

Лекция 3

- Режим ядра и пользовательский режим.
- Системные вызовы.
- Интерфейсы прикладного программирования.
- Управление ресурсами ОС.
- Объекты ядра.
- Процессы. Их реализация и управление ими.
- Создание процессов в Linux.

Взаимодействие прикладных программ и ОС

Режим ядра (режим супервизора, привилегированный режим):

- полный доступ к командам процессора;
- обработка прерываний и исключений;
- доступ к объектам ядра.

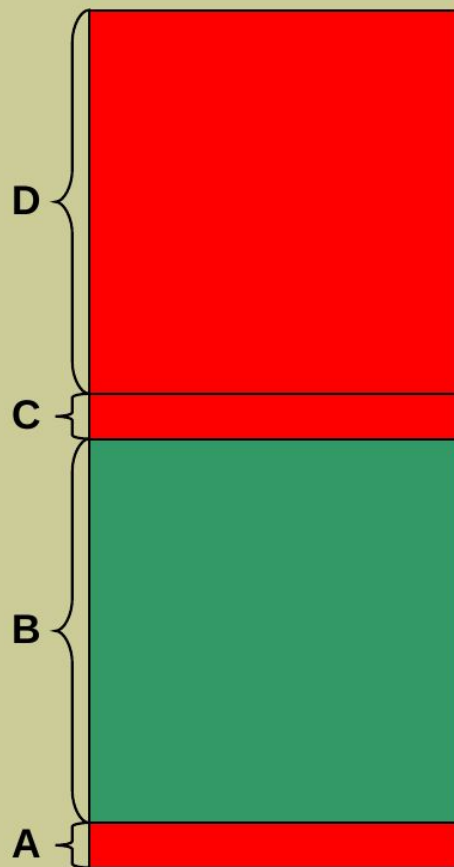
Пользовательский режим:

- ограниченный набор команд процессора;
- запрет на вызов обработчиков прерываний.

Интерфейс системных вызовов предоставляет контролируемый доступ прикладных программ к ресурсам компьютера посредством переход из пользовательского режима в режим ядра.

Интерфейс прикладного программирования - библиотечные функции.

Пример структуры адресного пространства 32-разрядной ОС.



- A. 0x00000000 – 0x0000FFFF; используется для неинициализированных указателей; **недоступно** в пользовательском режиме.
- B. 0x00010000 – 0x7FFEFFFE; адресное пространство процессов, содержит прикладные модули .exe и .dll, win32 (kernel32.dll, user32.dll и т.д.), файлы, отображаемые в память; **доступно** в пользовательском режиме.
- C. 0x7FFF0000 – 0x7FFFFFFF; используется для некорректно инициализированных указателей; **недоступно** в пользовательском режиме.
- D. 0x80000000 – 0xFFFFFFFF; зарезервировано ОС Windows для исполнительной системы, ядра и драйверов устройств; **недоступно** в пользовательском режиме.

write(hFile, pBuffer, nToWrite) – библиотечная функция

0xFFFFFFFF

Пространство
пользователя

4 Возврат к функции

2 Перехват и переход в ядро

1 Вызов write

Библиотечная
функция write

Прикладная
программа,
вызывающая
write

Ядро

3 Обработчик системного вызова

0x00000000

write(int fd, const void *buf, size_t count);

- библиотечная функция

mov edx, 1	;сколько байт записать
mov ecx, hex	;буфер, откуда писать
mov ebx, 1	;куда записывать, 1 - stdout
mov eax, 4	;номер системного вызова
int 80h	;шлюз к ядру

Таблица системных вызовов

%eax	Name	Source	%ebx	%ecx	%edx
1	sys_exit	kernel/exit.c	int	-	-
2	sys_fork	arch/i386/kernel/process.c	struct pt_regs	-	-
3	sys_read	fs/read_write.c	unsigned int	char*	size_t
4	sys_write	fs/read_write.c	unsigned int	const char*	size_t
5	sys_open	fs/open.c	const char*	int	int
6	sys_close	fs/open.c	unsigned int	-	-

Управление ресурсами ВС:

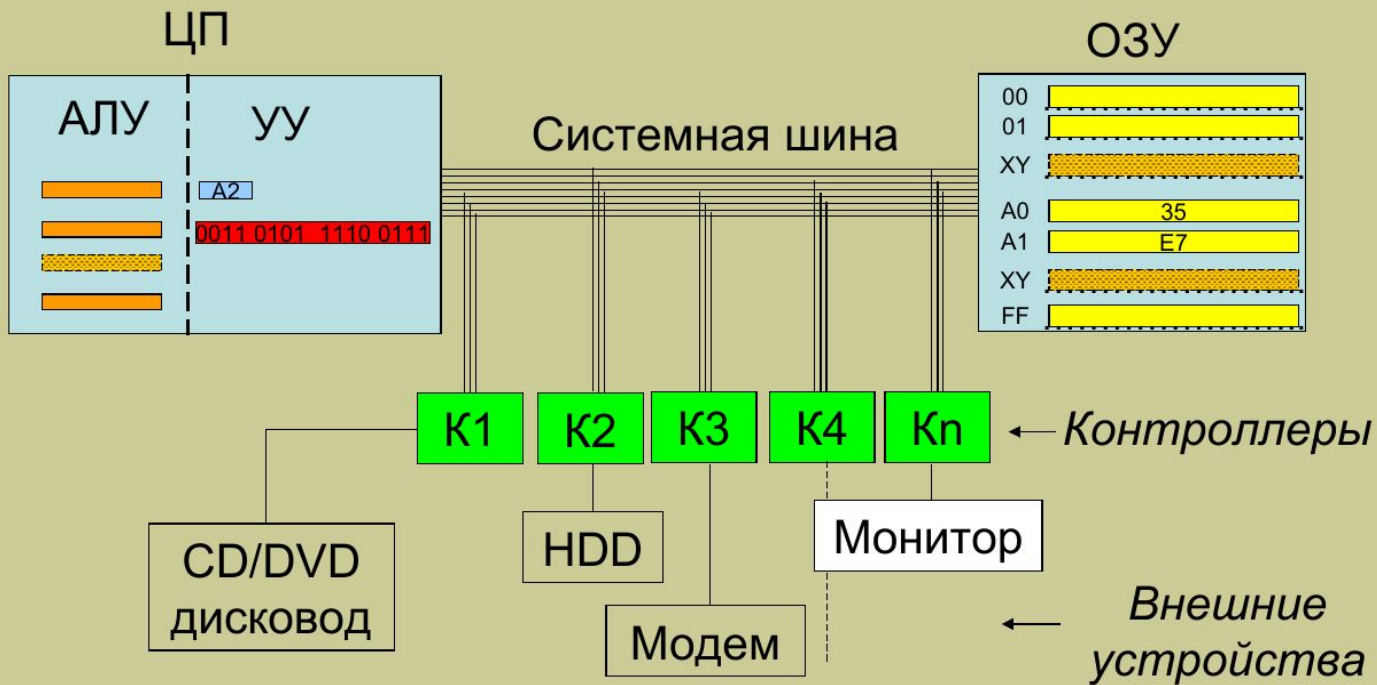
Программное обеспечение ввода-вывода уровня пользователя

Устройство-независимое программное обеспечение операционной системы

Драйверы устройств

Обработчики прерываний

Аппаратура



ЦПУ

Контроллер ввода/вывода

Драйвер устройства инициирует I/O

ЦПУ выполняет проверки на прерывания
между инструкциями

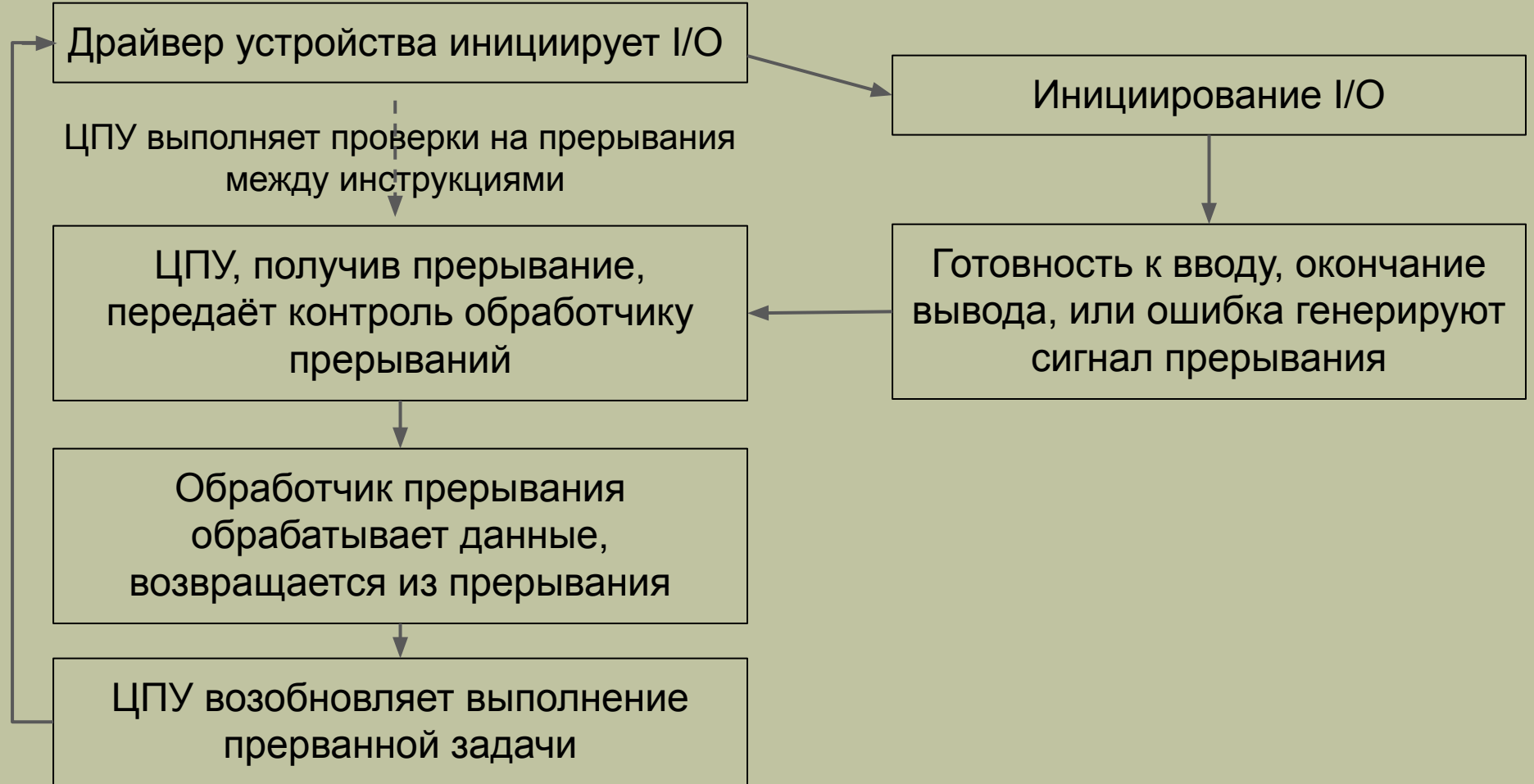
ЦПУ, получив прерывание,
передаёт контроль обработчику
прерываний

Обработчик прерывания
обрабатывает данные,
возвращается из прерывания

ЦПУ возобновляет выполнение
прерванной задачи

Инициирование I/O

Готовность к вводу, окончание
вывода, или ошибка генерируют
сигнал прерывания



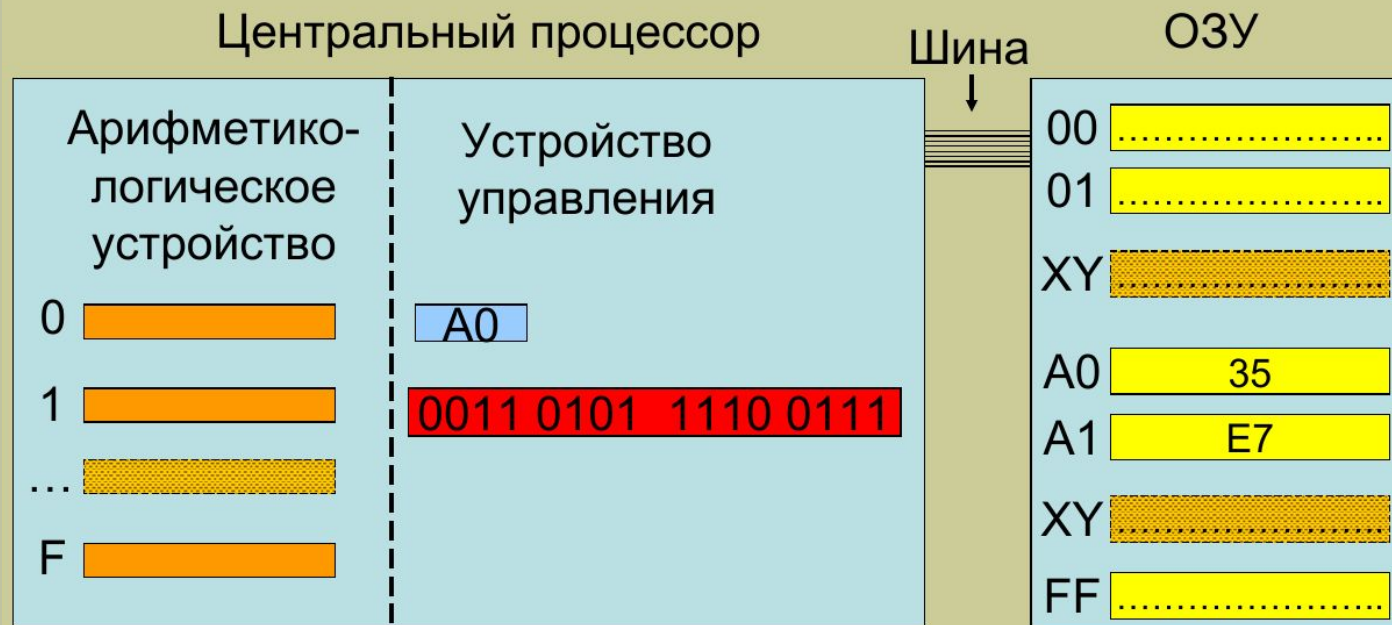
Объекты ядра операционной системы:

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

...

Процессы

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



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

Счетчик команд

Ячейки памяти

Регистр команд

```

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG
D:\>debug
-a
06B0:0100 mov ax, 98
06B0:0103 mov bl, 15
06B0:0105 mov bh, 10
06B0:0107 mov cx, 90
06B0:010A
-d
06B0:0100 BB 98 00 B3 15 B7 10 B9-90 00 00 00 00 00 00 .....
06B0:0110 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
06B0:0120 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
06B0:0130 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
06B0:0140 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
06B0:0150 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
06B0:0160 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
06B0:0170 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
-t
AX=0098 BX=0000 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=06B0 ES=06B0 SS=06B0 CS=06B0 IP=0103 NU UP EI PL NZ NA PO NC
06B0:0103 B315 MOV BL,15
-t
AX=0098 BX=0015 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=06B0 ES=06B0 SS=06B0 CS=06B0 IP=0105 NU UP EI PL NZ NA PO NC
06B0:0105 B710 MOV BH,10
-

```

```

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG
D:\>debug
-a
06B0:0100 mov ah, 10
06B0:0102 mov al, 15
06B0:0104 mov bx, 78
06B0:0107
-d
06B0:0100 B4 10 B0 15 BB 78 00 00-00 00 00 00 00 00 .....X.....
06B0:0110 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
06B0:0120 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
06B0:0130 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
06B0:0140 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
06B0:0150 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
06B0:0160 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
06B0:0170 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
-t
AX=1000 BX=0000 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=06B0 ES=06B0 SS=06B0 CS=06B0 IP=0102 NU UP EI PL NZ NA PO NC
06B0:0102 B015 MOV AL,15
-t
AX=1015 BX=0000 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=06B0 ES=06B0 SS=06B0 CS=06B0 IP=0104 NU UP EI PL NZ NA PO NC
06B0:0104 BB7800 MOV BX,0078
-

```

```

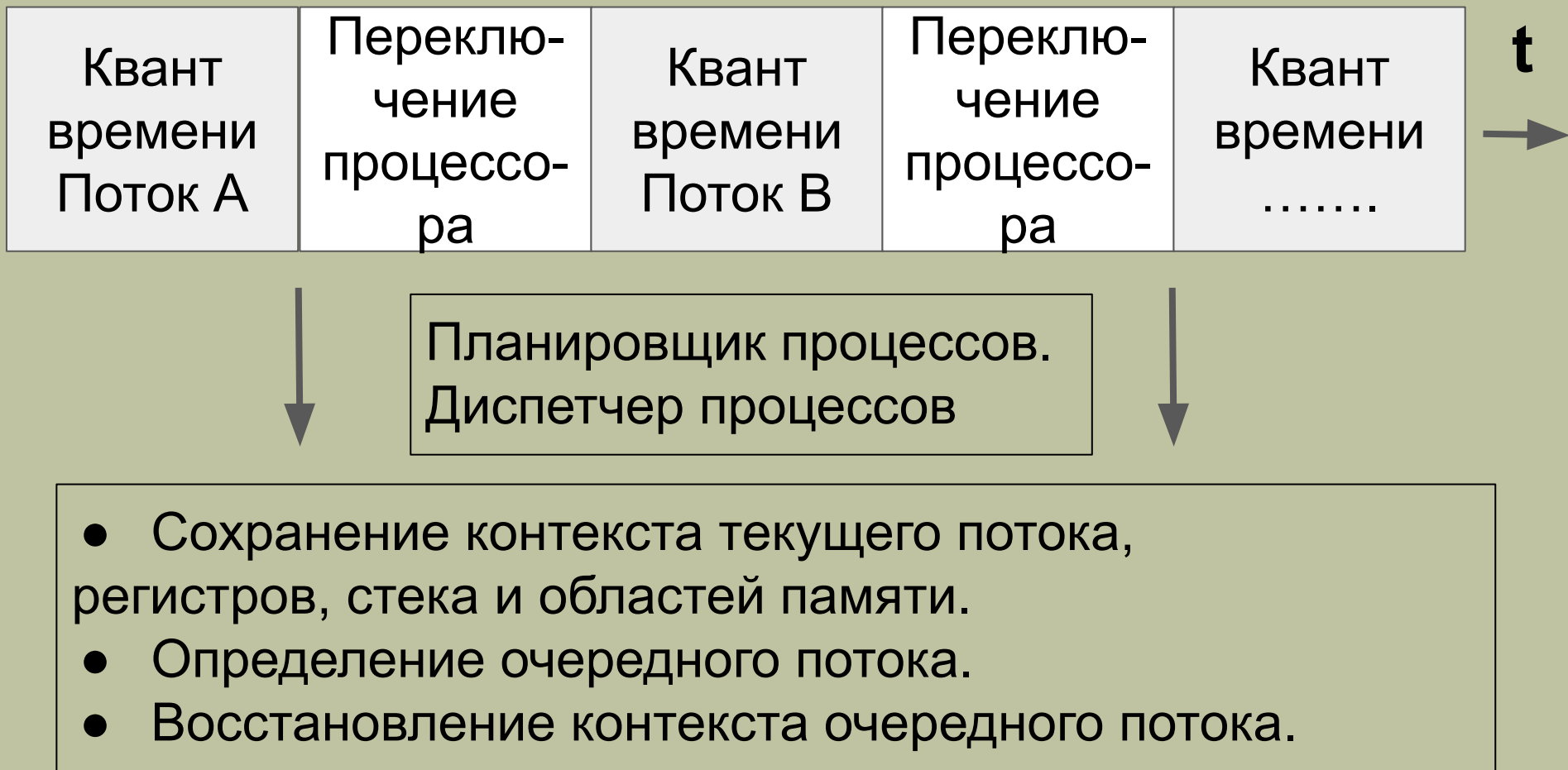
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG
D:\>debug
-a
06B0:0100 mov ax, 13
06B0:0103 mov bx, 10
06B0:0106 add bx, 5
06B0:0109
-d
06B0:0100 BB 13 00 BB 10 00 83 C3-05 00 00 00 00 00 00 .....
06B0:0110 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
06B0:0120 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
06B0:0130 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
06B0:0140 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
06B0:0150 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
06B0:0160 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
06B0:0170 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
-t
AX=0013 BX=0000 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=06B0 ES=06B0 SS=06B0 CS=06B0 IP=0103 NU UP EI PL NZ NA PO NC
06B0:0103 BB1000 MOV BX,0010
-t
AX=0013 BX=0010 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=06B0 ES=06B0 SS=06B0 CS=06B0 IP=0106 NU UP EI PL NZ NA PO NC
06B0:0106 83C05 ADD BX,+05
-

```

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 и т.д.;
- уникальный идентификационный номер, называемый идентификатором процесса;
- потоки (по крайней мере, один поток).

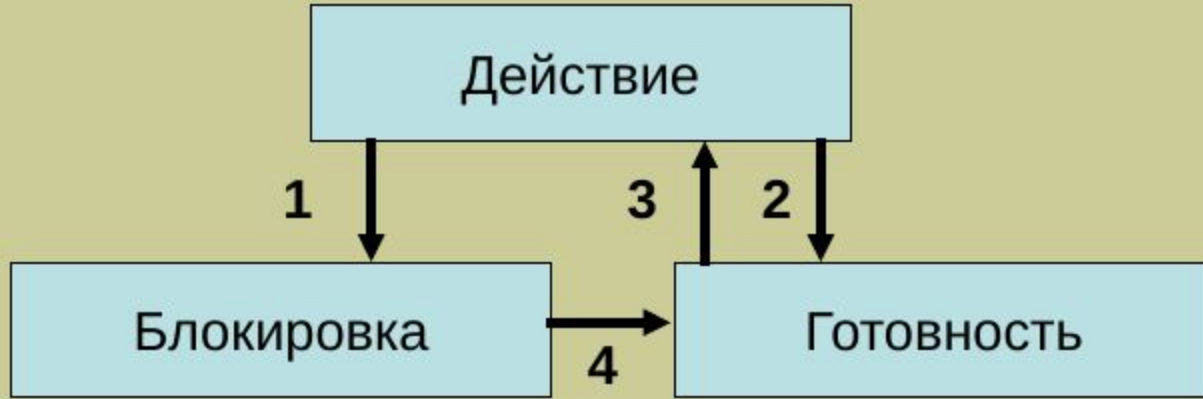
Модель процесса:



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**, программно реализованная, как список структур) (***Process Control Block***).

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());
```

```
}
```

~> ./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

~> ./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;  
}
```

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.