Операционные системы
Лекция 2
Процессы

Боровцов Е.Г., 2016 г.

Основные концепции ОС

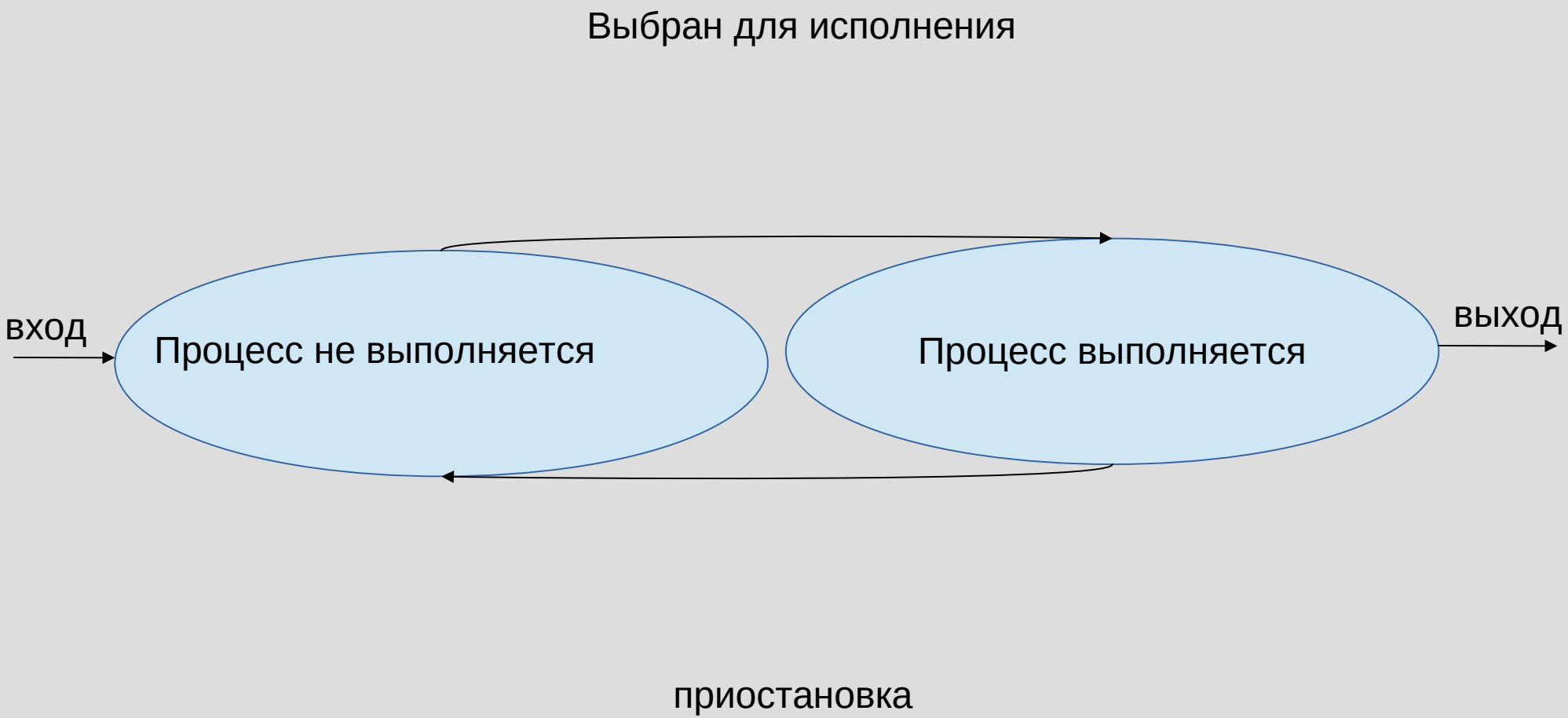
- Системные вызовы(System calls) (программные прерывания);
- Прерывания (Hardware Interrupts) (внешние аппаратные прерывания);
- Исключения (Exeptions) (внутренние аппаратные прерывания);
- Файлы и файловые системы;
- Процессы и нити;
- Блоки памяти, страничная и виртуальная память;

Понятие процесса

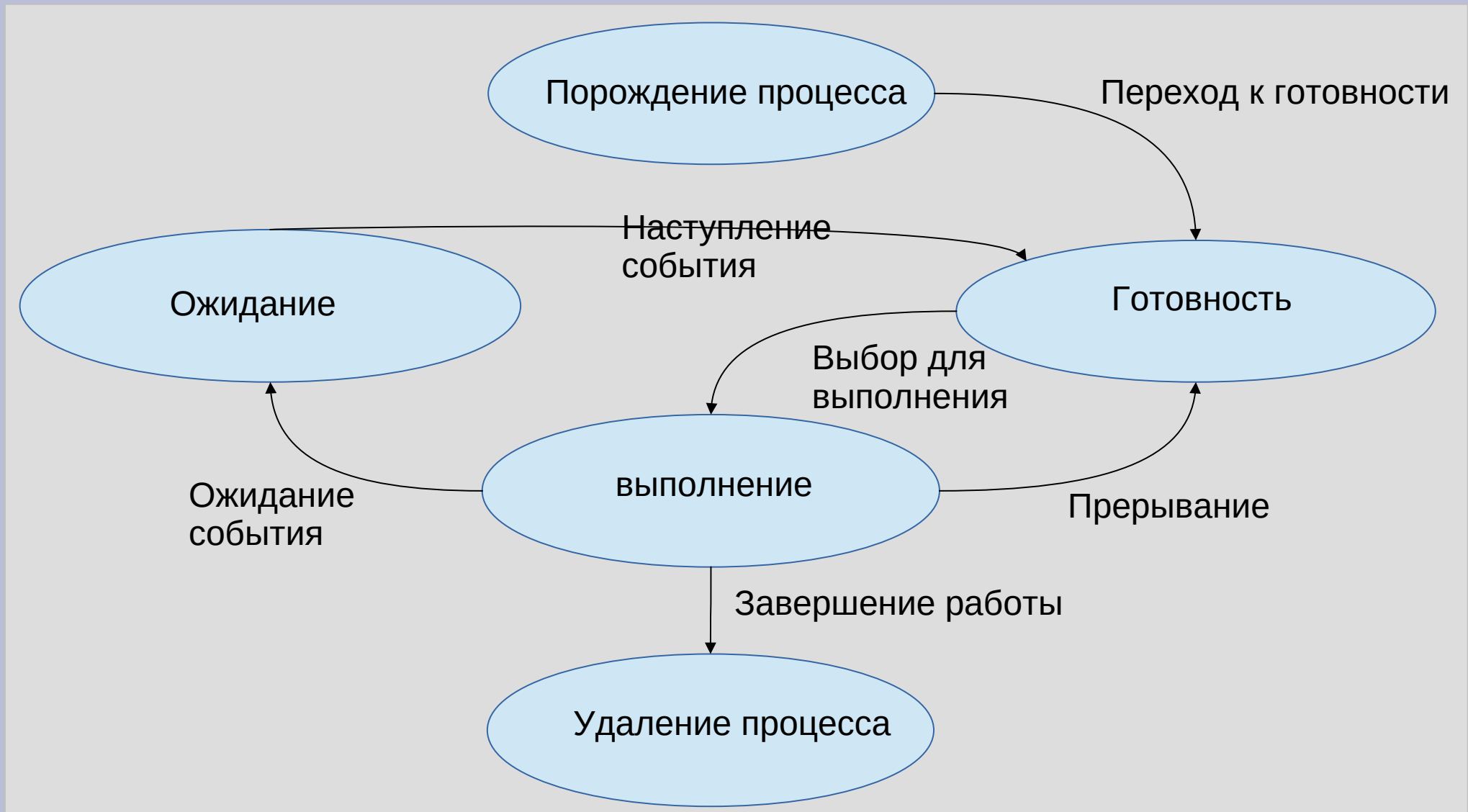
Процесс - это совокупность находящихся под управлением ОС:

- Последовательность исполняющихся команд;
- Соответствующие ресурсы (выделенная для выполнения память или адресное пространство, файлы, устройства ввода/вывода и т. д.;
- текущее состояние — состояние программного счетчика и регистра флагов, состояние регистров, стека, рабочих переменных;

Состояния процесса



Состояния процесса-2



Создание процесса

1. Инициализация системы;
2. Выполнение системного запроса на создание процесса от работающего процесса (fork - UNIX, CreateProcess — Windows API);
3. Запрос пользователя на создание процесса;
4. Инициирование пакетного задания (bat-файла, shell-скрипта)

Создание процесса - fork()

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main() {
pid_t ChildPID;

printf("\n\r **** Begin Process ****");
ChildPID=fork();
if(ChildPID< 0)  {
  printf("\n\r **** Abnormal fork termination ****");
  return 1;
}
if(ChildPID==0){
    // Child Process
    printf("\n\r **** Child Process PID **** %d \n\r",getpid());
} else {
    // Parent process
    printf("\n\r **** Parent Process PID **** %d \n\r",getpid());
    wait(NULL);
}
return 0;
}
```

Создание процесса - fork()+execve

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

extern char **environ;

int main(void) {
pid_t ChildPID;
char * ls_args[]={
       "ls","-l","/",NULL
};

ChildPID=fork();
if(ChildPID< 0)  {
  fprintf(stderr,"\n\r **** Abnormal fork termination ****");
  return 1;
}
if(ChildPID==0){
    // Child Process
    execve("/bin/ls",ls_args,environ);
    fprintf(stderr,"\n\r!!!!!! EXEC ERROR !!!!!\n");
    return 1;
}

return 0;
}
```

Завершение процесса

1. Нормальный выход («код возврата»=0, преднамеренно);
2. Выход по ошибке(«код возврата»> 0, преднамеренно);
3. Выход по неисправимой ошибке (непреднамеренно);
4. Уничтожение другим процессом (непреднамеренно)

Переходы в состояния «готовность», «выполнение», «ожидание»

Порождение процесса — готовность:
после порождения процесса тем или иным способом, выделения процессу ресурсов, создания и заполнения блока управления процессом;

Готовность — выполнение:
выбор процесса и очереди готовых к выполнению процессов в соответствии с используемым в системе алгоритмом планирования; *Осуществляется восстановление контекста;*

Выполнение — готовность:
аппаратное прерывание для обработки внешнего события; *Осуществляется сохранение контекста процесса;*

Выполнение — ожидание(блокировка):
Системный вызов **(**запрос доступа к ресурсам); *Осуществляется сохранение контекста процесса;*

Ожидание — готовность:
доступность ожидаемых данных или ресурсов.

Дерево процессов

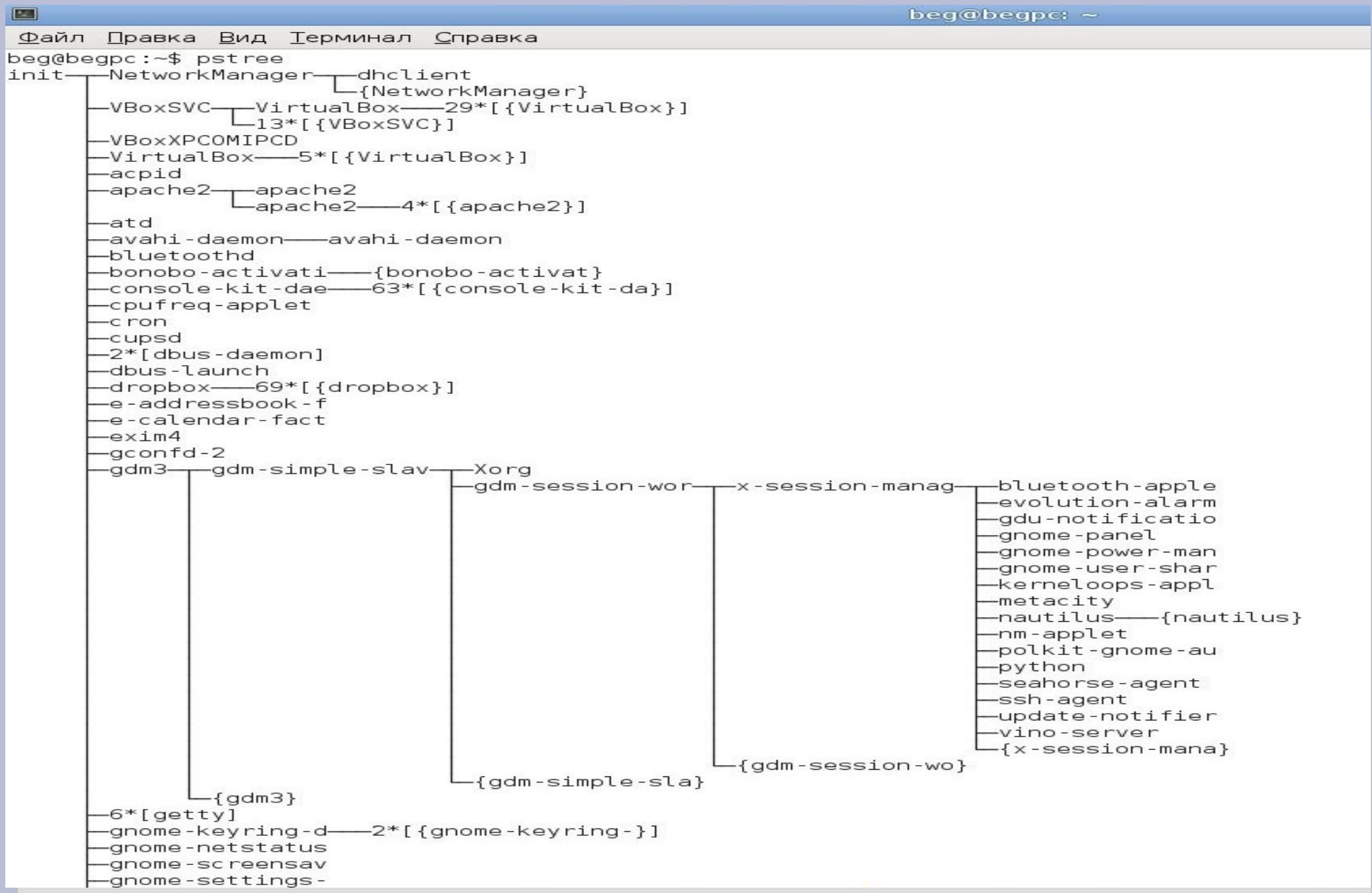


Таблица процессов и содержимое блока управления процессом

Управление процессом

- счётчик команд IP;
- регистры;
- psw(регистр флагов);
- указатель стека SP;

- текущее сост. процесса;
- значение приоритета;
- параметры планирования;
- PID;
- PPID;
- сигналы;
- GID процесса;
- время начала процесса;
- использованное CPU Time;
- Child CPU Time;

Управление памятью

- указатель на сегмент кода;
- указатель на сегмент данных;
- указатель на сегмент стека;

Управление файлами

- корневой каталог;
- рабочий каталог;
- дескрипторы файлов;
- UID;
- GID;

Потоки(нити)

В отличие от процессов потоки(нити) работают в контексте одного процесса-родителя

Совместно используемые Элементы процесса	**элементы потока**
Адресное пространство	Счетчик команд
Глобальные переменные	Регистры CPU
Открытые файлы	Стек
Дочерние процессы	Состояние
Необработанные аварийные сигналы	
Сигналы и их обработчики	
Информация об использовании ресурсов	

Реализация потоков:

- В пространстве ядра
- В пространстве пользователя

Потоки в пространстве ядра

Достоинства:

- возможно планирование работы нескольких потоков одного и того же процесса на нескольких процессорах (ядрах);
- реализуется мультипрограммирование в рамках всех процессов;
- при блокировании одного из потоков процесса ядро может выбрать для исполнения другой поток этого же процесса;
- процессы ядра сами могут быть многопоточными;

Недостатки:

необходимость двукратного переключения режима пользователь-ядро, ядро-пользователь для передачи управления от одного потока другому в рамках одного и того же процесса.

Потоки в пространстве пользователя

Достоинства:
- можно реализовать «собственными » средствами в ОС, не поддерживающей потоки без каких-либо изменений в ядре ОС;
- высокая производительность, поскольку при переключении потоков процессу не надо переключаться в режим ядра и обратно;
- ядро о потоках ничего не знает и управляет однопоточными процессами;
- имеется возможность применять любые алгоритмы планирования исполнения потоков с учетом специфики решаемой задачи;
- правление потоками возлагается на программу пользователя;

Недостатки:
- системный вызов блокирует не только работающий поток, но и все потоки того процесса, которому он принадлежит;
- приложение не может работать в многопроцессорном режиме, так как ядро выделяет каждому процессу один процессор;
- при запуске одного потока другой поток в рамках одного процесса не будет запущен, пока первый добровольно не освободит ресурсы;
- внутри одного потока нет прерываний по таймеру, в результате чего невозможно создать в рамках отдельного процесса собственный планировщик по таймеру для поочередного выполнения потоков.