ДИСЦИПЛИНА	Операционные системы
	(полное наименование дисциплины без сокращений)
ИНСТИТУТ	Институт информационных технологий
КАФЕДРА	информационных технологий в атомной энергетике
	(полное наименование кафедры)
ВИД УЧЕБНОГО МАТЕРИАЛА	Лекция
	(в соответствии с пп 1-11)
ПРЕПОДАВАТЕЛЬ	Пугачев Андрей Васильевич
	(фамилия, имя, отчество)
СЕМЕСТР	IV семестр 2024 – 2025 учебный год
	(указать семестр обучения, учебный год)

Тема № 2: ”Процессы и потоки”

«Операционные системы»

МИРЭА – Российский технологический университет

Москва. 2024-2025 у.г.

Определение

Программа - файл, хранящийся в памяти вычислительной системы и предназначенная для выполнения.

Процесс — программа на стадии выполнения.

Процесс — логический объект, обладающий собственным адресным пространством.

Определение

Адресное пространство — это набор адресов оперативной памяти, доступных процессу для чтения, записи, выполнения.

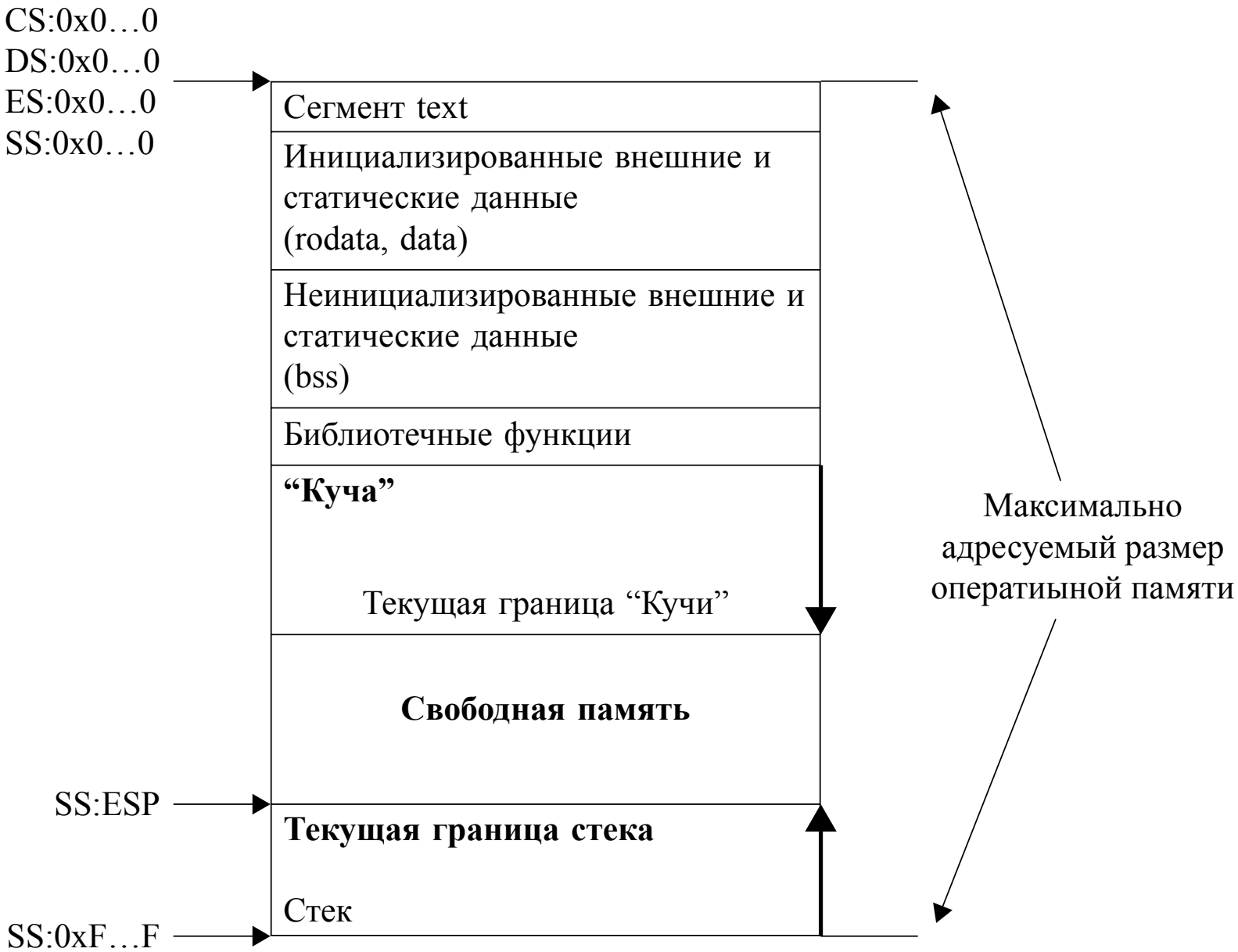
Структура адресного пространства

- Область кода (text).
- Область данных:
  - Область инициализированных данных (data).
  - Область неинициализированных данных (bss).
  - Область данных, доступных только для чтения (rodata).
- Область стека (stack).
- “Куча” (heap).

Модели памяти процесса

- tiny;
- small;
- medium;
- compact;
- large;
- huge;
- flat.

Модели памяти FLAT



Модели памяти FLAT (Демонстрация)

Исходный код

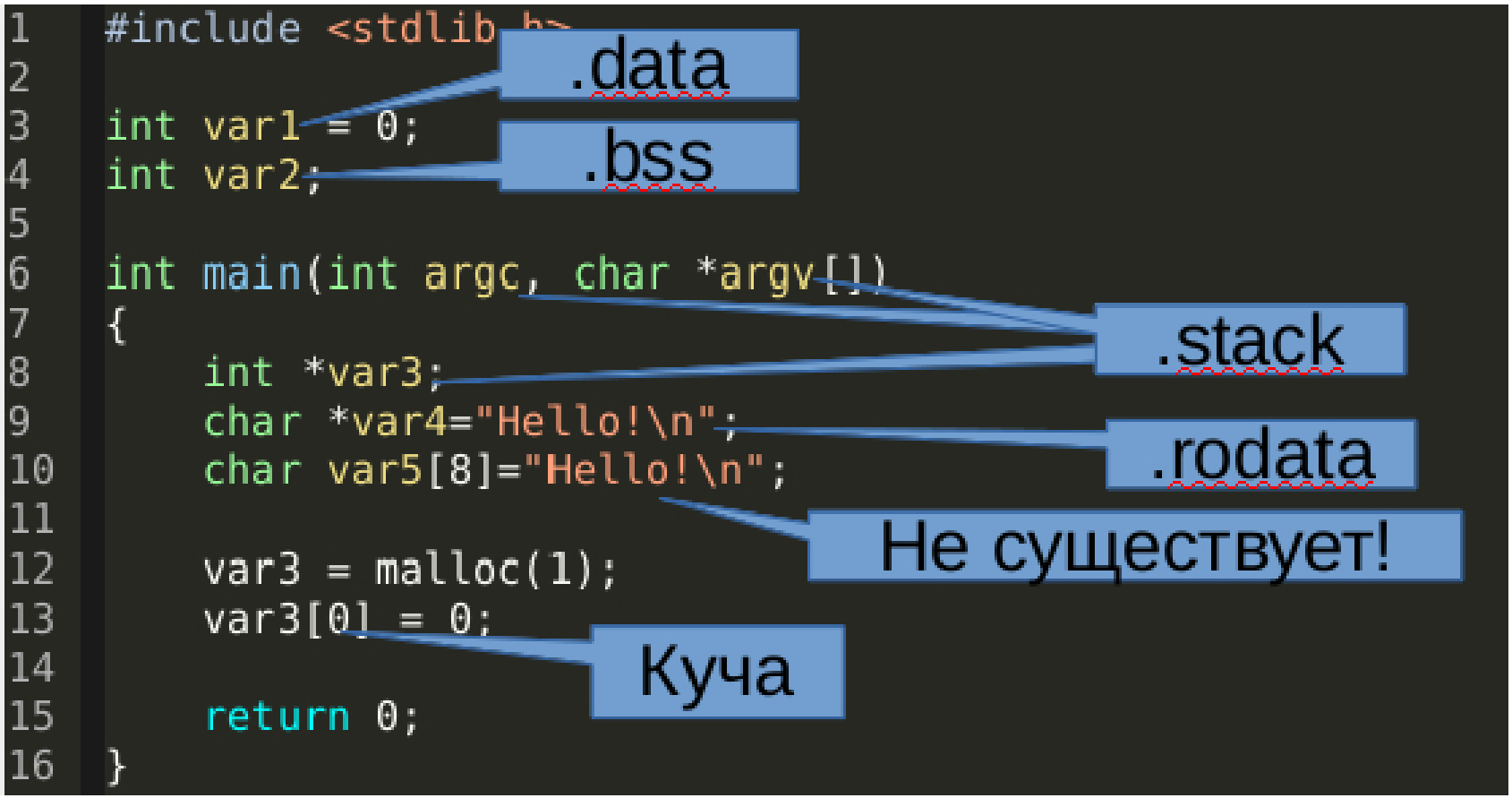
```
#include <stdlib.h>
#include <stdio.h>

int main(int argc,char* argv)
{
    char *a;
    a = malloc(1);
    printf("stack:□0x%08X□(%d)\n",&a,&a);
    printf("head:□□0x%08X□(%d)\n",a, a);
    free(a);
    return 0;
}
```

Результат

stack: 0xFFBF98CC (-4220724)

head: 0x0804B008 (134524936)

Модели памяти FLAT (Демонстрация)

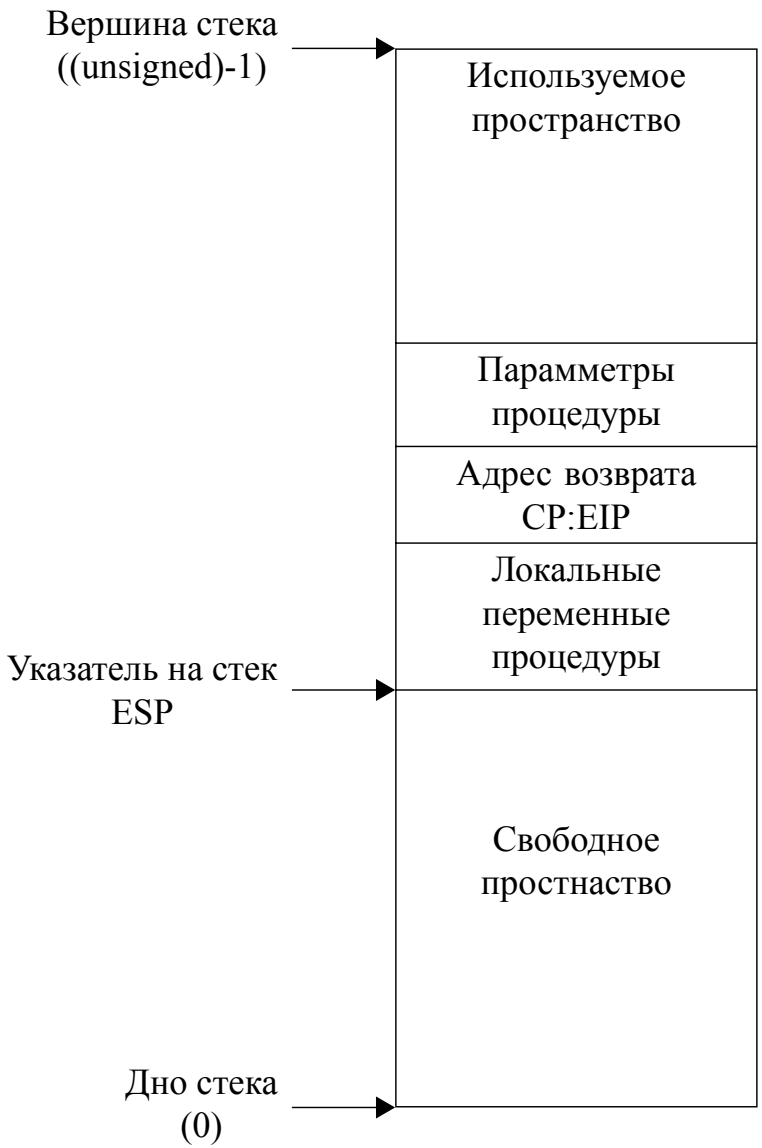


Область стека

- временное сохранение данных;
- хранение адреса возврата из процедуры;
- передача параметров при вызове процедур [1];
- выделение локальных переменных.

[1] для процедур типа cdecl, pascal, stdcall

Область стека



Нестандартное использование стека

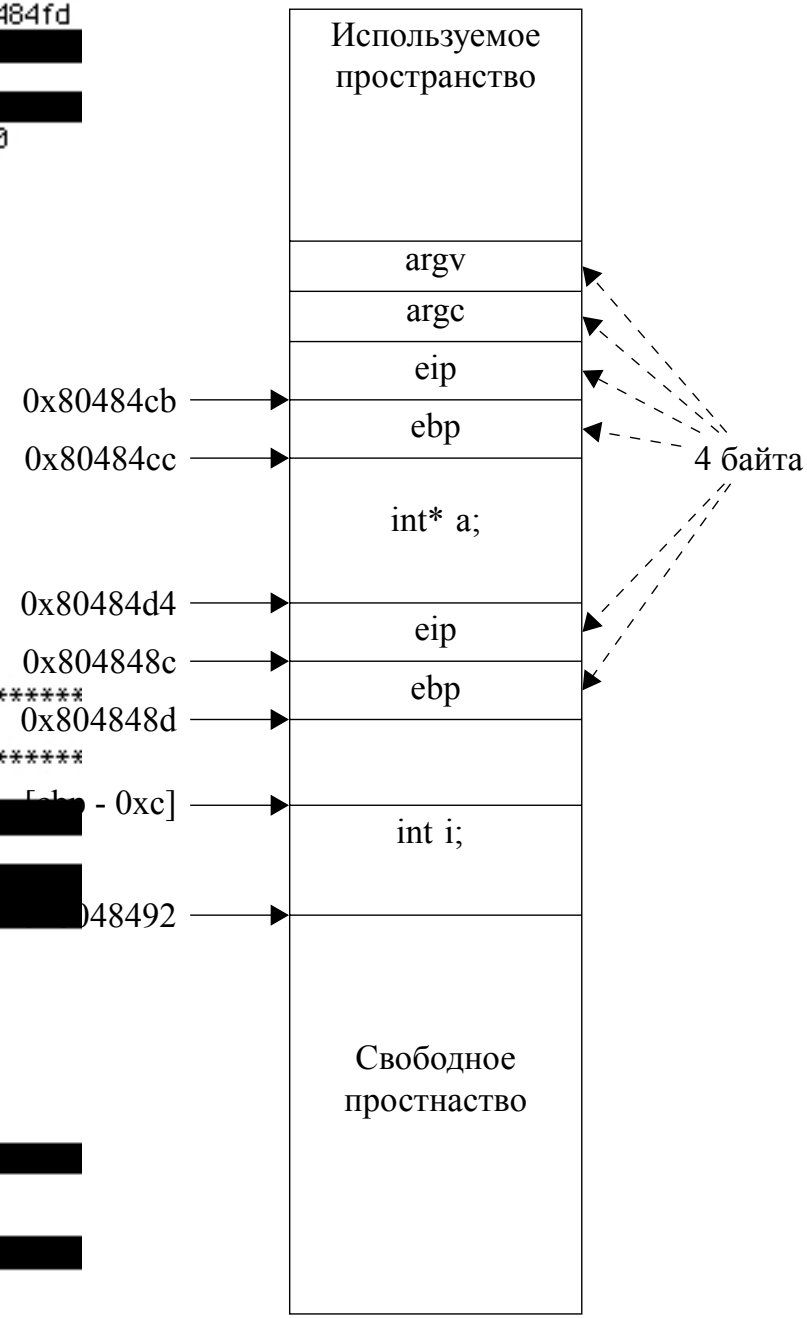
Исходный код

```
#include <stdlib.h>
#include <stdio.h>

void func(void) {
    int i;
    printf("Print:□");
    for( ; i > 0 ; i-- ) printf("%d□", i);
    printf("Done\n");
}
int main(int argc,char* argv)
{
    int *a;
    a = (int*)((((int)(&argc) - 8) & (-0x10)) - 0x24);
    *a=10; func();
    *a=20; func();
    return 0;
}
```

Результат

```
 /tmp $ ./a.out
Print: 10 9 8 7 6 5 4 3 2 1 Done
Print: 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 Done
```

```
....... ! ;*******************************************************
....... ! ; function func (global)
....... ! ;*******************************************************
804848c ! func:                                ;xref c80484ee c80484fd
....... !   push        ebp
804848d !   mov         ebp, esp
804848f !   sub         esp, 28h
8048492 !   mov         dword ptr [esp], strz_Print:__80485c0
8048499 !   call        printf
804849e !   jmp         for
80484a0 !
....... ! for_body:                             ;xref j80484bb
....... !   mov         eax, [ebp-0ch]
80484a3 !   mov         [esp+4], eax
80484a7 !   mov         dword ptr [esp], data_80485c8
80484ae !   call        printf
80484b3 !   sub         dword ptr [ebp-0ch], 1
80484b7 !
....... ! for:                                  ;xref j804849e
....... !   cmp         dword ptr [ebp-0ch], 0
80484bb !   jg          for_body
80484bd !   mov         dword ptr [esp], strz_Done_80485cc
80484c4 !   call        puts
80484c9 !   leave
80484ca !   ret
....... ! ;*******************************************************
....... ! ; function main (global)
....... ! ;*******************************************************
80484cb ! main:                                ;xref o8048397
....... !   push        ebp
80484cc !   mov         ebp, esp
80484ce !   and         esp, 0fffffff0h
80484d1 !   sub         esp, 10h
80484d4 !   lea         eax, [ebp+8]
80484d7 !   sub         eax, 8
80484da !   and         eax, 0fffffff0h
80484dd !   sub         eax, 24h
80484e0 !   mov         [esp+0ch], eax
80484e4 !   mov         eax, [esp+0ch]
80484e8 !   mov         dword ptr [eax], 0ah
80484ee !   call        func
80484f3 !   mov         eax, [esp+0ch]
80484f7 !   mov         dword ptr [eax], 14h
80484fd !   call        func
8048502 !   mov         eax, 0
8048507 !   leave
8048508 !   ret
```



Многозадачность

Многозадачность

Определение

Многозадачность - технология, позволяющая выполнять несколько программ параллельно (псевдопараллельно).

Виды

- невытесняющая (кооперативная) многозадачность;
- вытесняющая многозадачность.

Контекст процесса

- Аппаратный уровень:
  - состоянии регистров;
  - маска доступа портов ввода-вывода.
- Уровень ОС:
  - состояние процесса;
  - используемые ресурсы и блокировки;
  - карта памяти;
  - права доступа;
  - т.д.

Потоки

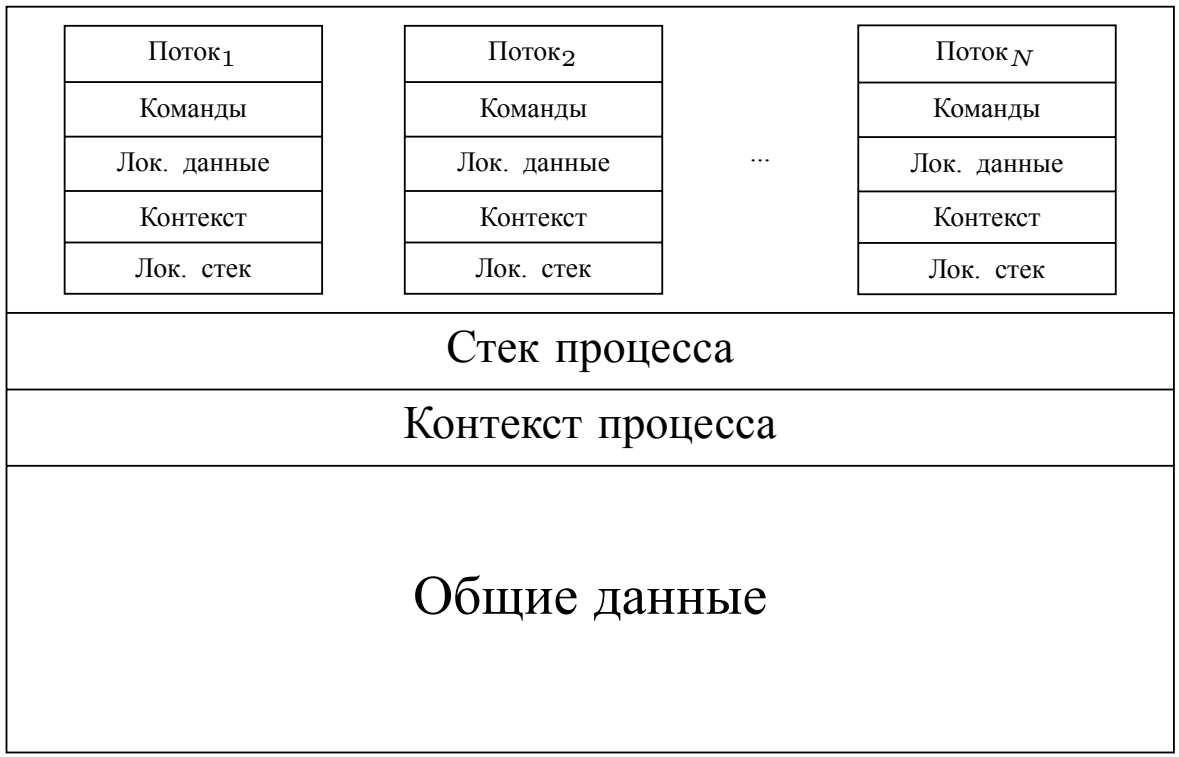
Определение

Многопоточность - технология, позволяющая формировать программу из фрагментов кода (набора команд), которые могут выполняется одновременно(параллельно).

Определение

Поток - фрагмент кода (набор команд) процесса, которые может быть выполнен одновременно (параллельно) с остальным набором команд процесса.

Общая структура процесса



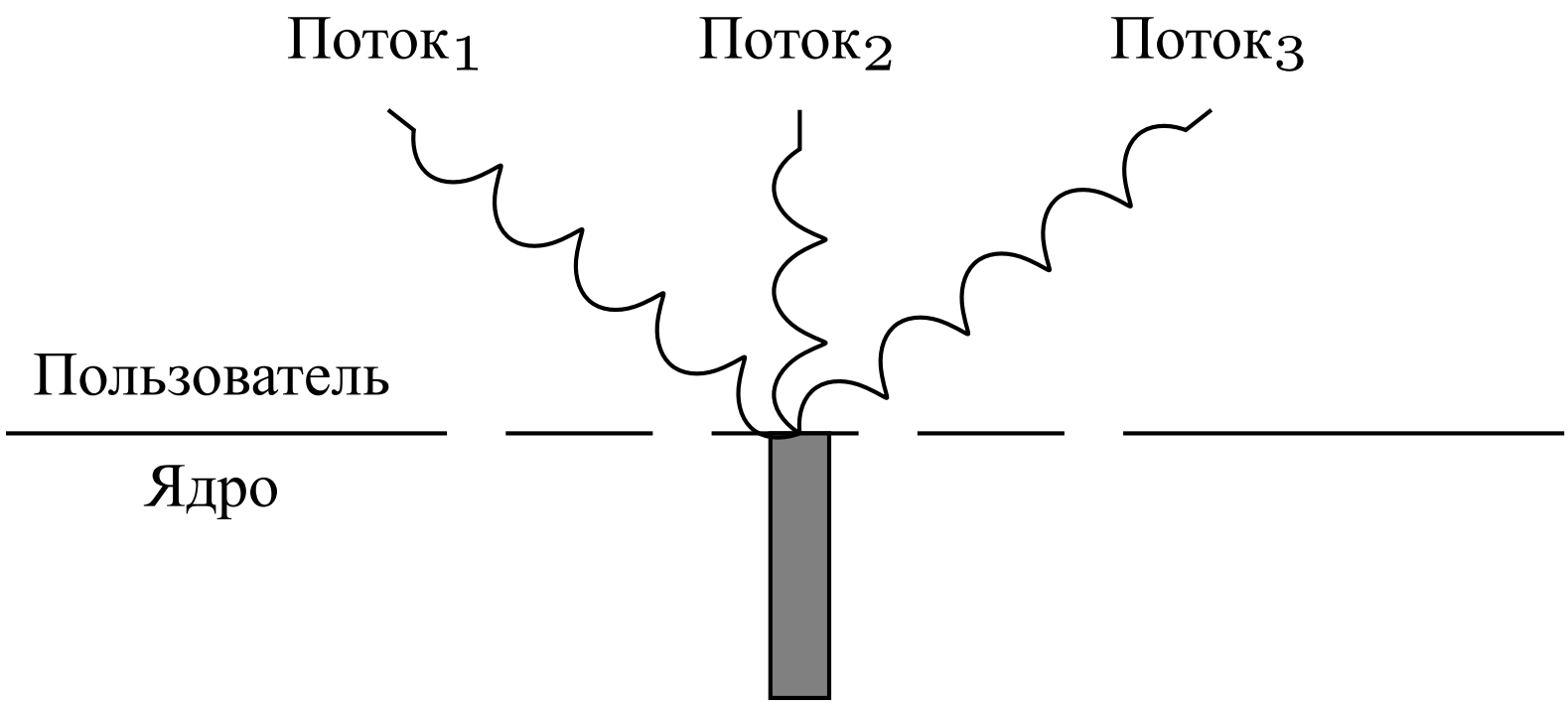
Модели потоков

- поток уровня пользователя;
- поток уровня ядра;
- гибридная модель потока.

Модель потока уровня пользователя

Особенности

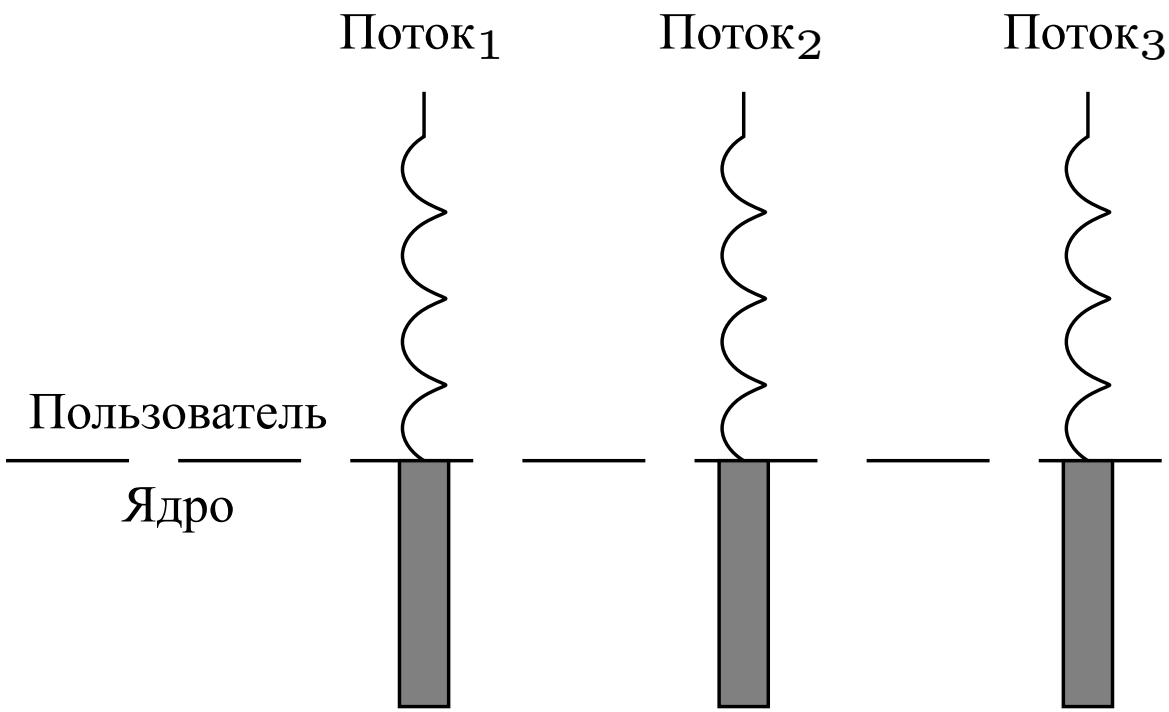
+ скорость работы;
+ гибкость планирования;
– система выделяет такое же количество квантов времени на все потоки как и на один процесс.



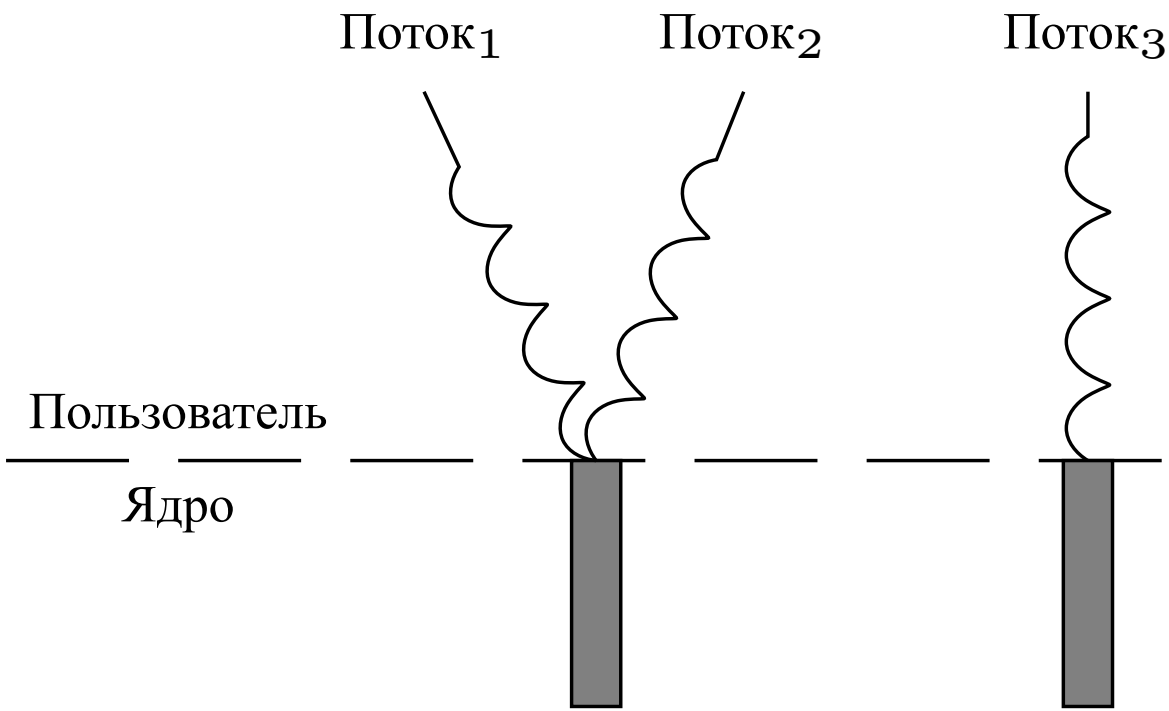
Модель потока уровня ядра

Особенности

+ распределение времени;
+ упрощение пользовательской реализации;
– менее гибкое планирование;
– дополнительные накладные расходы для ОС.

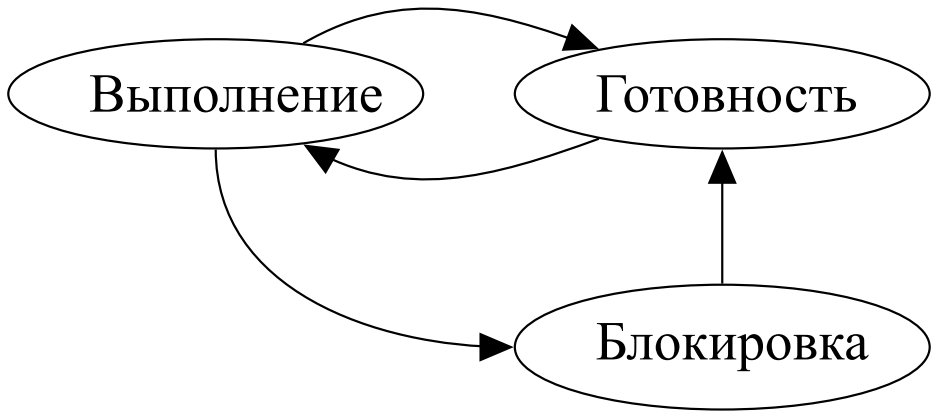


Гибридная модель потока

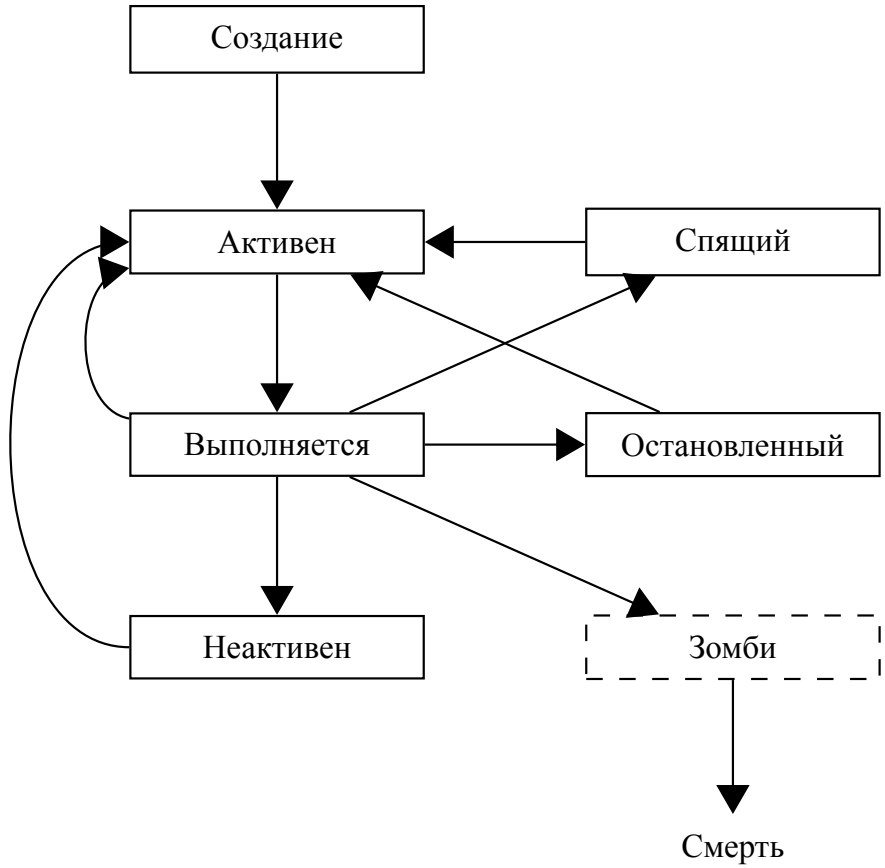


Жизненный цикл процесса/потока

Общая модель



Жизненный цикл в Linux



IPC

IPC (механизмов межпроцессного взаимодействия) – механизмы, применяемые для организации взаимодействия процсов.

Цули использования IPC

1. Передача данных (в т.ч. оповещение о событиях).
2. Синхронизация доступа процессов к ресурсам (взаимоисключения).

Межпроцессное взаимодействие (IPC)

- файл;
- каналы;
- сигналы;
- сокет;
- разделяемая память;
- сообщения;
- почтовый ящик.

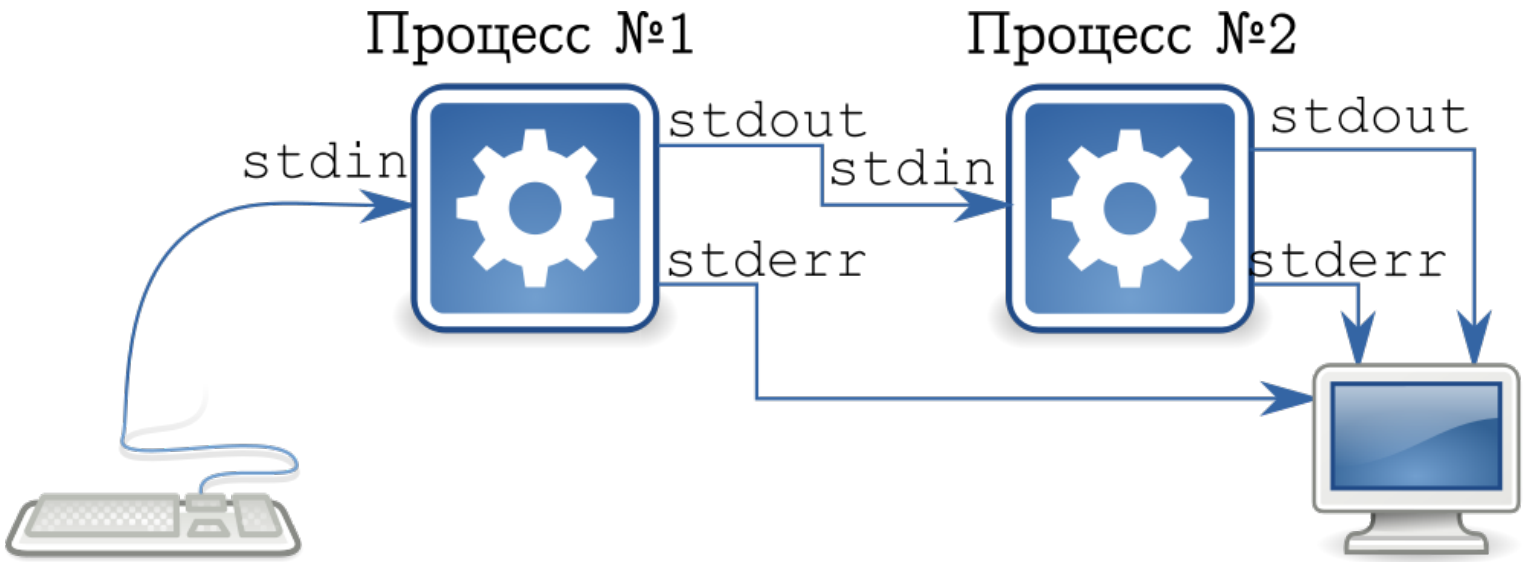
Каналы

Канал — механизм межпроцессного взаимодействия, формирующий прямой поток даных между двумя процессами.

Виды каналов

- именованные;
- неименованные.

Конвейер


Процесс №1
Процесс №2
stdin
stdout
stdin
stdout
stderr
stderr

Сигналы

Сигнал – вырабатываемое программным обеспечением сообщение, информирующее о наступлении определенного события либо возникновении ошибки. Сигналы не позволяют передавать какие-либо данные.

Виды сигналов

- перехватываемые;
- неперехватываемые.

- сокет;
- разделяемая память;
- сообщения;
- почтовый ящик.

Вопросы?