

Лекция 3

Содержание

- Объекты ядра.
- Процессы. Их реализация и управление ими.
- Создание процессов в Linux.

Объекты ядра операционной системы:

- **Process**
- Thread
- File
- File-mapping
- Pipe
- Event
- Mutex
- Semaphore
- ...

Процессы

Процесс – это исполняемый экземпляр программы и набор ресурсов, которые выделяются данной исполняемой программе.

Ресурсы процесса:

- виртуальное адресное пространство;
- системные ресурсы –области физической памяти, процессорное время, файлы, растровые изображения и т.д.;
- модули процесса, то есть исполняемые модули, загруженные (отображенные) в его адресное пространство – основной загрузочный модуль, библиотеки динамической компоновки, драйверы устройств и т.д.;
- уникальный идентификационный номер, называемый идентификатором процесса;
- потоки (по крайней мере, один поток).

Машинные команды. Выполнение программы.

Центральный процессор

Шина

ОЗУ

Арифметико-
логическое
устройство

Устройство
управления

0



A0

1



0011 0101 1110 0111

...



F



00



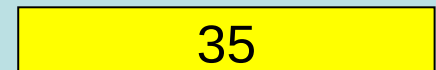
01



XY



A0



A1




XY



FF



 Регистры общего назначения – сумматор, регистр данных, адресный регистр и т.д.



Счетчик команд

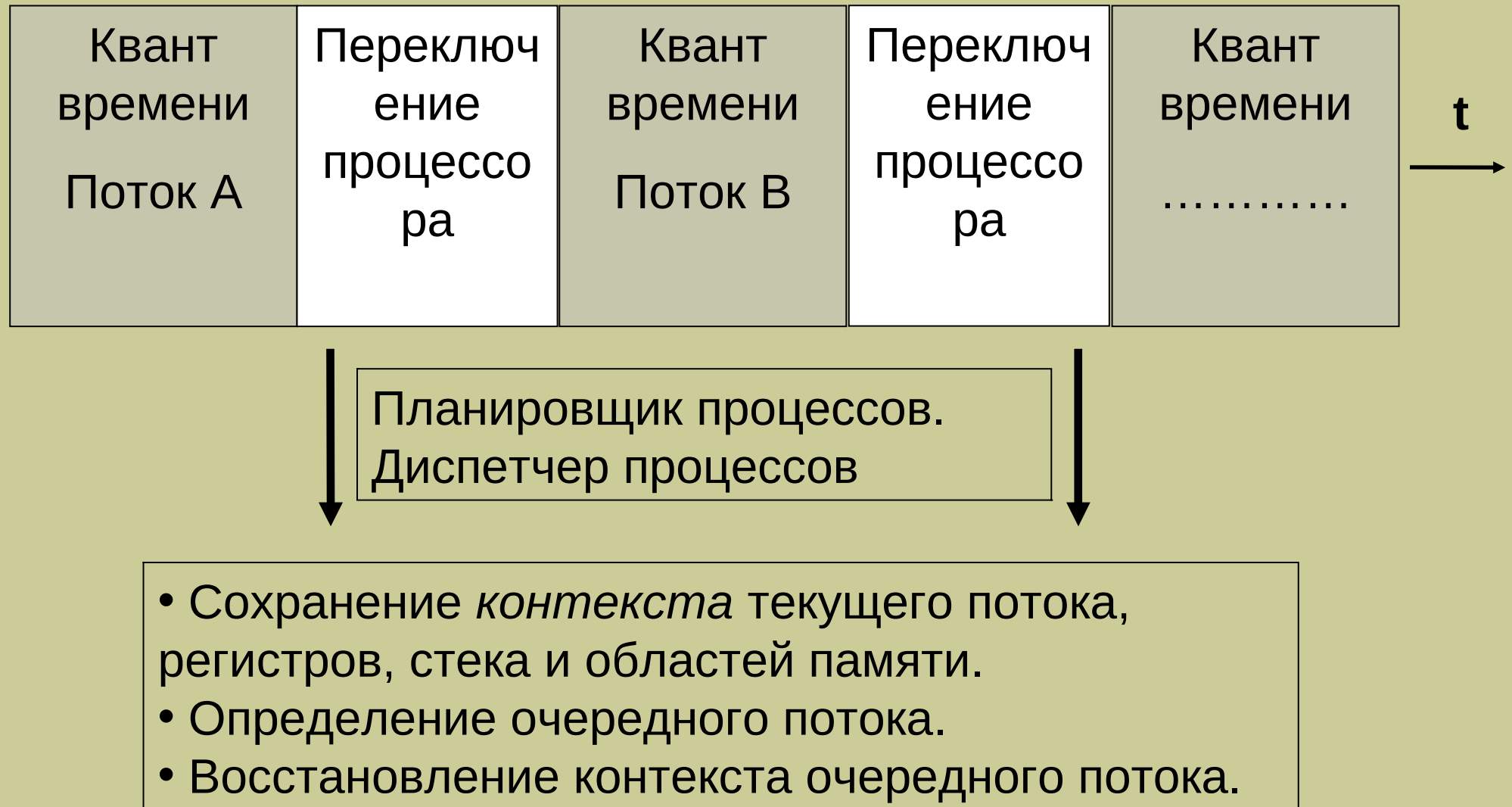


Ячейки памяти



Регистр команд

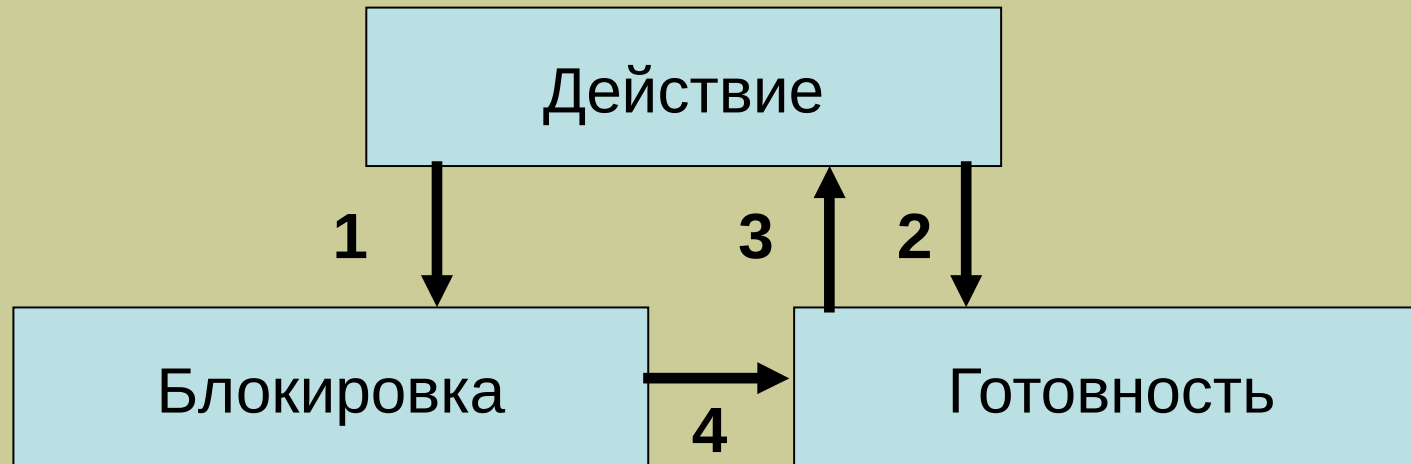
Модель процесса:



Последовательность исполнения потоков в среде с вытесняющей многозадачностью:

В системе определен *квант времени* (порядка десятков миллисекунд) – процессорное время выделяемое одному потоку (каждому - своё). ***Длительность выполнения одного потока не может превышать одного кванта.*** Когда это время заканчивается, *диспетчер процессов переключает процессор* на выполнение другого потока. При этом состояние регистров, стека и областей памяти – *контекст потока*, сохраняется в *стеке потока*. Очередность потоков определяется их *состоянием и приоритетом*.

Состояние процессов:



1. Процесс заблокирован в ожидании ввода.
2. Диспетчер выбирает другой процесс.
3. Диспетчер выбирает данный процесс.
4. Входные данные стали доступны.

Реализацией процессов является **таблица процессов**, - линейный список (программно реализованный, как массив структур) (***P**rocess **C**ontrol **B**lock*).

Информация о процессах хранится в *таблице процессов* и обновляется *планировщиком процессов*.

Некоторые поля типичной записи таблицы процессов:

Регистры
Счетчик команд
Состояние процесса
Приоритет
Идентификатор процесса
Родительский процесс
Время запуска процессора
Использованное время
процессора

Корневой каталог
Рабочий каталог
Дескрипторы файлов
Идентификатор
пользователя

Создание процесса:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
void oldman(); void recreation();
int main(){
    pid_t child_pid, parent_pid;
    int i=0;
    fprintf(stdout, "Before RECREATION %i\n",
                parent_pid=(int) getpid());
    child_pid=fork();
    while(i++<5)
        if(child_pid!=0)
            oldman();
        else
            recreation();
    return 0;
}
```

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

```
#include <unistd.h>
```

```
void oldman(){
```

```
    fprintf(stdout, "I'm not yet dead! My ID is %i\n", (int) getpid());  
}
```

```
void recreation(){
```

```
    fprintf(stdout, "Who I am? My ID is %i\n", (int) getpid());  
}
```

С точки зрения планировщика дочерний и родительский процессы независимы:

```
ewgenij@linux-g5md:~/2011-spring/  
Lect2> ./2
```

Before RECREATION 6169

I'm not yet dead! My ID is 6169

I'm not yet dead! My ID is 6169

I'm not yet dead! My ID is 6169

Who I am? My ID is 6170

I'm not yet dead! My ID is 6169

I'm not yet dead! My ID is 6169

Who I am? My ID is 6170

Who I am? My ID is 6170

Who I am? My ID is 6170

Who I am? My ID is 6170

```
ewgenij@linux-g5md:~/2011-spring/  
Lect2> ./2
```

Before RECREATION 6154

I'm not yet dead! My ID is 6154

I'm not yet dead! My ID is 6154

Who I am? My ID is 6155

Who I am? My ID is 6155

Who I am? My ID is 6155

Who I am? My ID is 6155

Who I am? My ID is 6155

I'm not yet dead! My ID is 6154

I'm not yet dead! My ID is 6154

I'm not yet dead! My ID is 6154

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main(){
    pid_t  child_pid, parent_pid;
    double s=0.0;
    child_pid=fork();
    if(child_pid!=0){
        s+=3.14;
        fprintf(stdout, "CHILD:  %i  s=%g &s=%u\n", (int) getpid(),s,&s);
    }
    else{
        s+=2.72;
        fprintf(stdout, "PARENT: %i  s=%g &s=%u\n", (int) getpid(),s, &s);
    }
    return 0;
}
```

Output:

```
PARENT: 5404 s=2.72 &s=2309295864  
CHILD:  5403 s=3.14 &s=2309295864
```

При создании процесса с помощью системного вызова **fork()** копируется адресное пространство, - переменная *s* имеет один и тот же адрес. Однако отображение на физическую память для родительского и дочернего процесса различно, - значения переменной *s* различны.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main(){
    pid_t  child_pid;
    pid_t  parent_pid;
    double s=0.0;;
    FILE* fp;

    child_pid=fork();

    fp=fopen("test.dat","a+");
```

```
if(child_pid!=0){
    s+=3.14;
    fprintf(fp, "CHILD: %i s=%g &s=%u fp=%u\n", (int) getpid(),
                                                    s, &s, fp);
}
else{
    s+=2.72;
    fprintf(fp, "PARENT: %i s=%g &s=%u fp=%u\n", (int) getpid(),
                                                    s, &s, fp);
}
fclose(fp);
return 0;
}
```

test.dat

```
PARENT: 5450 s=2.72 &s=760346688 fp=6299664
CHILD:  5449 s=3.14 &s=760346688 fp=6299664
```

Дескрипторы файлов при копировании сохраняются.