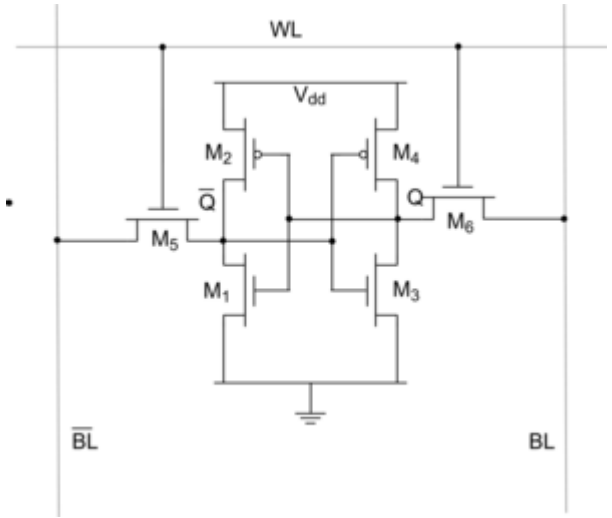
Архитектура, процессы, системные вызовы

rkondakov@ya.ru Кондаков Р.В.

Зачем



- int *p = new int;
- rm -rf *
- route add default 10.1.1.100

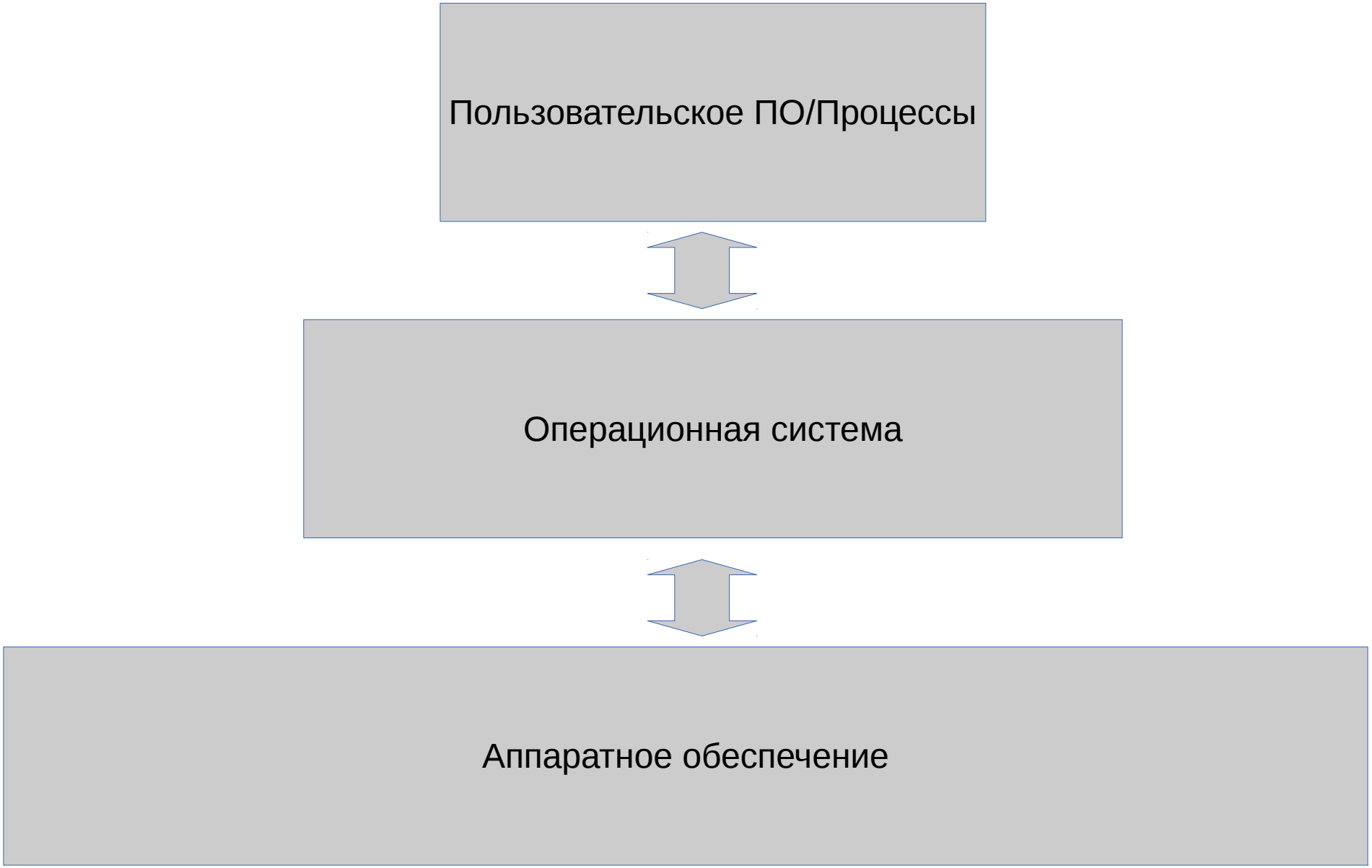
Meltdown и Spectre

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

using namespace std;

int main() {
    cout << "A";
    if (fork()) cout << "AA";
}
```

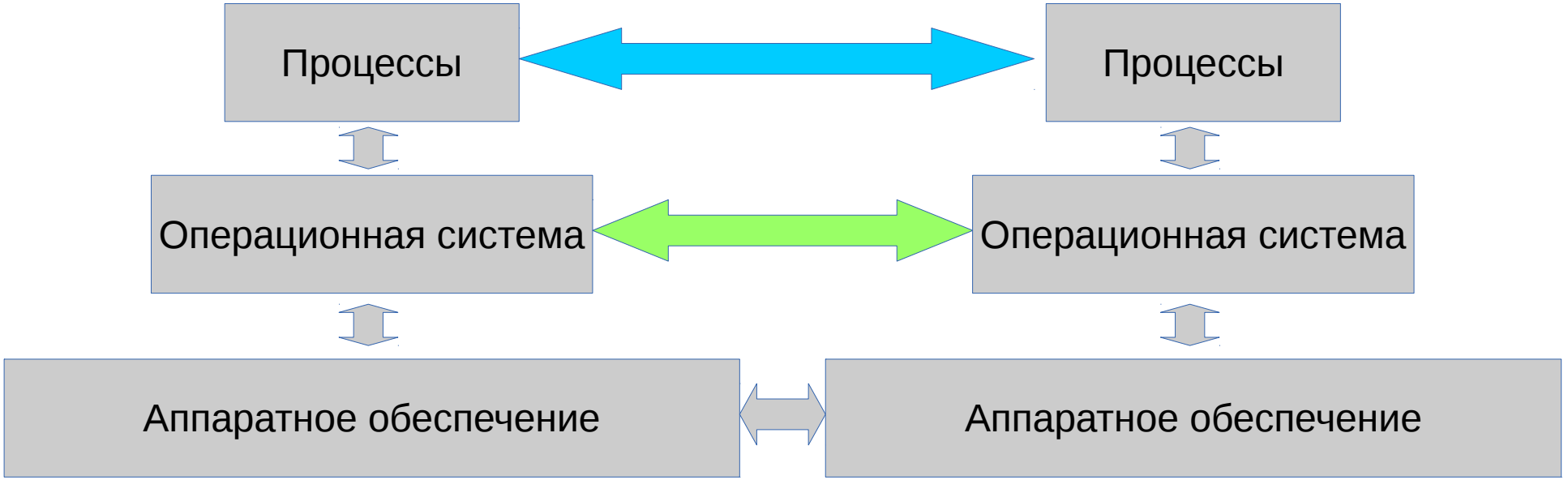
План

1.Процессы и Архитектура
2.Память, аллокаторы
3.Файлы
4.Консервативное взаимодействие процессов
5.Новое взаимодействие процессов
6.Устройство сетей
7.Сеть с точки зрения программирования
8.Файловые системы
9.Диски, разделы, устройства, загрузка ОС
10.Устройство ОС
11.Архитектура компьютера и защита процессов
12.Архитектура системы и общая производительность

Объект изучения



Объект изучения



Unix это...

- Процессы:
  - «тяжелые»
  - «легкие»
- Файлы:
  - Файловая система
  - Файлы со стороны процессов (потоки, дескрипторы)
- Взаимодействие процессов (IPC):
  - Локальное
  - Сетевое

Требуется для курса

- **Unix-совместимая операционная система (примеры: Redhat/Debian/Ubuntu (GNU/Linux), FreeBSD, MAC OS X, …)**
- **Наличие POSIX-совместимого компилятора языка Си (примеры: gcc, clang)**
- **Наличие базовой документации по unix-системе (man и info)**

Литература

Таненбаум Э.С. «Современные Операционные системы» — Питер 2015г. 1115 стр

Стивенс У.С. «UNIX. разработка сетевых приложений» — Питер, 2003г, 1088 стр

Стивенс У.С., Раго С.А. «UNIX. Профессиональное программирование» — Символ-Плюс, 2014г, 1100 стр

Рочкинд М.Д. «Программирование для UNIX» — БХВ-Петербург, 2005г, 704 стр

Стивенс У.С. «IPC» — Питер, 2002г, 576 стр

Карпов В.Е., Коньков К.А. «Операционные системы» — Интернет-университет информационных технологий, 2005 г, 632 стр

Робачевский А.М. «Операционная система UNIX» — БХВ-Петербург, 2010г 656 стр (написана хорошим языком, но IPC лишь Sys V)

Эви Немет, Гарт Снайдер и Трент Хейн «Unix и Linux. Руководство системного администратора» — Вильямс, 2012г, 1312 стр

Брайант Р., О'Халларон Д. «Компьютерные системы: архитектура и программирование. Взгляд программиста.» — БХВ-Петербург, 2005г, 1090 стр (Подробно рассмотрено представление данных, средства ускорения вычислений, взаимодействия процессов (и проблем)).

Стандарты

- SUS (Single UNIX Specification)
  - /bin/sh
  - /usr/bin/vi
  - /usr/bin/c99
- «POSIX is an acronym for Portable Operating System Interface» (http://www.opengroup.org/austin/papers/posix_faq.html)
- FHS (Filesystem Hierarchy Standard), man 7 hier

Секции man страниц (man man)

1) Исполняемые программы или команды оболочки (shell)
2) Системные вызовы (функции, предоставляемые ядром)
3) Библиотечные вызовы (функции, предоставляемые программными библиотеками)
4) Специальные файлы (обычно находящиеся в каталоге /dev)
5) Форматы файлов и соглашения, например о /etc/passwd
6) Игры
7) Разное (включает пакеты макросов и соглашения), например man(7), groff(7)
8) Команды администрирования системы (обычно нужные лишь суперпользователю)
9) Процедуры ядра [нестандартный раздел]

Сегодня

- Взаимодействия в нашей тройке
- Процессы:
  - Полноценные/тяжелые
  - Нити
  - Волокна

Взаимодействие

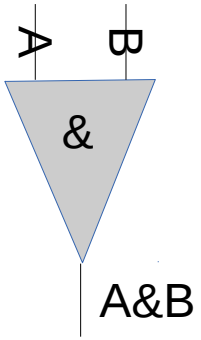
От кого	Кому	Как
Аппаратура	ОС	Аппаратные прерывания
ОС	Аппаратура	Служебные регистры
ОС	Процесс	Сигналы
Процесс	ОС	Системный вызов

Прерывания

- Сигнал, сообщающий процессору, что ему надо прервать текущее выполнение.
- Бывают:
  - Аппаратные (асинхронные):
    - Маскируемые или немаскируемые (неигнорируемые)
  - Внутренние (синхронные):
    - Программные (int 0x80)
    - Результат операций (деление на 0, выход за границу сегмента и т.д.)

Регистры

- Набор из n разрядов (битность процессора)



На каждый бит по цепочке, поэтому:

- реализовано физически лишь для внутренних элементов процессора (регистров)
- простые операции на всех регистрах
- сложные (умножение/деление) – на выделенных
- операции с памятью: явно или неявно через регистры

- Регистр инструкций (счетчик инструкций)
- Общего назначения
- Регистр стека
- ...

Ассемблер

- Ассемблер: компилятор с языка ассемблера в машинный код
- В курсе будет использоваться как вспомогательный язык

Синтаксис:

- AT&T (movl a(%edx), %eax):
  - as (SUS)
- Intel (mov eax, [a+edx]):
  - fasm
  - nasm
  - masm

Аналогия среди оболочек:

- sh:
  - ash/dash
  - bash
  - ksh
  - zsh
- csh/tcsh

```
format ELF
section '.text' executable
public _start
_start:
        push msg_len    ; size of message
        push msg        ; offset of message
        push 1          ; stdout
        mov  eax,4      ; 4 =  sys_write
        push eax
        int  0x80
        add  esp,4*3    ; чистим стек


        xor  eax,eax
        push eax        ; код выхода
        inc  eax        ; 1 = sys_exit
        int  0x80
section '.data' writeable
        msg db "Hello world",0
        msg_len = $-msg
```

- TEXT: выполнимый код (RO)
- DATA: инициализируемые данные
- BSS: неинициализируемые данные, возможно изменение размера
- STACK: стек процесса, хранение данных для функций

```
kernel:
        int    0x80; Call kernel
        ret
……..
mov  eax,4
call  kernel
```

Понятие процесса

- Поток выполнения
- Объект таблицы процессов

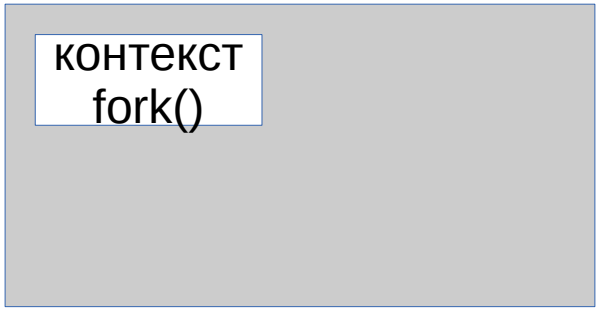
Процесс и контекст процесса

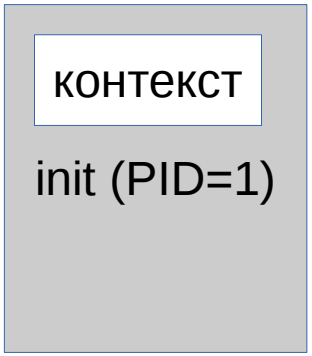
- Машина Тьюринга:
  - В каком состоянии
  - Позиция на ленте (на какой ячейке ленты)
  - Содержимое ленты
- Процесс:
  - Содержимое регистров (включая регистр инструкций)
  - Служебные данные (процессу не видны)
  - Страницы памяти

Рождение и смерть процесса

- Рождение:
  - man 2 fork:
    - pid_t fork(void);
- Смерть:
  - man 2 _exit:
    - void _exit(int status);
- Для сравнения:
  - man 3 exit
    - библиотечная функция
    - man 3 atexit
    - вызывает _exit

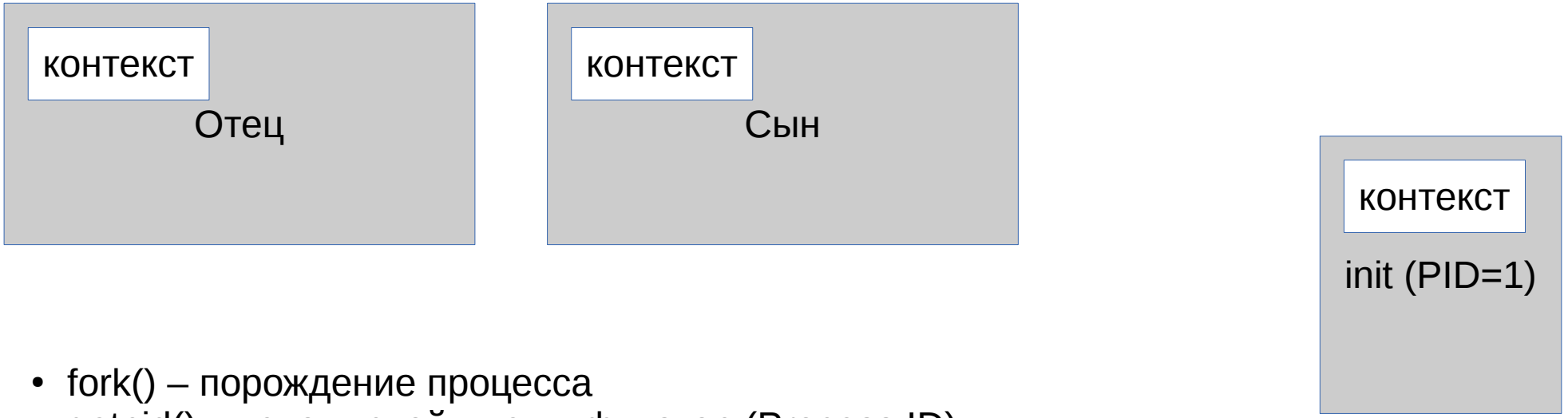
Обычные процессы: было





Вызываем fork (сам вызов находится в памяти, но адрес обрабатываемой инструкции лежит в регистре инструкций, т.е. то, что именно выполняется в данный момент определяется контекстом)

Обычные процессы: стало



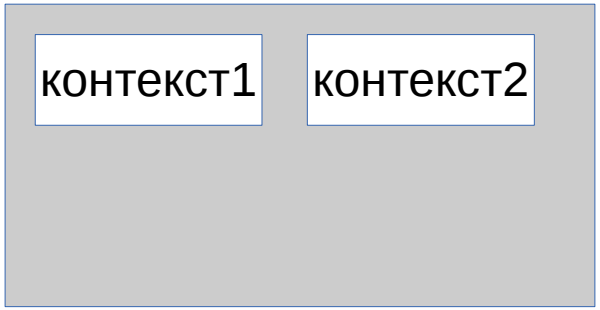
- fork() – порождение процесса
- getpid() – узнать свой идентификатор (Process ID)
- getppid() – идентификатор отца
- wait (int * status) – дождаться смерти сына
- execve (const char *filename, char *const argv[], char *const envp[])
- _exit(int status)

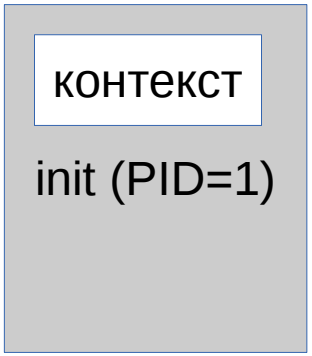
Подробности: man 2 <имя системного вызова>

- Процессы-зомби:
  - Нет памяти, состояния, только PID и статус завершения
- Вопрос:
  «Что вернет getppid, если «отец» умрет к этом моменту?»
- Ответ: «1», init не только является «прапра...праотцом» любого процесса, но и «усыновляет» всех «сирот»
- Как за всеми уследить:
  - waitpid
  - waitid

Легкие процессы: было

Легкие процессы: стало

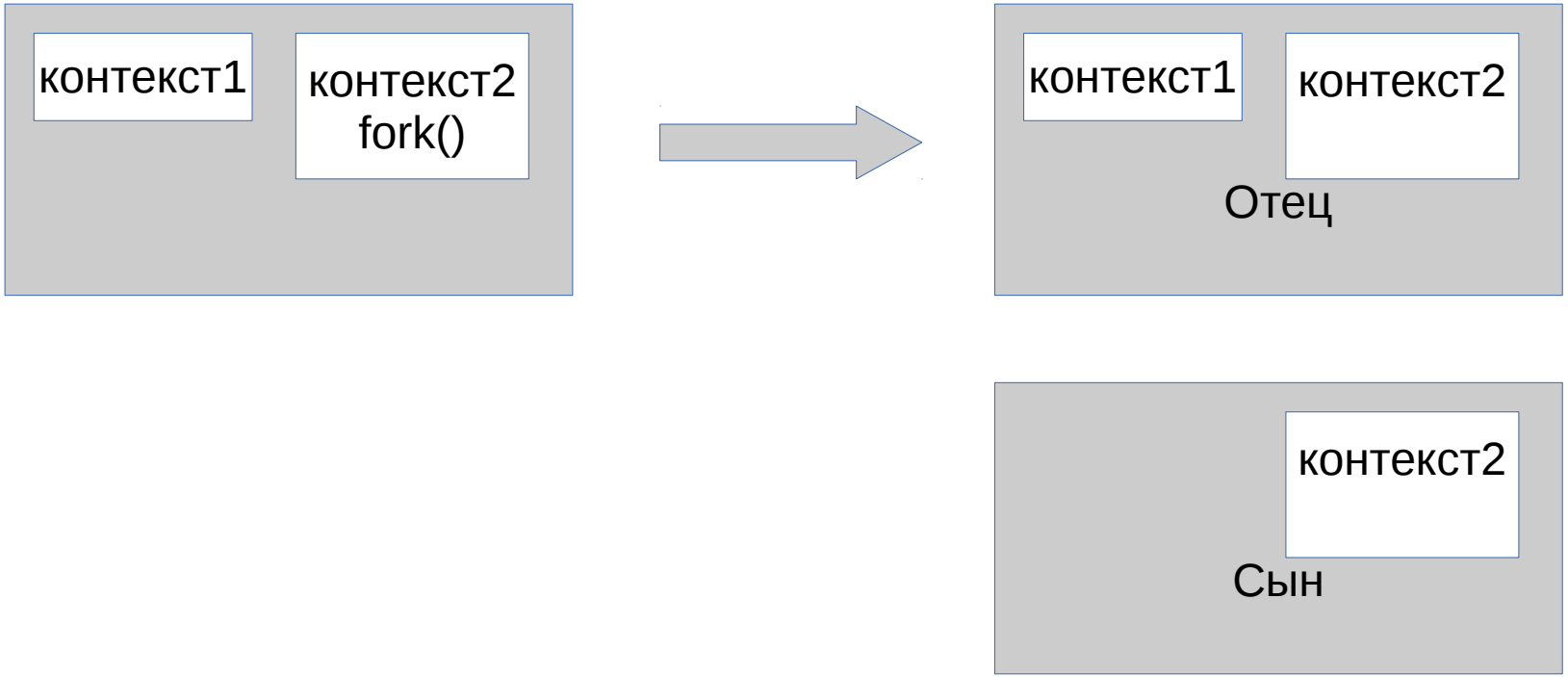




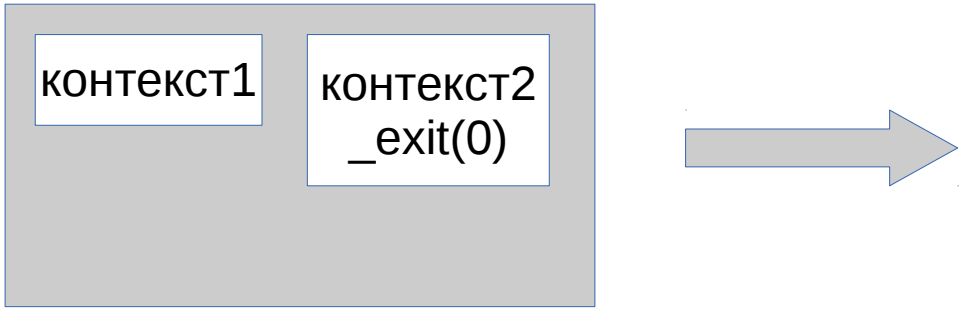
- pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine)(void *), void *arg);
- pthread_join(pthread_t thread, void **value_ptr);
- pthread_exit(void *value_ptr);

Подробности: man 3 <имя функции>

Легкие процессы и fork



Легкие процессы и _exit



- exit (man 3 exit) вызывает _exit
- Завершение функции main (что доступно основной нити) эквивалентен вызову exit

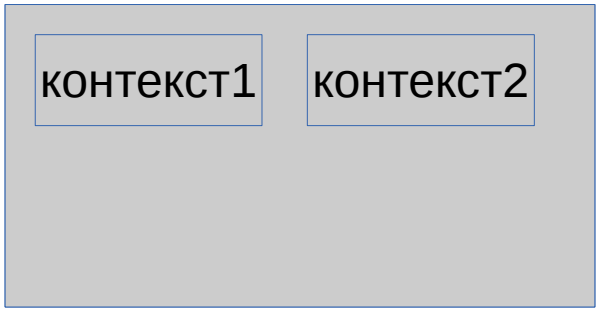
Нити N:K, 1:1, волокна

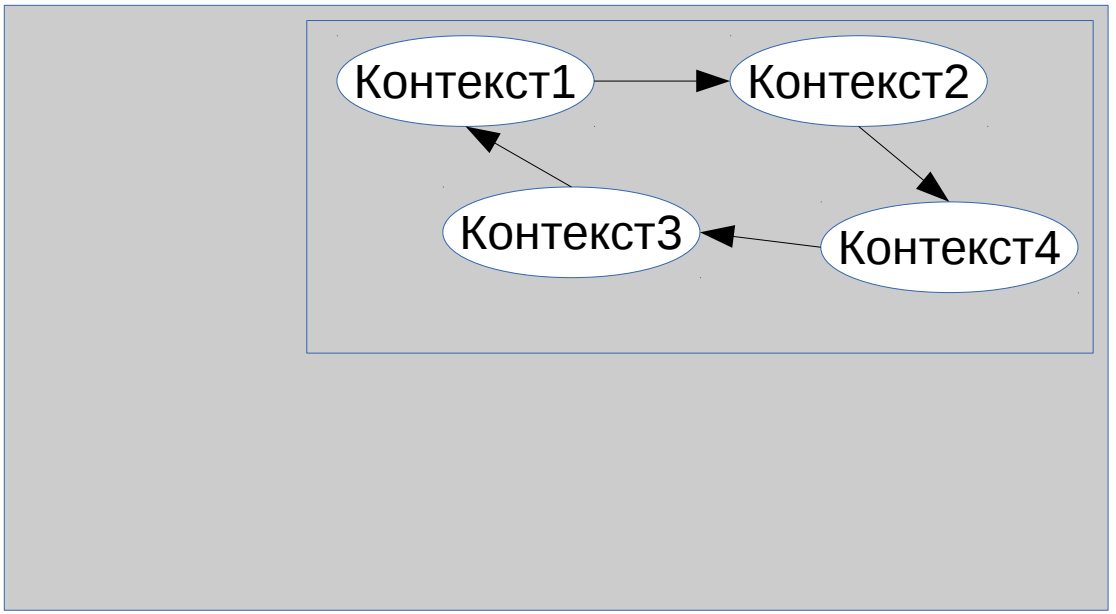
- Кто разделяет время выполнения процесса между нитями:
  - Сам процесс: модель N:K (ранние реализации)
  - Ядро: модель 1:1 (все текущие)
- Волокна (fiber):
  - int getcontext(ucontext_t *ucp);
  - int setcontext(const ucontext_t *ucp);
  - int swapcontext(ucontext_t *oucp, ucontext_t *ucp);
  - void makecontext(ucontext_t *ucp, void (*func)(), int argc, ...);

Волокна vs Нити

Нити:
- общее адресное пространство,
- переключение между потоками осуществляет ядро

Волокна:
- общее адресное пространство,
- Выполняются в рамках одного потока выполнения
- переключение между состояниями должно осуществляться самостоятельно





Два подхода в архитектуре

Фон Неймана:

- Одна область для хранения
- Одна шина передачи

Гарвардская:

- Раздельное хранение
- Две шины передачи

Общее:

- Вычислитель, который «вычисляет» программу
- Данные и код лежат вне вычислителя
- Вычислитель получает доступ к данным и коду через адрес
- Бинарное кодирование

Сравнение