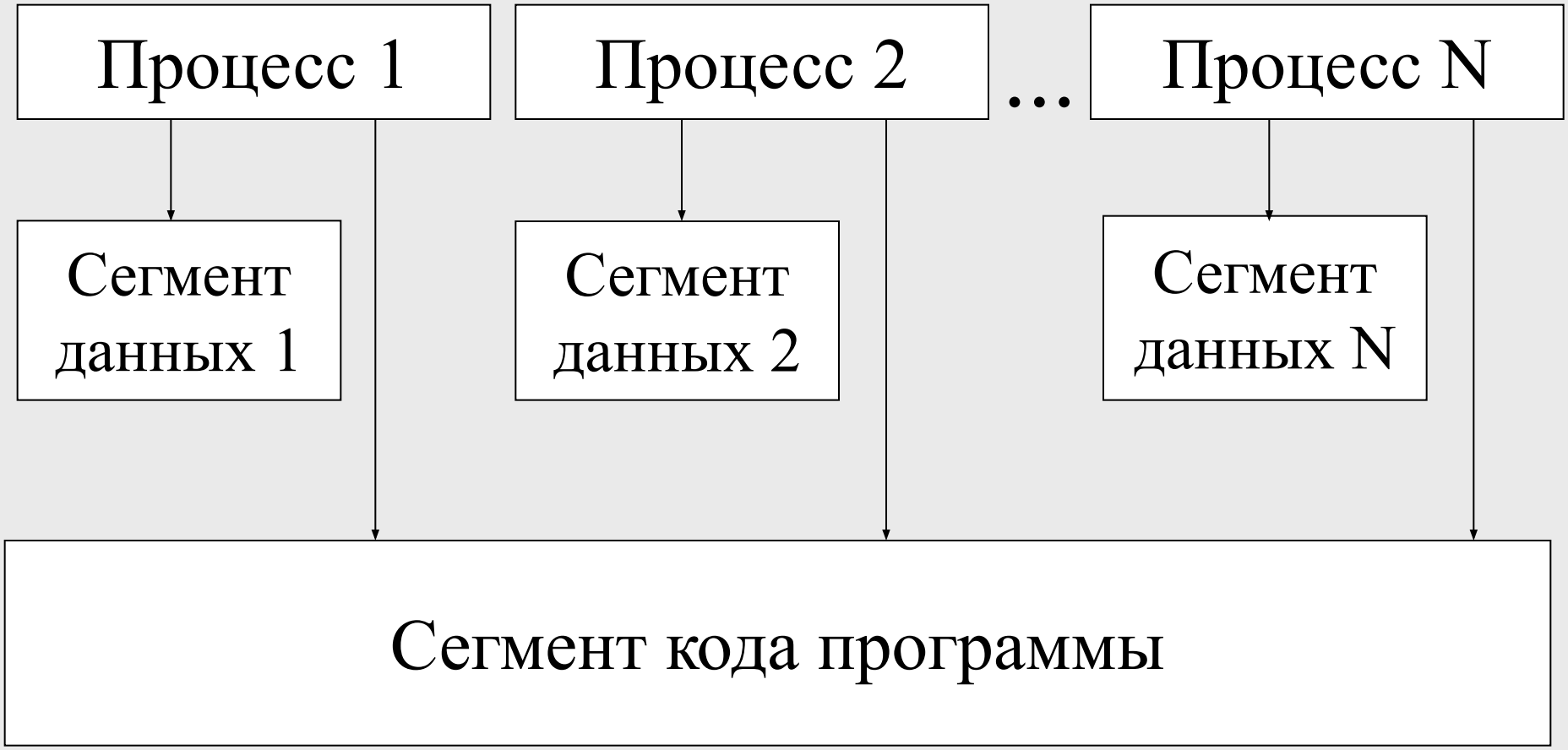


Определение процесса в UNIX

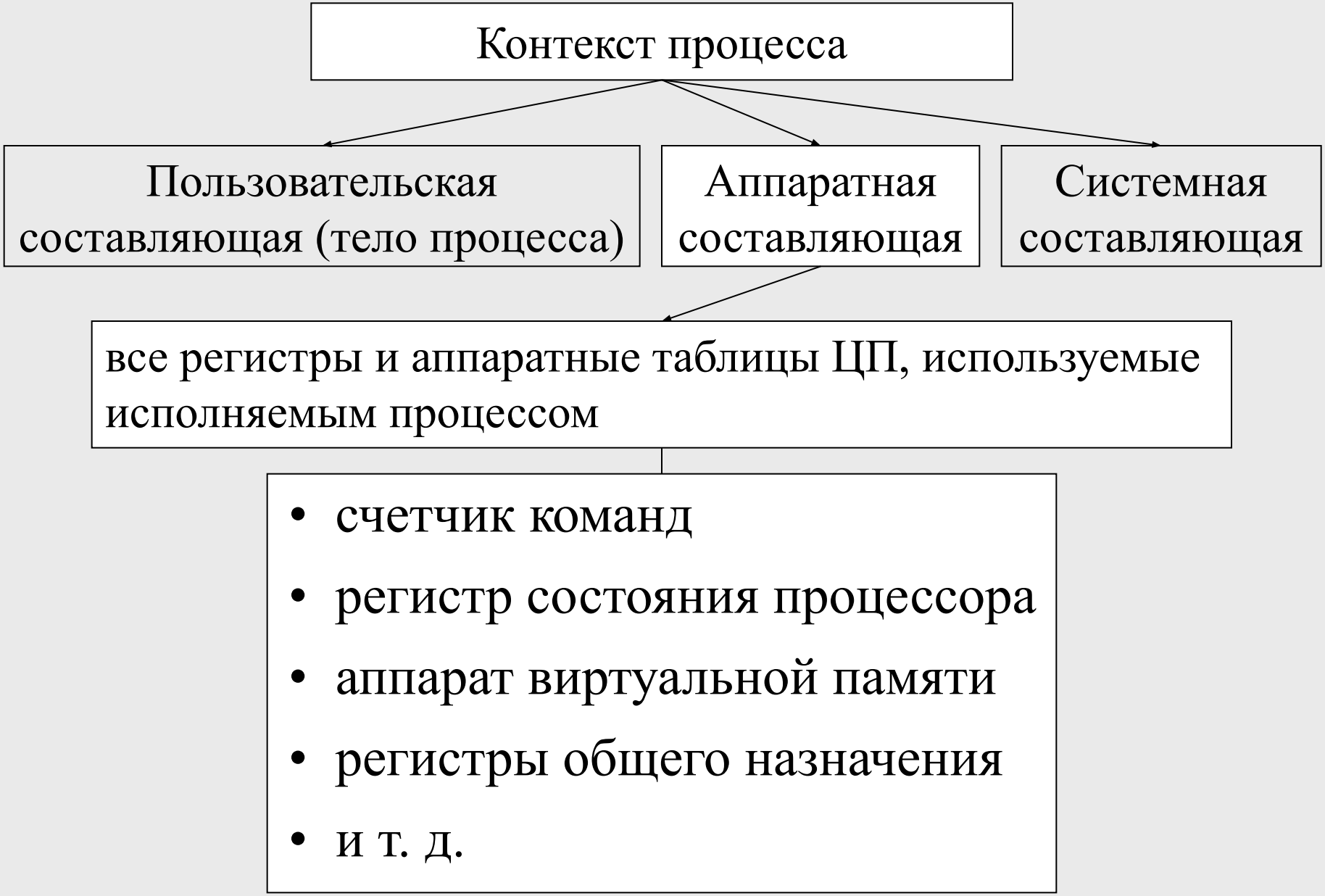
Процесс в UNIX — объект, зарегистрированный в таблице процессов UNIX.

Контекст процесса

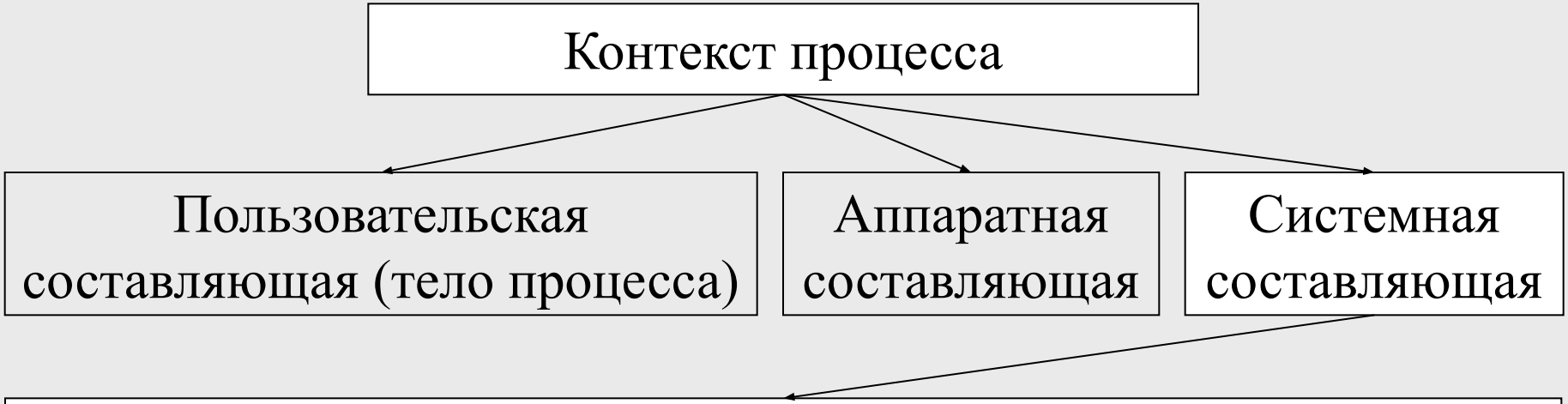
Разделение сегмента кода



Контекст процесса



Контекст процесса



- идентификатор родительского процесса
- текущее состояние процесса
- приоритет процесса
- реальный и эффективный идентификаторы пользователя-владельца
- реальный и эффективный идентификатор идентификатор группы, к которой принадлежит владелец
- список областей памяти
- таблица открытых файлов процесса
- информация об установленной реакции на тот или иной сигнал
- информация о сигналах, ожидающих доставки в данный процесс
- сохраненные значения аппаратной составляющей

Второе определение процесса в UNIX

Процесс в UNIX — это объект, порожденный системным вызовом **fork()**.

Системный вызов — обращение процесса к ядру ОС за выполнением тех или иных действий.

Создание нового процесса

#include <sys/types.h>
#include <unistd.h>
pid_t fork (void);

- При удачном завершении возвращается:
  - сыновнему процессу значение 0
  - родительскому процессу PID порожденного процесса
- При неудачном завершении возвращается –1, код ошибки устанавливается в переменной **errno**

- Заносится новая запись в таблицу процессов
- Новый процесс получает уникальный идентификатор
- Создание контекста для сыновнего процесса

Создание нового процесса

Составляющие контекста, наследуемые при вызове fork()

- Окружение
- Файлы, открытые в процессе-отце
- Способы обработки сигналов
- Разрешение переустановки эффективного идентификатора пользователя
- Разделяемые ресурсы процесса-отца
- Текущий рабочий каталог и домашний каталоги
- …

Создание нового процесса

Составляющие контекста, не наследуемые при вызове fork()

- Идентификатор процесса (PID)
- Идентификатор родительского процесса (PPID)
- Сигналы, ждущие доставки в родительский процесс
- Время посылки ожидающего сигнала, установленное системным вызовом **alarm()**
- Блокировки файлов, установленные родительским процессом

Схема создания нового процесса

PID = 2757

```
main()                сегмент кода
{ …
if ( (pid=fork()) > 0 )
{ … }
else
{ … }
}
```

fork()

PID = 2757

```
main()                сегмент кода
{ …
if ( (pid=fork()) > 0 )
{ … }
else
{ … }
}
```

Предок: выполняются операторы в if-секции

PID = 2760

```
main()                сегмент кода
{ …
if ( (pid=fork()) > 0 )
{ … }
else
{ … }
}
```

Потомок: выполняются операторы в else-секции

Пример

```
int main ( int argc, char **argv )
{
   printf ( "PID = %d; PPID = %d \n", getpid(), getppid() );
   fork ();
   printf ( "PID = %d; PPID = %d \n", getpid(), getppid() );
   return 0;
}
```

Семейство системных вызовов exec()

#include <unistd.h>

int execl (const char *path, char *arg0, …, char *argn, 0);

- **path** — имя файла, содержащего исполняемый код программы
- **arg0** — имя файла, содержащего вызываемую на выполнение программу
- **arg1, …, argn** — аргументы программы, передаваемые ей при вызове

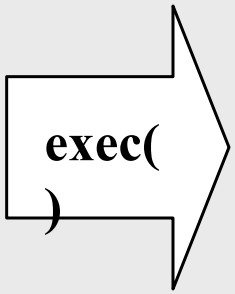
Возвращается:	
в случае ошибки	–1

Семейство системных вызовов exec()

Схема работы системного вызова exec()

PID = 2760

```
main()
{
   execl("/bin/ls", "ls",
   " l", (char*)0);
}
```



PID = 2760

```
main()
{
   // реализация программы
   ls
}
```

Семейство системных вызовов exec()

Сохраняются:

- Идентификатор процесса
- Идентификатор родительского процесса
- Таблица дескрипторов файлов
- Приоритет и большинство атрибутов

Изменяются:

- Режимы обработки сигналов
- Эффективные идентификаторы владельца и группы
- Файловые дескрипторы (закрытие некоторых файлов)

Пример

```
#include <unistd.h>

int main ( int argc, char **argv )
{
    …
    /* тело программы */
    …
    execl ( “/bin/ls”, ”ls”, ”-l”, (char*) 0 );
    /* или execlp ( “ls” ,”ls”, ”-l”, (char*) 0 ); */
    printf ( “это напечатается в случае неудачного обращения к
    предыдущей функции, к примеру, если не был найден файл
    ls.\n” );
    …
}
```

Использование схемы fork-exec

PID = 2757

```
main()
{
    if((pid=fork())== 0)
    {
        execl(“/bin/ls”, ”ls”, ”
-l”,
        (char*)0);
    } else {…}
}
```

PID = 2760

```
main ()
{
// реализация программы
}
```

fork()

exec()

PID = 2757

```
main()
{
    if((pid=fork())== 0)
    {
        execl(“/bin/ls”, ”ls”, ”
-l”,        (char*)0);
    } else {…}
}
```

PID = 2760

```
main()
{
    if((pid=fork())== 0)
    {
        execl(“/bin/ls”, ”ls”, ”
-l”,        (char*)0);
    } else {…}
}
```

Завершение процесса

- Системный вызов **_exit()**
- Выполнение оператора **return**, входящего в состав функции **main()**
- Получение сигнала

Завершение процесса

#include <unistd.h>
void _exit (int status);

status :

= 0 при успешном завершении

0 при неудаче (возможно, номер варианта)

Завершение процесса

- Освобождается сегмента кода и сегмента данных процесса
- Закрываются все открытые дескрипторы файлов
- Если у процесса имеются потомки, их предком назначается процесс с идентификатором 1
- Освобождается большая часть контекста процесса (кроме статуса завершения и статистики выполнения)
- Процессу-предку посылается сигнал SIGCHLD

Получение информации о завершении своего потомка

#include <sys/types.h>
#include <sys/wait.h>
pid_t wait (int *status);

status по завершению содержит:

- в старшем байте — код завершения процесса-потомка (**пользовательский код завершения процесса**)
- в младшем байте — индикатор причины завершения процесса-потомка, устанавливаемый ядром UNIX (**системный код завершения процесса**)

Возвращается: PID завершенного процесса или –1 в случае ошибки или прерывания

Получение информации о завершении своего потомка

- Приостановка родительского процесса до завершения (остановки) какого-либо из потомков
- После передачи информации о статусе завершения предку, все структуры, связанные с процессом-«зомби» освобождаются, удаляется запись о нем из таблицы процессов

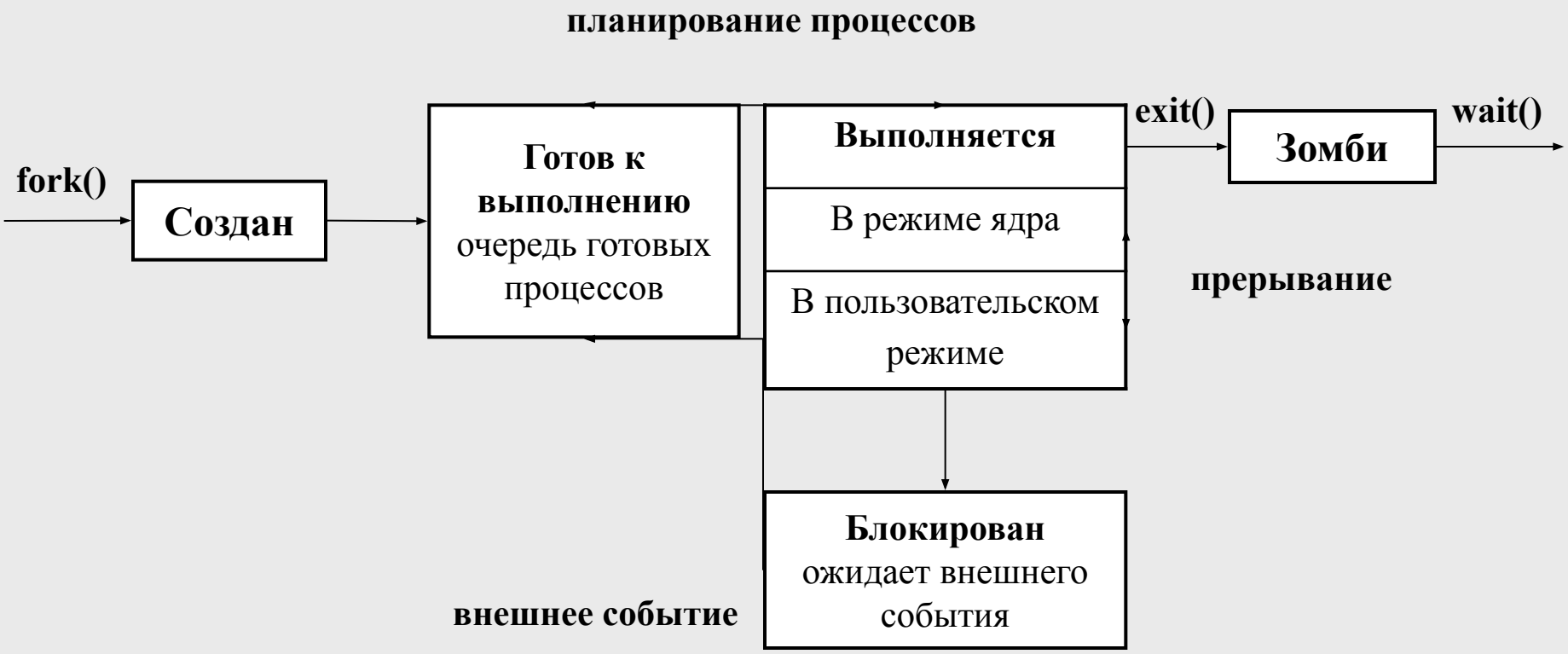
Пример. Использование системного вызова wait()

```
#include <stdio.h>
int main ( int argc, char **argv )
{
    int i;
    for ( i=1; i<argc; i++ ) {
     int status;
     if ( fork () > 0 ) {
          wait( &status );
          printf( “process-father\n” );
          continue;
     }
     execlp ( argv[i], argv[i], 0 );
     exit ();
    }
}
```

file prog1 prog2 prog3

<текст от prog1>
process-father
<текст от prog2>
process-father
<текст от prog3>
process-father

Жизненный цикл процессов



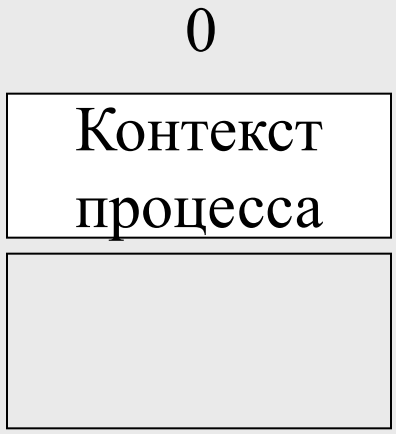
Начальная загрузка

Начальная загрузка — загрузка ядра системы в оперативную память, запуск ядра.

- Чтение нулевого блока системного устройства аппаратным загрузчиком
- Поиск и считывание в память файла /unix
- Запуск на исполнение файла /unix

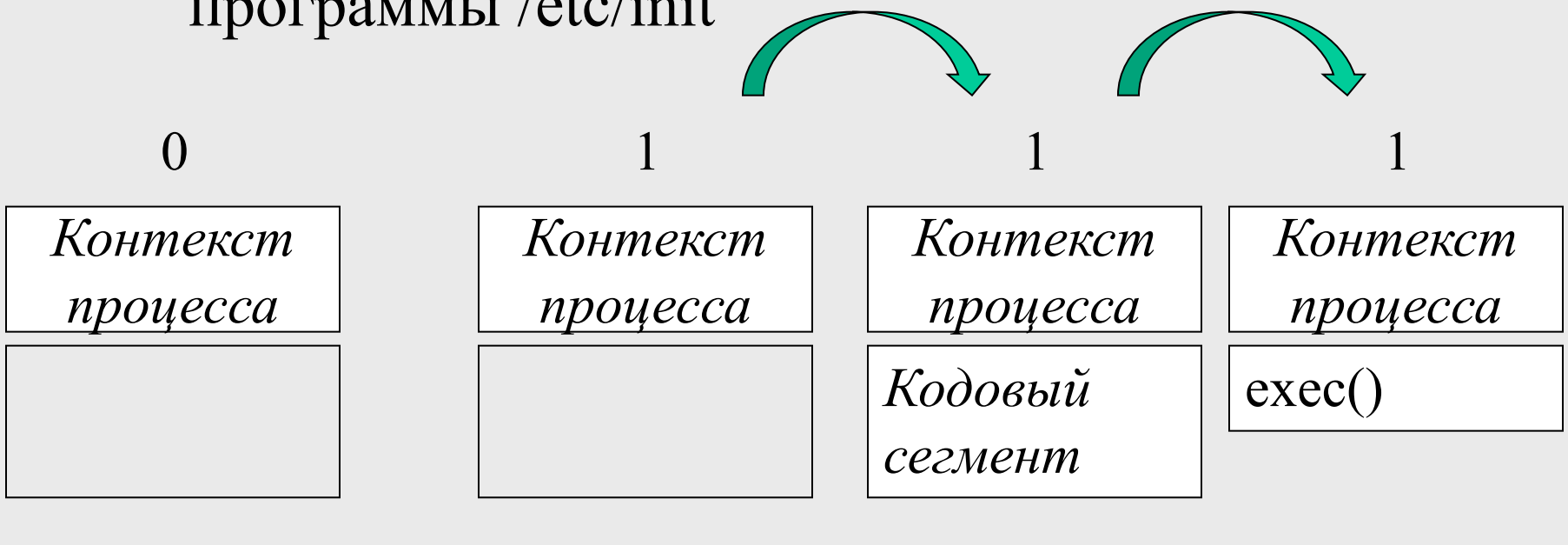
Инициализация Unix системы

- Начальная инициализация компонентов компьютера (установка часов, инициализация контроллера памяти и пр.)
- Инициализация системных структур данных
- Инициализация процесса с номером “0”:
  - не имеет кодового сегмента
  - существует в течении всего времени работы системы



Инициализация системы

- Создание ядром первого процесса
  - Копируется процесс “0” (запись таблицы процессов)
  - Создание области кода процесса “1”
  - Копирование в область кода процесса “1” программы, реализующей системный вызов **exec()**, который необходим для выполнения программы /etc/init



Инициализация системы

- Замена команды процесса “1” кодом из файла /etc/init (запуск exec())
- Подключение интерпретатора команд к системной консоли
- Создание многопользовательской среды

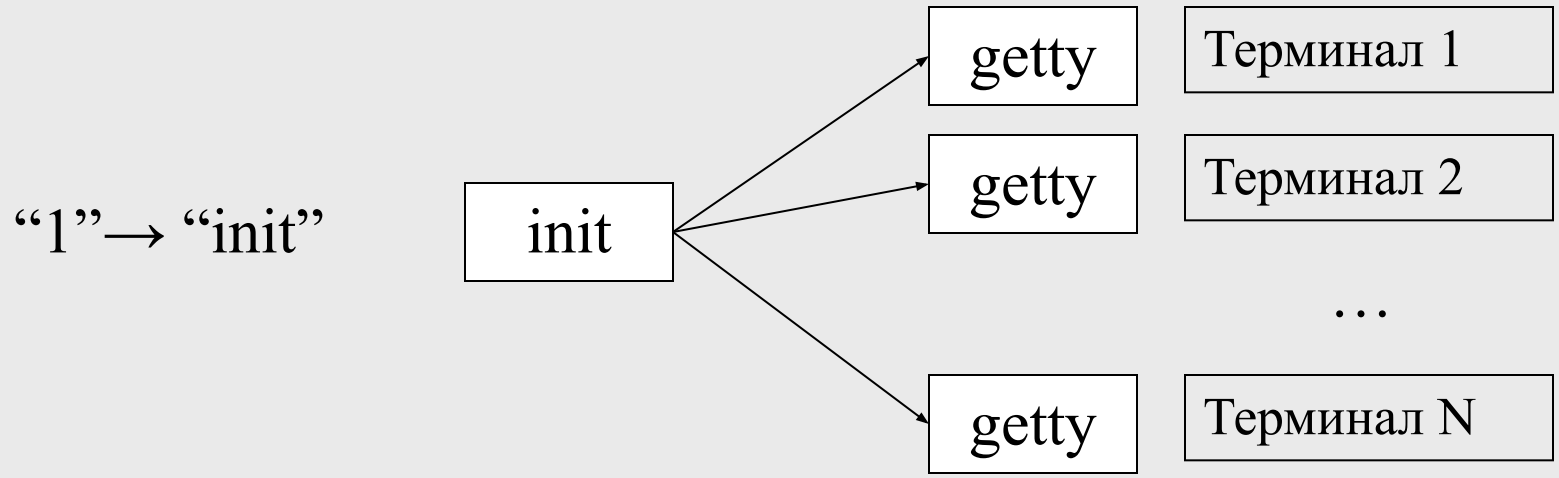


Схема дальнейшей работы системы