Факультатив по программированию на языке C

Занятие 9
Основы устройства ОС

План занятий

№	Тема	Описание
1	Введение в курс	Языки программирования. Основы работы с Linux.
2	Основы языка С	Написание и компиляция простейших программ с использованием gcc. Правила написания кода.
3	Компиляция	Разбиение программы на отдельные файлы. Make файлы. Компиляция.
4	Ввод данных. Библиотеки	Работа со вводом/выводом. Статические и динамические библиотеки.
5	Хранение данных. Память	Хранение процесса в памяти компьютера. Виртуальная память, сегментация. Секции программы.
6	Устройство памяти.	Elf файлы. Указатели и массивы. Типы данных. Gdb и отладка
7	Аллокация памяти	Аллокация памяти. Битовые операции – сдвиги, логические операции. Битовые поля. Перечисления. Static переменные. Inline функции.
8	Язык ассемблера	Язык ассемблера. Вызов функции. Безопасные функции. Макросы
9	Основы работы ОС	Изучение основ загрузки ОС
10	Архиватор. Распознание голосов птиц	Программирование архиватора. Основы биоакустики

Дерево языка

Основные определения из курса ОС

Процесс – программа во время исполнения и все её элементы: адресное пространство, глобальные переменные , регистры, стек, счетчик команд, состояние, открытые файлы, дочерние процессы и т. д

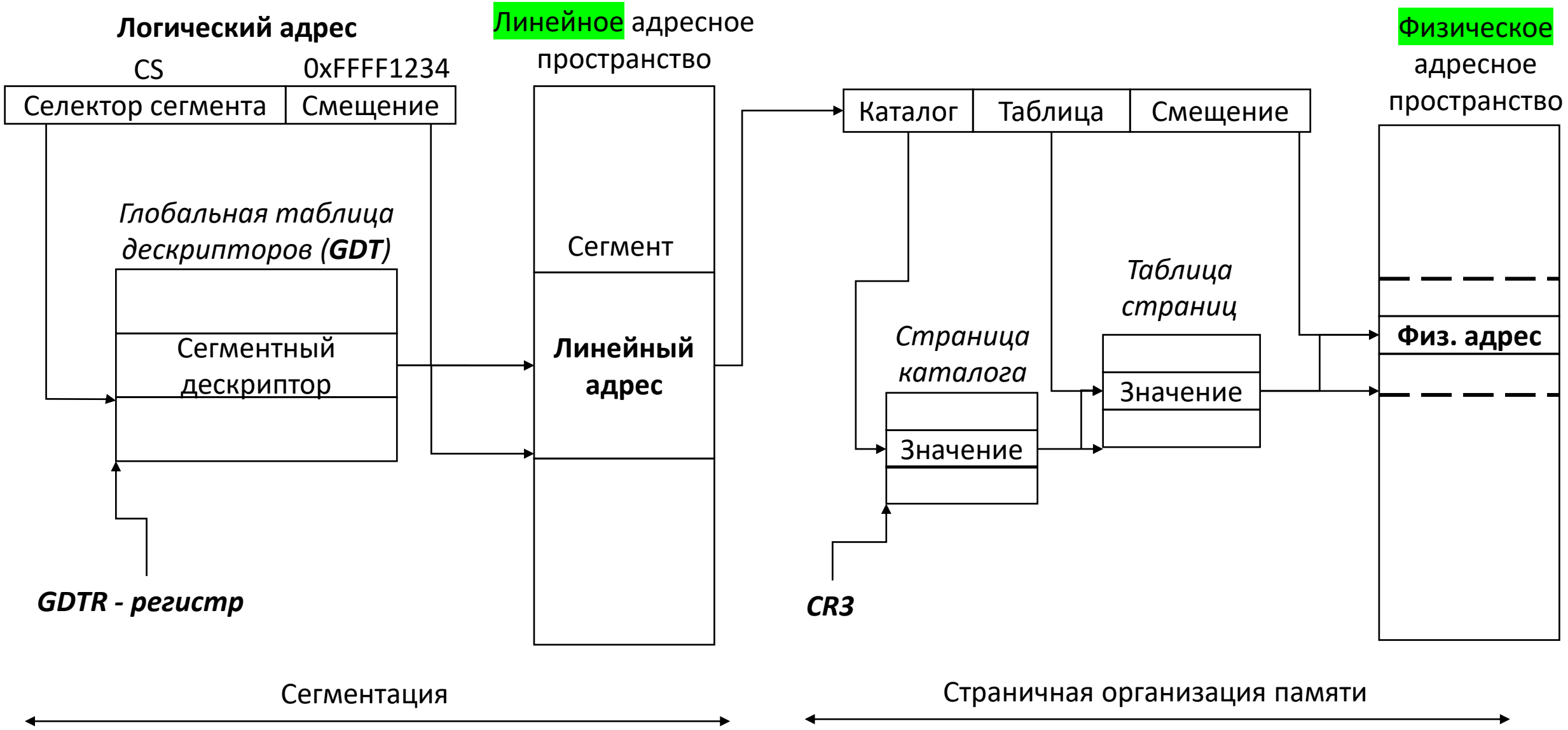
Поток - самостоятельная цепочка последовательно выполняемых операторов программы, соответствующих некоторой подзадаче

Прерывание – событие, при котором меняется последовательность команд, выполняемых процессором

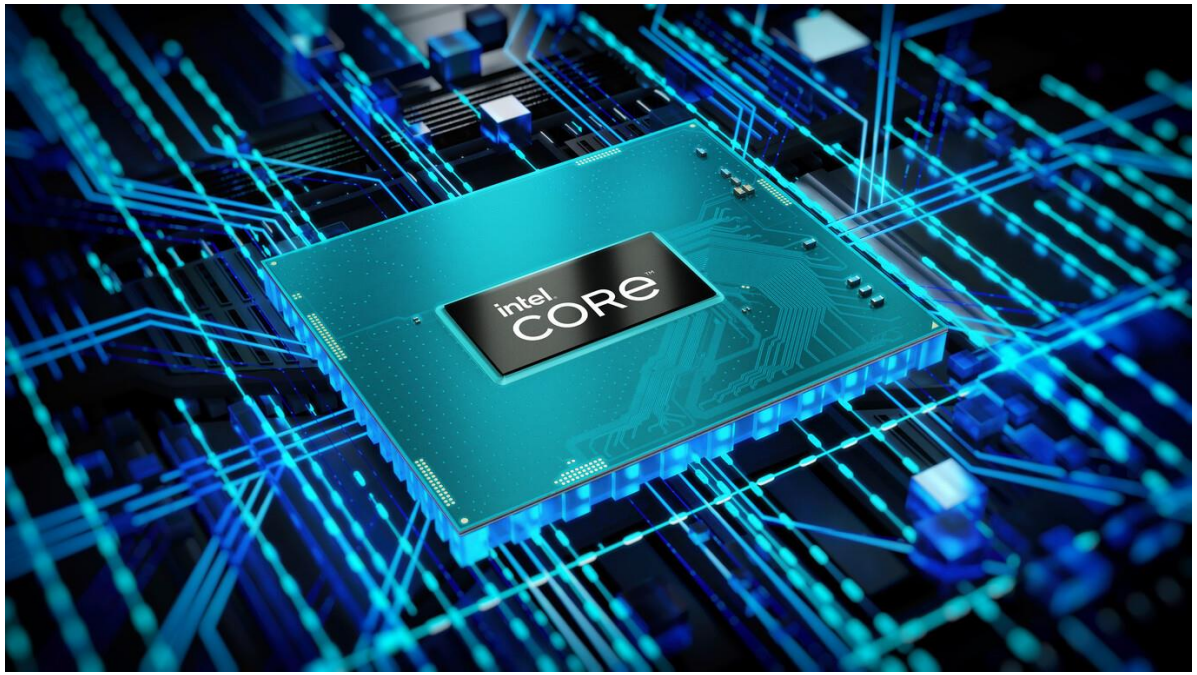
Системный вызов – это интерфейс для получения услуг операционной системы

Файл – поименованная совокупность данных

Устройство памяти (x86, 32 bit)



Некоторые оговорки



UNIX

- Архитектура – x86 (Intel)
- Пример OS – xv6 (за основу взят UNIX)

Операционная система xv6

```
init: starting sh
$ ls
.               1 1 512
..              1 1 512
README          2 2 2286
cat             2 3 16296
copy            2 4 15064
echo            2 5 15152
forktest        2 6 9460
grep            2 7 18516
init            2 8 15736
kill            2 9 15184
ln              2 10 15036
ls              2 11 17664
mkdir           2 12 15280
memalloc        2 13 15580
rm              2 14 15256
sh              2 15 27900
stressfs        2 16 16172
sleep           2 17 14916
usertests       2 18 67276
wc              2 19 17036
zombie          2 20 14848
console         3 21 0
```

xv6 — современная реализация 6-й версии операционной системы UNIX для архитектуры x86, написанная на ANSI C. Она используется в учебных целях в MIT в курсе проектирования операционных систем (Operating Systems Engineering (6.828)).

https://github.com/mit-pdos/xv6-public

Intel 8088

Технические характеристики:

- Дата анонса: 1 июля 1979 года;
- Тактовая частота, МГц: 5,8,10
- **Разрядность регистров**: 16 бит;
- Разрядность шины данных: 8 бит;
- **Разрядность шины адреса**: 20 бит;
- Объём адресуемой памяти: 1 Мбайт;
- Техпроцесс: 3 мкм;

Адресация в Intel 8088

Перенос игнорируется!

Intel 80286

Технические характеристики:

- Дата анонса: 1 февраля 1982 года;
- Тактовая частота, МГц: -6, 8, 10, 12,5;
- **Разрядность регистров**: 16 бит;
- Разрядность шины данных: 16 бит;
- **Разрядность шины адреса**: 24 бит;
- Объём адресуемой памяти: 16 Мбайт;
- Объём виртуальной памяти: 1 Гбайт;
- Техпроцесс (нм): 1500 (1,5 мкм);

Real mode / Protected mode

Real mode	Protected mode
• В этом режиме процессор работает как 8088/8086. • Возможна адресация только 1 МБ памяти. • Обработка одной задачи • В этой памяти трансляция адресов не требуется. • Процессор напрямую взаимодействуют с портами и устройствами.	• При этом процессор работает на полную мощность. • Этот режим имеет возможность адресации памяти от 1 МБ до нескольких ГБ. • Обработка нескольких задач одновременно. • В этой памяти требуется трансляция адреса. • Процессор общается с портами и устройствами **через ОС**.

Еще немного о регистрах

Регистр	Назначение
%eax	хранение результатов промежуточных вычислений
%ebx	хранения адреса (указателя) на некоторый объект в памяти
%ecx	счетчик
%edx	хранения результатов промежуточных вычислений
%esp	содержит адрес вершины стека
%ebp	указатель базы кадра стека
%esi	индекс источника
%edi	индекс приёмника

Регистр	Назначение
%cs	Сегмент кода
%ds	Сегмент данных
%es	Extra сегмент
%ss	Сегмент стека
%fs	Может использоваться для библиотек
%gs	Различные операции

Регистр	Назначение
%eip	счетчик инструкций

Еще немного о регистрах

Регистр	**Назначение**
%cr0	Основная настройка процессора
%cr1	Зарезервировано
%cr2	Содержит линейные адреса ошибки страницы
%cr3	Поддержка виртуальной памяти (paging) Хранит **физический адрес** каталога страниц
%cr4	Определенные настройки
%cr5-7	Зарезервировано

Порядок загрузки

1) Запуск первой инструкции при подаче питания
2) Отключение прерываний
3) Включение линии Gate-A20
4) Переключение в защищенный режим
5) Настройка GDT
6) Чтение из первого сектора диска по указанному адресу
7) Запуск ОС

bootasm.S, bootmain.c

Дальнейшая работа ОС

1) Инициализация paging
2) Настройка прерываний
3) Настройка периферии
4) Создание первого процесса

entry.S, main.c

spinlock

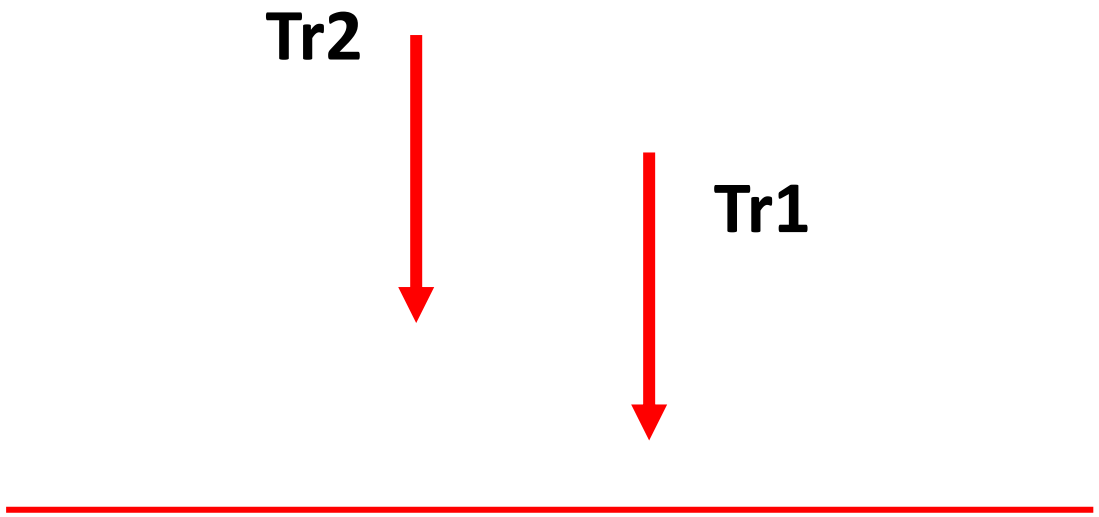
Смотрим код!

spinlock.cpp

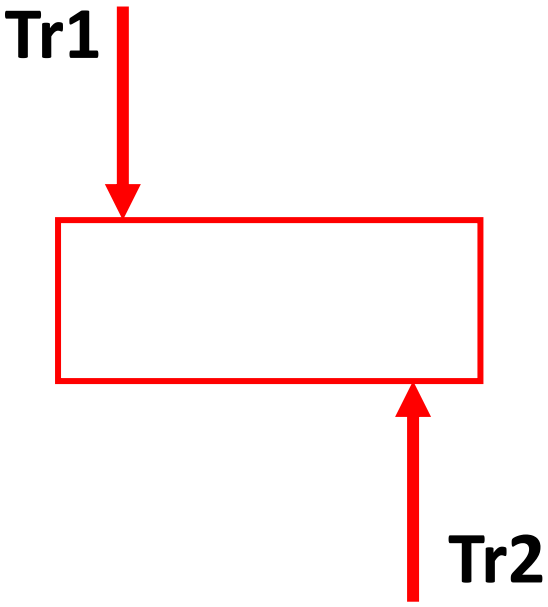
Какое значение globalvar после завершения программы?

spinlock

Race Conditions



Data Races



spinlock

Race Conditions

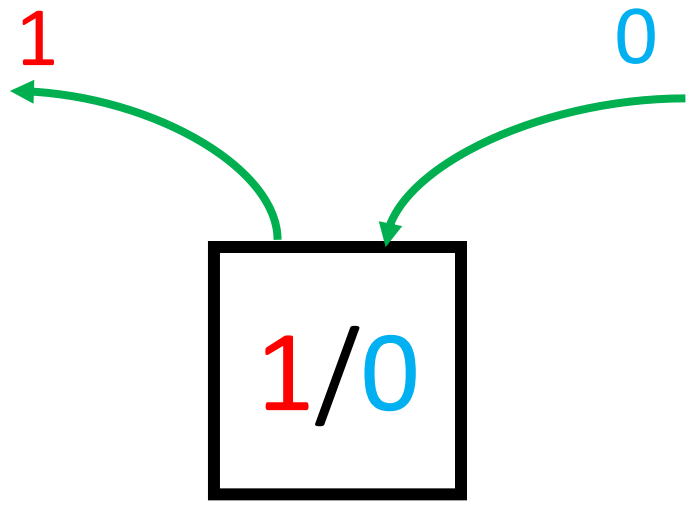
```
std::thread thr_1([]() {
        std::cout << "Thread 1\n";
        });

std::thread thr_2([]() {
        std::cout << "Thread 2\n";
        });
```

Data Races

```
std::thread thr_1([]() {
        globalvar = 15;
        });
std::thread thr_2([]() {
        globalvar = 25;
        });
```

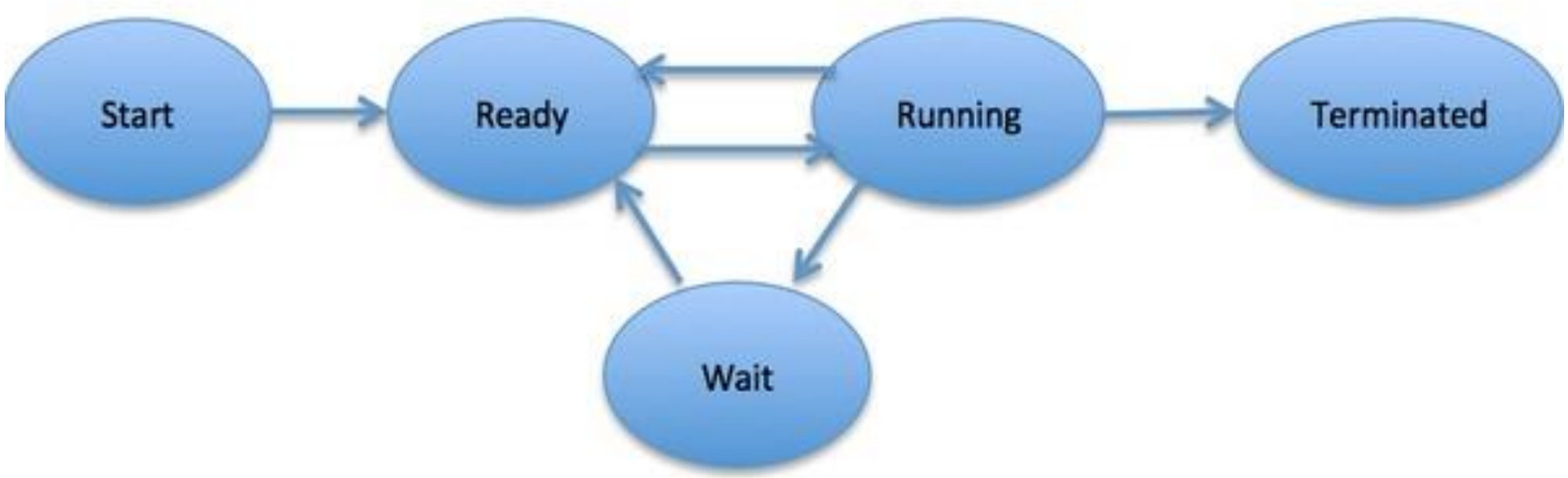
spinlock

```
class Spinlock
{
    private:
        std::atomic<bool> locked_{ false };
    public:
        void lock()
        {
                while (locked_.exchange(true)) {};
        }
        void unlock()
        {
                locked_.store(false);
        }
};
```



Возвращаем замещенное значение

Состояния процесса



Создание процесса

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

int main()
    {
    pid_t PID = fork();
    if (PID == 0)
    {
        printf("Hello ");
    }
    else
    {
        wait(0);
        printf("world!\n");
    }
    return 0;
}
```

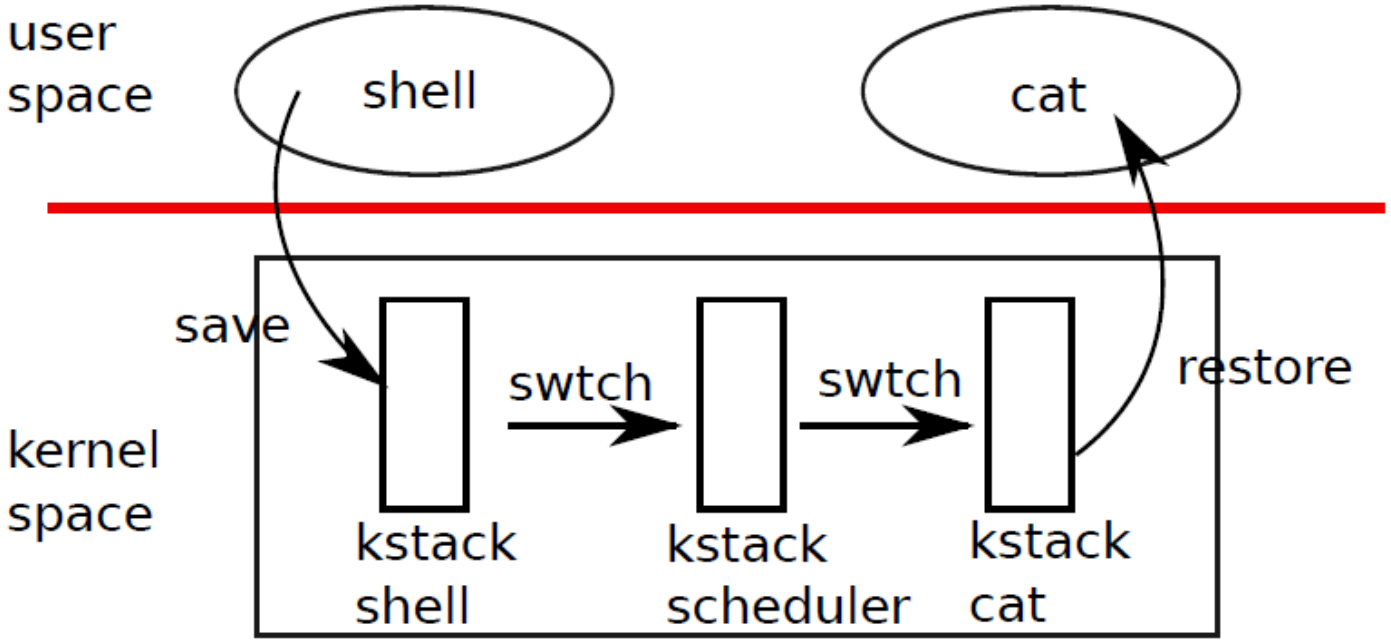
Системный вызов fork() создает точную копию родительского процесса.

При успешном создании нового процесса в **родительский** процесс возвращается PID дочернего, а в **дочерний** процесс возвращается – 0

fork() - характерен только для Linux!

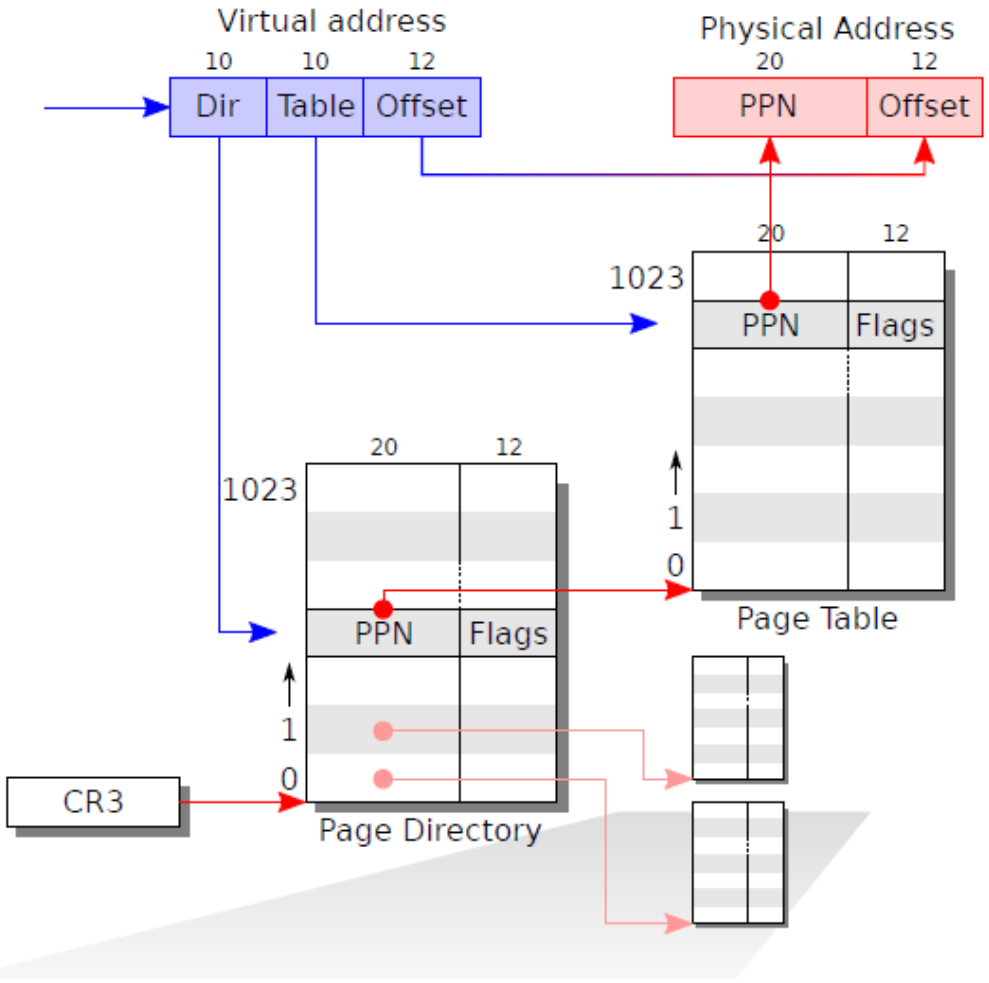
Создание процесса

Переключение контекста



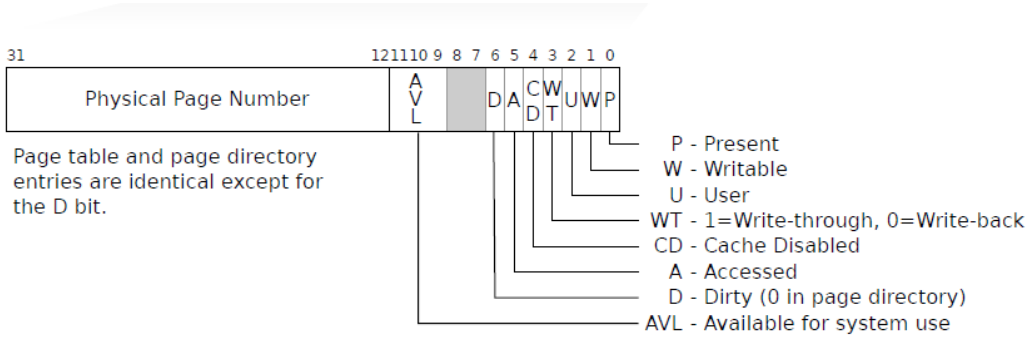
При переключении контекста происходит переключение непосредственно на сам scheduler перед выбором следующего процесса

Paiging



PPN – Physical Page Number

У каждого процесса своя таблица страниц



Смотрим код!

Спасибо за внимание!