Лекция 4
Многопоточное программирование
POSIX Threads

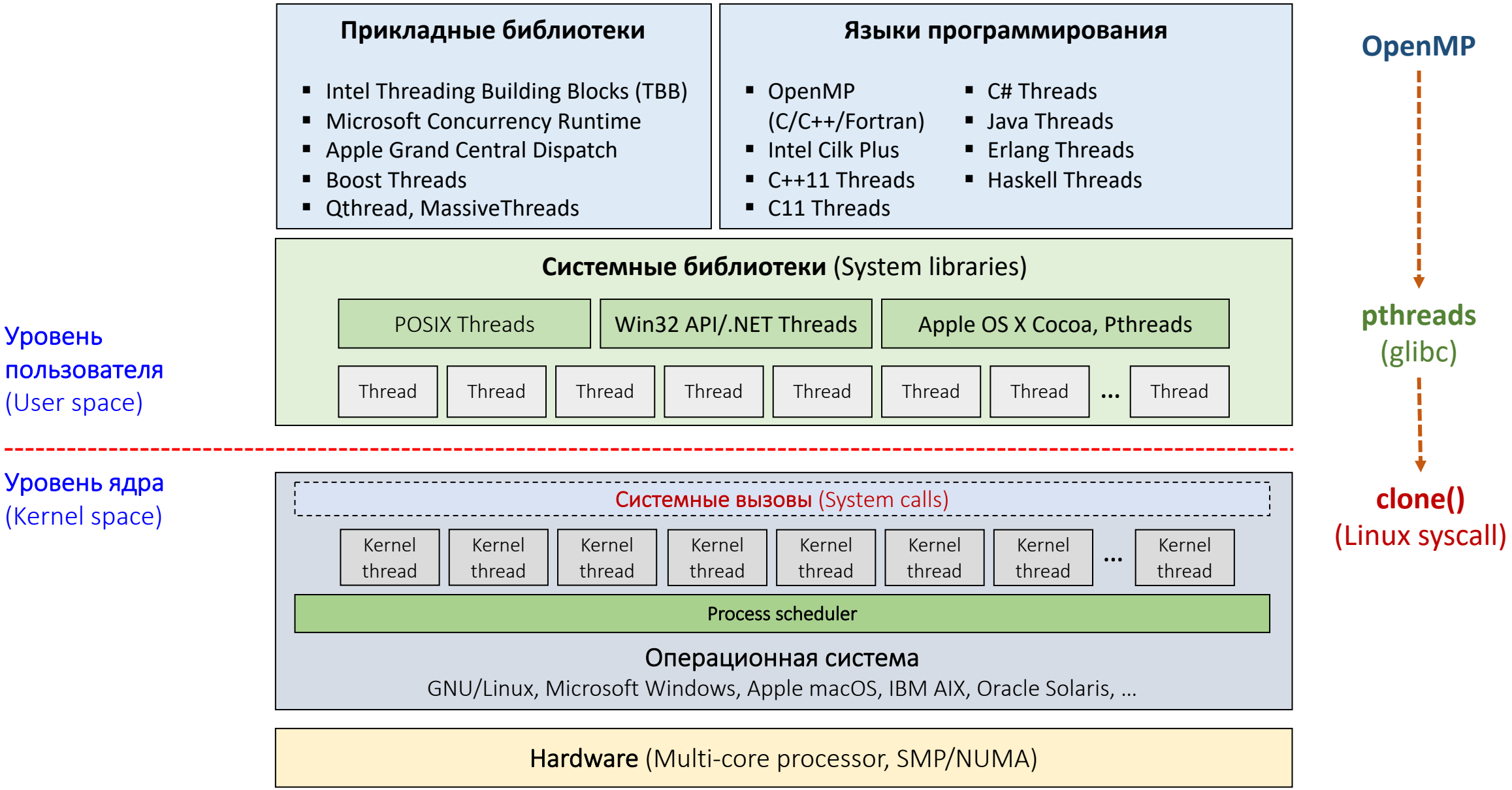
Курносов Михаил Георгиевич

E-mail: mkurnosov@gmail.com
WWW: www.mkurnosov.net

Курс «Параллельные вычислительные технологии»
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)
Весенний семестр, 2020

CC-BY

Средства многопоточного программирования



Процессы и потоки

- **Программа** (program) – исполняемый файл в определенном формате
  - GNU/Linux – ELF
  - Microsoft Windows – PE
  - Apple maxOS – Mach-O
- **Процесс** (process) – экземпляр запущенной программы
  - Виртуальное адресное пространство (address space)
  - Главный поток выполнения (thread)
- **Поток выполнения** (thread) – код и локальная область памяти для хранения данных:
  стек и TLS – thread local storage

Программы и процессы

```c
#include <stdio.h>

const int globalConst = 100;
int globalVar = 10;
int globalVarNI;

int main()
{
    int stackVar = 10;
    static int staticVar = 100;

    printf("Hello, World!\n");
    return 0;
}
```

```
$ gcc -o prog ./prog.c
```

```
# Символы исполняемого файла
$ objdump --syms ./prog

./prog:     file format elf64-x86-64

SYMBOL TABLE:
00000000004005b0 g     O .rodata  0000000000000004     globalConst
0000000000601024 g     O .data    0000000000000004     globalVar
0000000000601028 l     O .data    0000000000000004     staticVar.2214
00000000004004f6 g     F .text    0000000000000020     main
0000000000601030 g     O .bss     0000000000000004     globalVarNI
```

- Исполняемый файл содержит секции, которые сформированы компилятором (gcc) и компоновщиком (ld)
  - .rodata – глобальные константы
  - .data – инициализированные глобальные переменные
  - .bss – неинициализированные глобальные переменные
  - .text – код программы
- При запуске программы секции загружаются в память процесса

Программы и процессы

```c
#include <stdio.h>

const int globalConst = 100;
int globalVar = 10;
int globalVarNI;

int main()
{
    int stackVar = 10;
    static int staticVar = 100;

    int stackArray[10000000];

    return 0;
}
```

```
$ ulimit -a
core file size          (blocks, -c) unlimited
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 61834
max locked memory       (kbytes, -l) 64
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) 61834
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited

$ ./prog
Segmentation fault (core dumped)
```

- Размер стека главного потока равен 8 KiB
  (8 * 1024 * 1024 / 4 элементов типа int, 4 * 1024 * 1024)
- В программе создан автоматический массив из 10 000 000 элементов

Процессы и потоки


```
// Uninitialized data (BSS)
int sum[100]; // BSS

// Initialized data (Data)
float grid[100][100] = {1.0};

int main()
{
    // Local variable (stack)
    double s = 0.0;

    // Allocate from the heap
    float *x = malloc(1000);
    // ...
    free(x);
}
```

Стандарт POSIX Threads

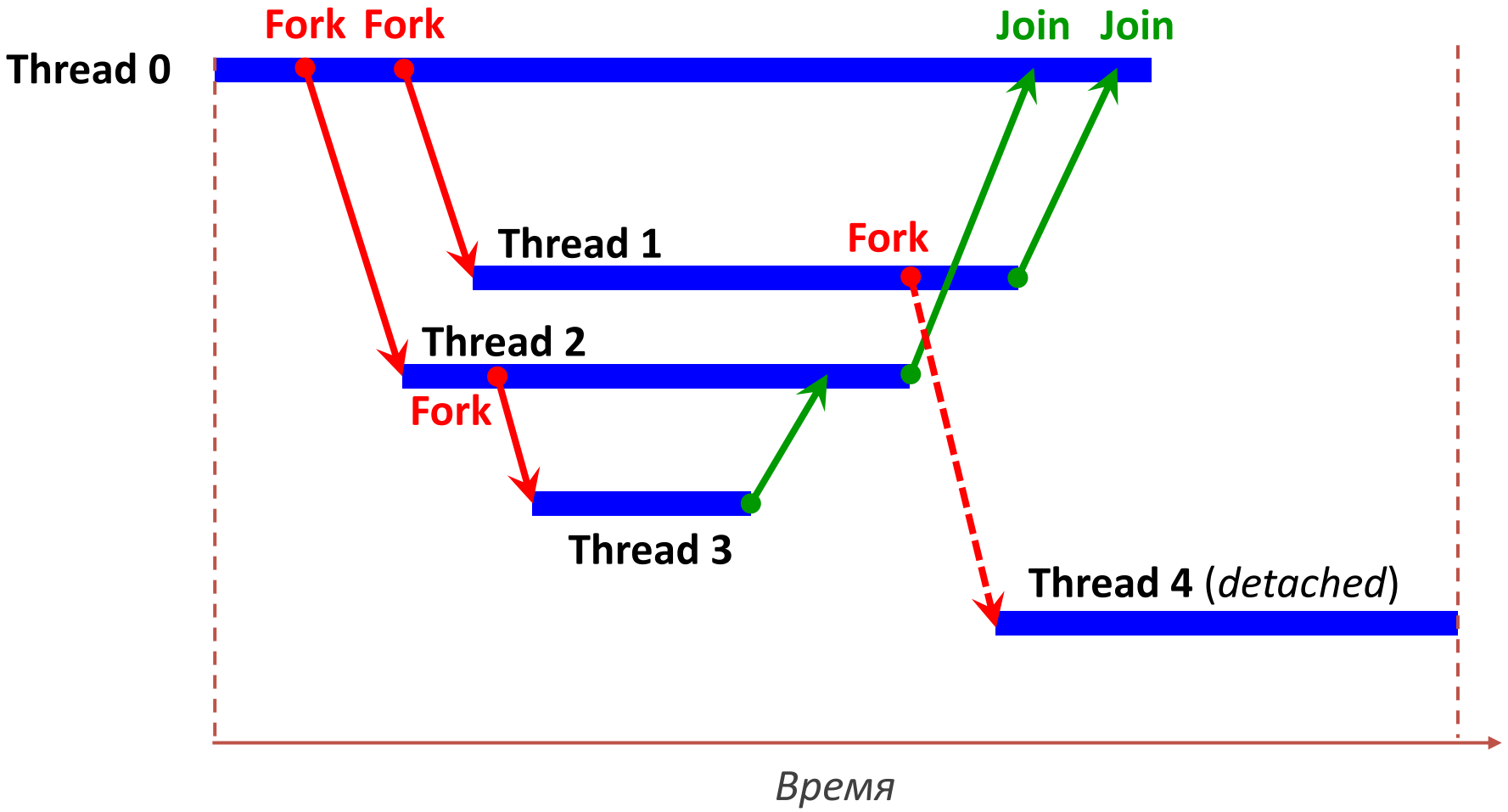
- **POSIX.1c: Threads extensions (IEEE Std 1003.1c-1995)** – определяет API для управления потоками
  - ❑ Thread Creation, Control, and Cleanup
  - ❑ Thread Scheduling
  - ❑ Thread Synchronization
  - ❑ Signal Handling

- **The Single UNIX Specification** // http://www.unix.org/version4/

- **Реализации POSIX Threads**
  - ❑ FreeBSD, NetBSD, OpenBSD, GNU/Linux, Apple OS X, Solaris
  - ❑ Windows Services for UNIX (POSIX subsystem), pthreads-w32, mingw-w64 pthreads

- `man pthread`
- `man nptl`

Модель fork-join управления потоками



- **Fork –** создание нового потока
- **Join** – ожидание одним потоком завершения другого (объединение потоков управления)

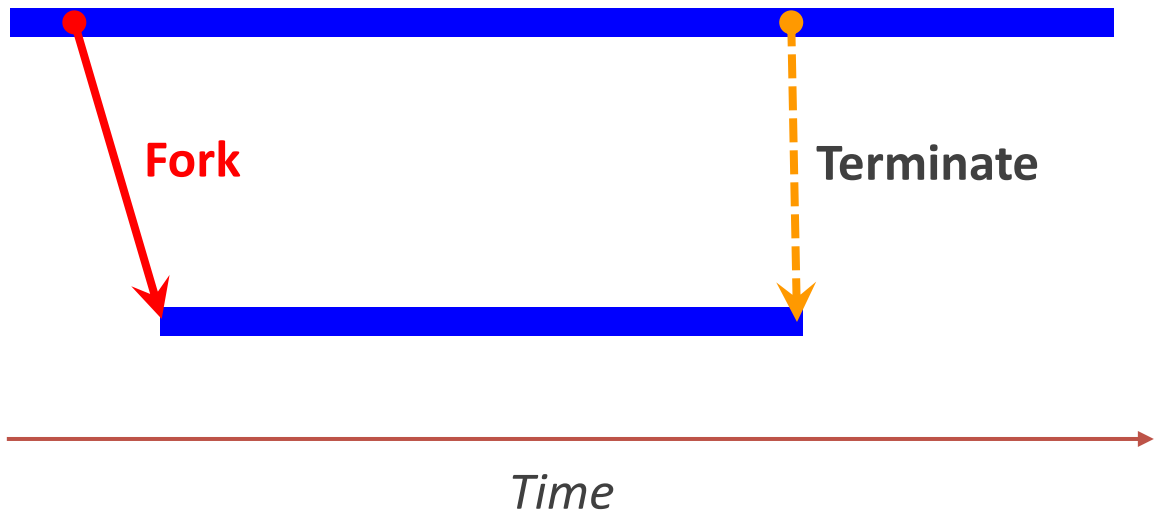
Модель fork-join управления потоками

Email Client

(GUI thread + data process thread)

Thread 0
Drawing GUI

Thread 1
Loading emails
in background



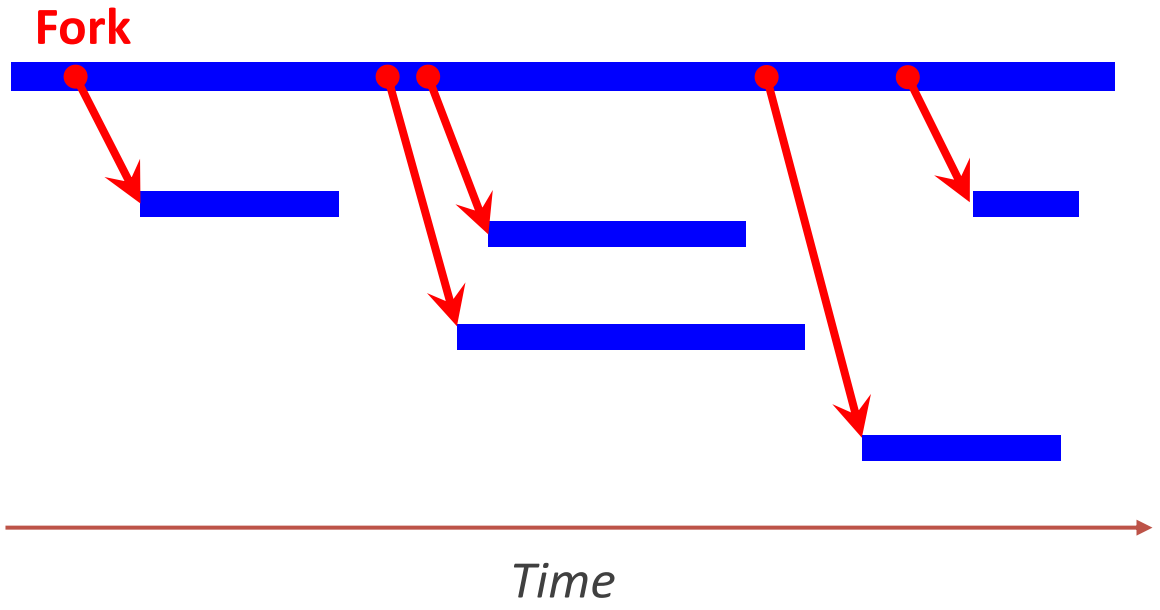
Модель fork-join управления потоками

Multithreaded HTTP-server

Master thread + 1 thread per request

Thread 0
Accepting requests

Worker thread
Loading HTML-document
and sending response



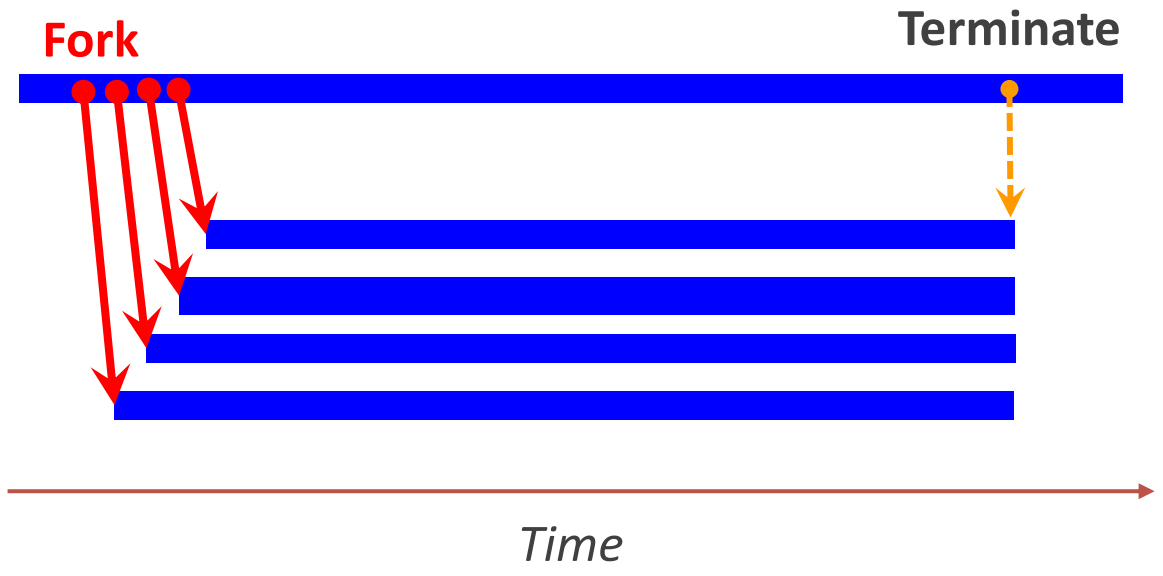
Модель fork-join управления потоками

Multithreaded HTTP-server

Master thread + **thread pool**

Thread 0
Queue of requests

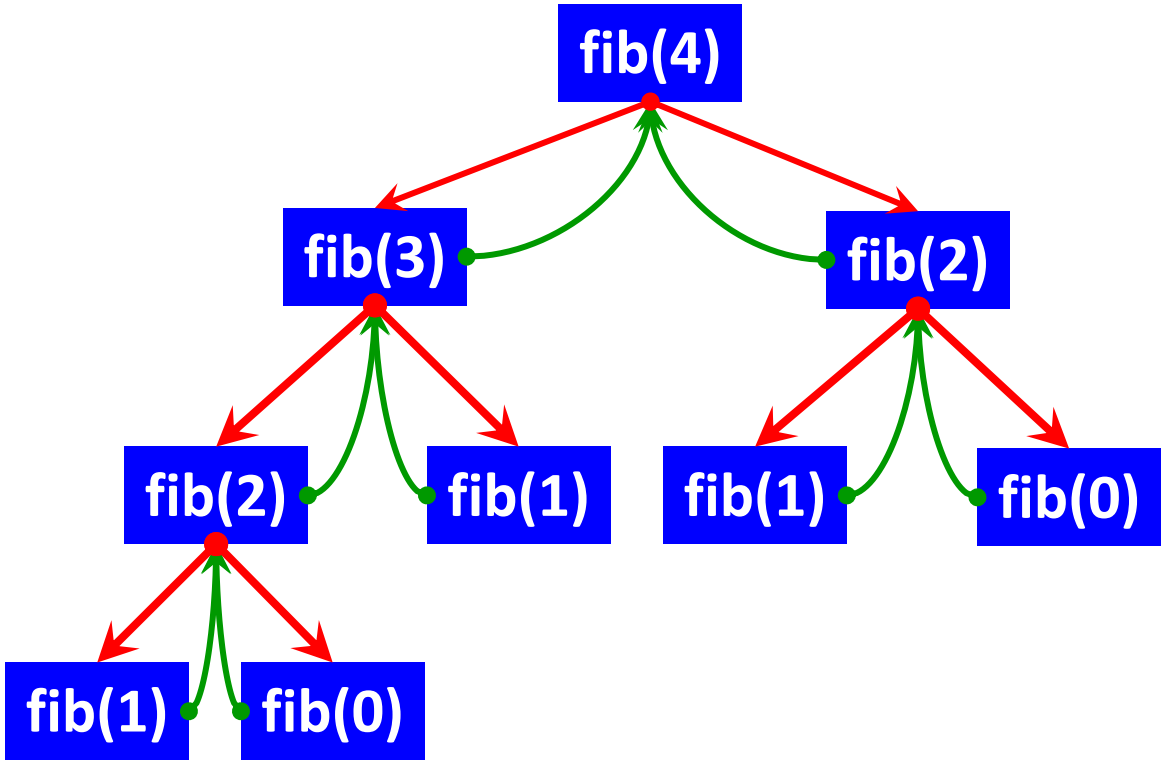
Worker thread
Processing requests
from the queue



Модель fork-join управления потоками

Recursive Parallelism – Divide & Conquer

(Parallel QuickSort, Reductions, For loops)




```
function fib(int n)
    if n < 2 then
        return n
    x = fork threadX fib(n - 1)
    y = fork threadY fib(n - 2)
    join threadX
    join threadY
    return x + y
end function
```

Обзор POSIX Threads

```
#include <pthread.h>

int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                   void *(*start_routine) (void *), void *arg);

int pthread_join(pthread_t thread, void **retval);

int pthread_attr_init(pthread_attr_t *attr);

int pthread_cancel(pthread_t thread);

pthread_t pthread_self(void);

int pthread_equal(pthread_t t1, pthread_t t2);
```

Hello, World of POSIX Threads!

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

void *thread_func(void *arg)
{
    // Код дочернего потока
    pthread_t tid = pthread_self();
    printf("Hello from slave thread %u\n", (unsigned int)tid);
    sleep(5);
    return NULL;
}

int main()
{
    pthread_t tid;
    int rc = pthread_create(&tid, NULL, thread_func, NULL);
    if (rc != 0) {
        fprintf(stderr, "Can't create thread\n");
        exit(EXIT_FAILURE);
    }
    printf("Hello from master thread %u\n", (unsigned int)pthread_self());

    // Мастер-поток ожидает завершение дочернего потока
    pthread_join(tid, NULL);
    return 0;
}
```

```
$ gcc -Wall -pthread ./prog.c –oprog

$ ./prog
Hello from master thread 1971812096
Hello from slave thread 1963546368
```

Поток завершается при:

- Вызове в нем pthread_exit()
- Выходе из функции потока
- Отмене вызовом pthread_cancel()
- При вызове exit() все потоки процесса принудительно завершаются (например, при выходе из main())

Hello, World of POSIX Threads!

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

void *thread_func(void *arg)
{
    // Код дочернего потока
    pthread_t tid = pthread_self();
    printf("Hello from slave thread %u\n", (unsigned int)tid);
    sleep(5);
    return NULL;
}

int main()
{
    pthread_t tid;
    int rc = pthread_create(&tid, NULL, thread_func, NULL);
    if (rc != 0) {
        fprintf(stderr, "Can't create thread\n");
        exit(EXIT_FAILURE);
    }
    printf("Hello from master thread %u\n", (unsigned int)pthread_self());

    // Мастер-поток ожидает завершение дочернего потока
    pthread_join(tid, NULL);
    return 0;
}
```

```
$ gcc -Wall -pthread ./prog.c –oprog

$ ./prog
Hello from master thread 4256687872
Hello from slave thread 4248422144
```

```
$ ps -L -o comm,pid,lwp,nlwp,psr,pmem,pcpu -C prog
COMMAND           PID   LWP NLWP PSR %MEM %CPU
prog            18357 18357    2   1  0.0  0.0
prog            18357 18358    2   0  0.0  0.0
```

- LWP – thread id
- NLWP – число потоков процесса
- PSR – номер процессора

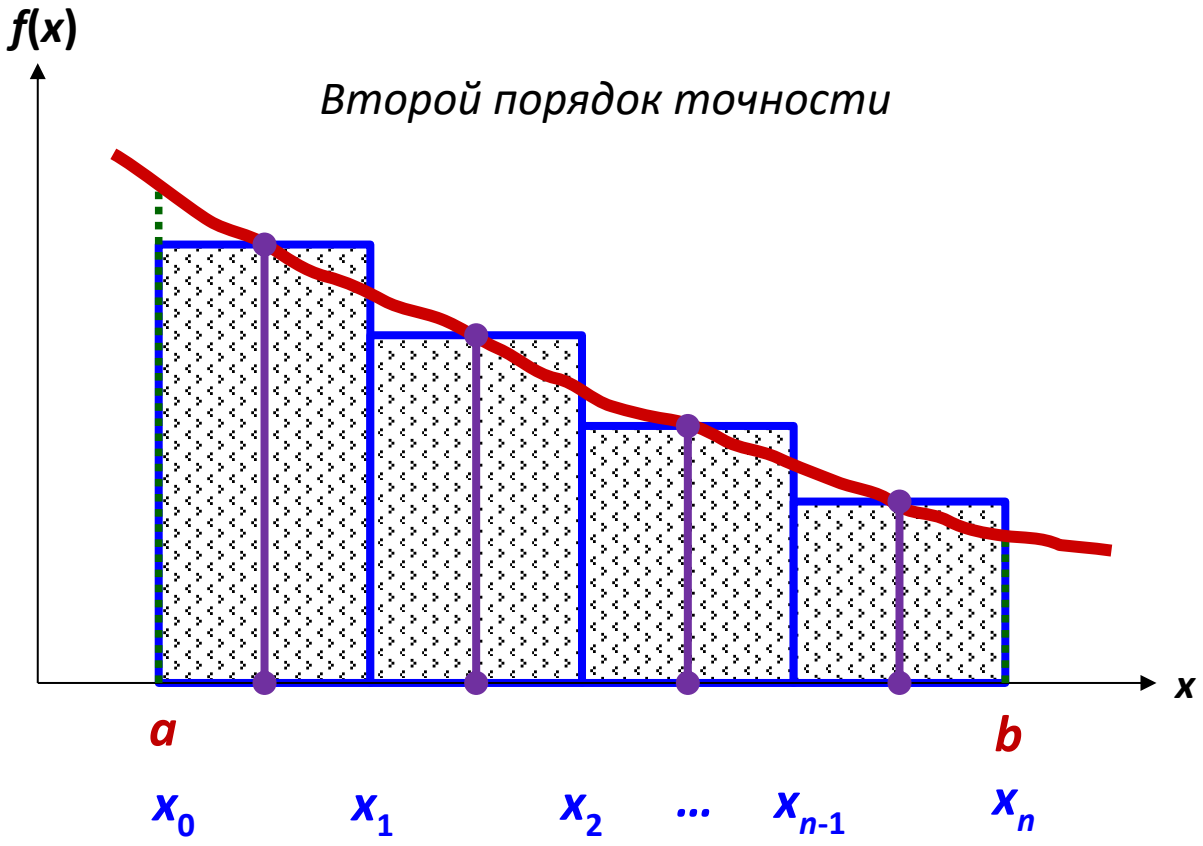
Запуск *N* потоков

```c
int main(int argc, char **argv)
{
    int nthreads = argc > 1 ? atoi(argv[1]) : 4;
    pthread_t *tids = malloc(sizeof(*tids) * nthreads);
    if (tids == NULL) {
        fprintf(stderr, "No enough memory\n");
        exit(EXIT_FAILURE);
    }
    printf("Launching %d threads\n", nthreads);
    for (int i = 0; i < nthreads; i++) {
        if (pthread_create(&tids[i], NULL, thread_func, NULL) != 0) {
            fprintf(stderr, "Can't create thread\n");
            exit(EXIT_FAILURE);
        }
    }
    // Now we have nthreads + 1 master thread
    printf("Hello from master thread %u\n",
           (unsigned int)pthread_self());

    for (int i = 0; i < nthreads; i++)
        pthread_join(tids[i], NULL);
    free(tids);
    return 0;
}
```

```
$ ./prog
Launching 4 threads
Hello from slave thread 2625984256
Hello from slave thread 2617591552
Hello from master thread 2634249984
Hello from slave thread 2609198848
Hello from slave thread 2600806144
```

```
$ ps -L -o comm,pid,lwp,nlwp,psr,pmem,pcpu -C prog
COMMAND           PID   LWP NLWP PSR %MEM %CPU
prog            18697 18697    5   1  0.0  0.0
prog            18697 18698    5   2  0.0  0.0
prog            18697 18699    5   2  0.0  0.0
prog            18697 18700    5   0  0.0  0.0
prog            18697 18701    5   0  0.0  0.0
```

Формула средних прямоугольников (midpoint rule) для вычисления определенного интеграла

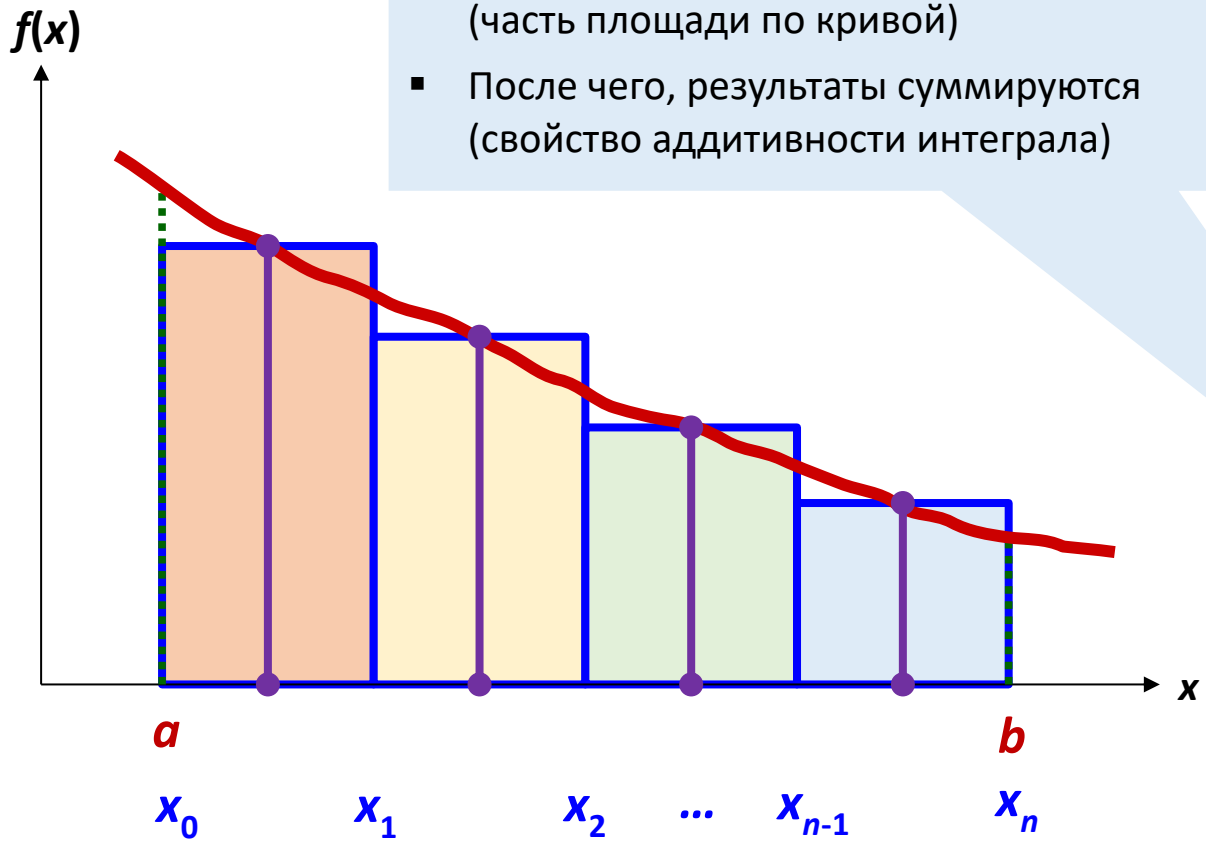


$$\int_a^b f(x)\,dx \approx h\sum_{i=1}^{n} f\left(x_{i-1}+\frac{h}{2}\right) = h\sum_{i=1}^{n} f\left(x_i-\frac{h}{2}\right), \qquad h=\frac{b-a}{n}$$

```
double func(double x)
{
    return exp(-x * x);
}

int main(int argc, char **argv)
{
    const double a = -4.0;
    const double b = 4.0;
    const int n = 100;

    double h = (b - a) / n;
    double s = 0.0;
    for (int i = 0; i < n; i++)
        s += func(a + h * (i + 0.5));
    s *= h;

    printf("Result Pi: %.12f\n", s * s);
    return 0;
}
```

Параллельный алгоритм интегрирования методом средних прямоугольников (midpoint rule)

- Каждый поток вычисляет частичную сумму (часть площади по кривой)
- После чего, результаты суммируются (свойство аддитивности интеграла)



$$\int_a^b f(x)\,dx \approx h\sum_{i=1}^{n} f\left(x_{i-1}+\frac{h}{2}\right) = h\sum_{i=1}^{n} f\left(x_i-\frac{h}{2}\right), \qquad h=\frac{b-a}{n}$$

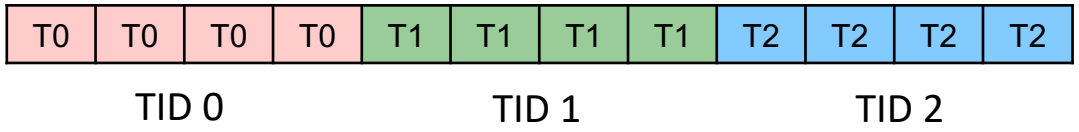
```
double func(double x)
{
    return exp(-x * x);
}

int main(int argc, char **argv)
{
    const double a = -4.0;
    const double b = 4.0;
    const int n = 100;

    double h = (b - a) / n;
    double s = 0.0;
    for (int i = 0; i < n; i++)
        s += func(a + h * (i + 0.5));
    s *= h;

    printf("Result Pi: %.12f\n", s * s);
    return 0;
}
```

Параллельный алгоритм интегрирования методом средних прямоугольников (midpoint rule)

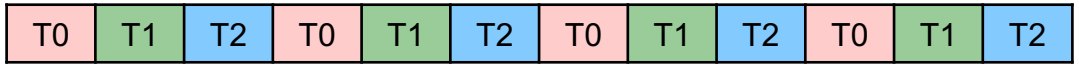
1. Итерации цикла *for* распределяются между потоками
2. Каждый поток вычисляет часть суммы (площади)
3. Суммирование результатов потоков (во всех или одном)

Варианты распределения итераций (точек) между потоками:

1) Разбиение на *p* смежных непрерывных частей



2) Циклическое распределение итераций по потокам



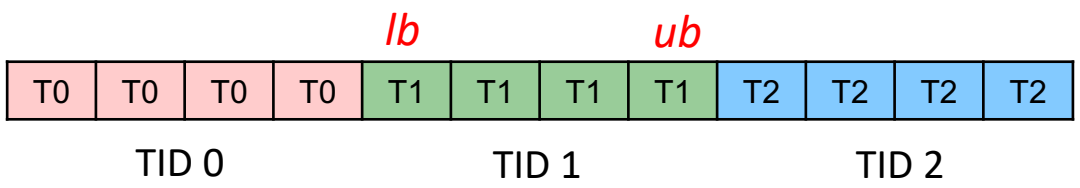
TID 0, TID 1, TID 2, TID 0, TID 1, TID 2, ...

```
double func(double x)
{
    return exp(-x * x);
}

int main(int argc, char **argv)
{
    const double a = -4.0;
    const double b = 4.0;
    const int n = 100;

    double h = (b - a) / n;
    double s = 0.0;
    for (int i = 0; i < n; i++)
        s += func(a + h * (i + 0.5));
    s *= h;

    printf("Result Pi: %.12f\n", s * s);
    return 0;
}
```

Параллельный алгоритм интегрирования методом средних прямоугольников (midpoint rule)

Каждому потоку необходимо передать:

- порядковый номер потока
- число потоков
- h, n, a




```
double func(double x)
{
    return exp(-x * x);
}

int main(int argc, char **argv)
{
    const double a = -4.0;
    const double b = 4.0;
    const int n = 100;

    double h = (b - a) / n;
    double s = 0.0;
    for (int i = 0; i < n; i++)
        s += func(a + h * (i + 0.5));
    s *= h;

    printf("Result Pi: %.12f\n", s * s);
    return 0;
}
```

Интегрирование методом средних прямоугольников (midpoint rule)

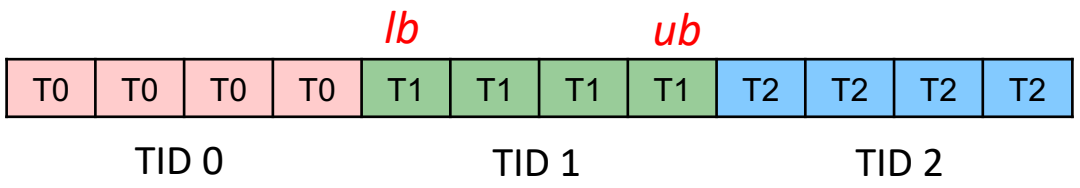
```
struct thread_data {
    pthread_t tid;
    int threadno;      // Порядковый номер потока 0, 1, 2, …
    int nthreads;      // Количество потоков
};

int main(int argc, char **argv)
{
    int nthreads = (argc > 1) ? atoi(argv[1]) : 2;
    printf("Numerical integration (%d threads): [%f, %f], n = %d\n", nthreads, a, b, n);

    struct thread_data *tdata = malloc(sizeof(*tdata) * nthreads);
    if (tdata == NULL) {
        fprintf(stderr, "No enough memory\n");
        exit(EXIT_FAILURE);
    }
    for (int i = 1; i < nthreads; i++) {      // Запуск nthreads – 1 потоков!
        tdata[i].threadno = i;
        tdata[i].nthreads = nthreads;
        if (pthread_create(&tdata[i].tid, NULL, integrate, &tdata[i]) != 0) {
            fprintf(stderr, "Can't create thread\n");
            exit(EXIT_FAILURE);
        }
    }
    tdata[0].threadno = 0;
    tdata[0].nthreads = nthreads;
    integrate(&tdata[0]);                     // Мастер-поток участвует в вычислениях, всего nthreads потоков
```

Интегрирование методом средних прямоугольников (midpoint rule)

```
    // продолжение main()

    for (int i = 1; i < nthreads; i++)
        pthread_join(tdata[i].tid, NULL);
    free(tdata);

    printf("Result Pi: %.12f\n", s * s);
    return 0;
}
```

Интегрирование методом средних прямоугольников (midpoint rule)

```
const double a = -4.0;
const double b = 4.0;
const int n = 10000000;

// Глобальная сумма
double s = 0.0;

double func(double x) {
    return exp(-x * x);
}

void *integrate(void *arg)
{
    struct thread_data *p = (struct thread_data *)arg;
    double h = (b - a) / n;

    int points_per_proc = n / p->nthreads;
    int lb = p->threadno * points_per_proc;
    int ub = (p->threadno == p->nthreads - 1) ? (n - 1) : (lb + points_per_proc - 1);

    for (int i = lb; i < ub; i++)           // Вся суть распараллеливания – цикл распределен между потоками
        s += func(a + h * (i + 0.5));

    s *= h;  // Обновление глобальной суммы
    return NULL;
}
```



Интегрирование методом средних прямоугольников (midpoint rule)

```c
struct thread_data {
    pthread_t tid;
    int threadno;      // Порядковый номер потока 0, 1, 2, …
    int nthreads;      // Количество потоков
};

int main(int argc, char **argv)
{
    int nthreads = (argc > 1) ? 
    printf("Numerical integratio

    struct thread_data *tdata = 
    if (tdata == NULL) {
        fprintf(stderr, "No enou
        exit(EXIT_FAILURE);
    }
    for (int i = 1; i < nthreads
        tdata[i].threadno = i;
        tdata[i].nthreads = nthr
        if (pthread_create(&tdat
            fprintf(stderr, "Can
            exit(EXIT_FAILURE);
        }
    }
    tdata[0].threadno = 0;
    tdata[0].nthreads = nthreads
    integrate(&tdata[0]);
```

Ошибка – при каждом запуске результат другой!

```
./integrate
Numerical integration (2 threads): [-4.000000, 4.000000], n = 10000000
Result Pi: 0.638820620709

./integrate
Numerical integration (2 threads): [-4.000000, 4.000000], n = 10000000
Result Pi: 0.000000000001

./integrate
Numerical integration (2 threads): [-4.000000, 4.000000], n = 10000000
Result Pi: 1.724699625421

./integrate
Numerical integration (2 threads): [-4.000000, 4.000000], n = 10000000
Result Pi: 0.701209910269
```

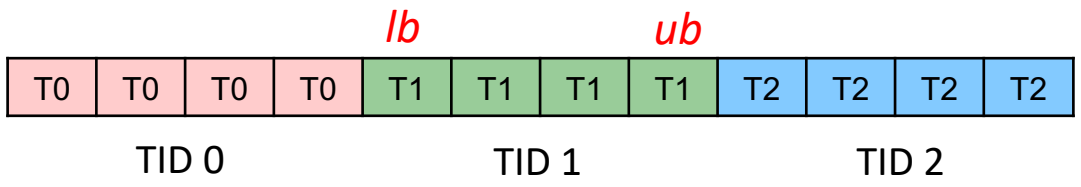
Интегрирование методом средних прямоугольников (midpoint rule)

```
const double a = -4.0;
const double b = 4.0;
const int n = 10000000;

// Глобальная сумма
double s = 0.0;

double func(double x) {
    return exp(-x * x);
}

void *integrate(void *arg)
{
    struct thread_data *p = (struct thread_data *)arg;
    double h = (b - a) / n;

    int points_per_proc = n / p->nthreads;
    int lb = p->threadno * points_per_proc;
    int ub = (p->threadno == p->nthreads - 1) ? (n - 1) : (lb + points_per_proc - 1);

    for (int i = lb; i < ub; i++)           // Вся суть распараллеливания – цикл распределен между потоками
        s += func(a + h * (i + 0.5));

    s *= h;  // Обновление глобальной суммы
    return NULL;
}
```

Состояние гонки данных (data race)

Несколько потоков осуществляют конкурентный доступ к разделяемой переменной s – одновременно читают ее и записывают

```
// s += X
Load s -> %reg0
Load X -> %reg1
Add %reg0 %reg1 -> %reg0
Store reg0 -> s
```



Состояние гонки данных (race condition, data race)

Два потока одновременно увеличивают значение переменной x на 1

(начальное значение x = 0)

Thread 0	Thread 1
x = x + 1;	**x = x + 1;**

Ожидаемый (идеальный) порядок выполнения потоков: первый поток увеличил x, затем второй

Time	Thread 0	Thread 1	x
0	Значение x = 0 загружается в регистр R процессора		0
1	Значение 0 в регистре R увеличивается на 1		0
2	Значение 1 из регистра R записывается в x		1
3		Значение x = 1 загружается в регистр R процессора	1
4		Значение 1 в регистре R увеличивается на 1	1
5		Значение 2 из регистра R	2

Ошибки нет

Состояние гонки данных (race condition, data race)

Два потока одновременно увеличивают значение переменной x на 1

(начальное значение x = 0)

Thread 0	Thread 1
x = x + 1;	**x = x + 1;**

Реальный порядок выполнения потоков (недетерминированный)

(потоки могут выполняться в любой последовательности, приостанавливаться и запускаться)

Time	Thread 0	Thread 1	x
0	Значение x = 0 загружается в регистр R процессора		0
1	Значение 0 в регистре R увеличивается на 1	Значение x = 0 загружается в регистр R процессора	0
2	Значение 1 из регистра R записывается в x	Значение 1 в регистре R увеличивается на 1	1
3		Значение 1 из регистра R записывается в x	1

Ошибка - data race

(ожидали 2)

Состояние гонки данных (data race)

- **Состояние гонки (race condition, data race)** – это состояние программы, в которой несколько потоков одновременно конкурируют за доступ к общей структуре данных (для чтения/записи)

- Порядок выполнения потоков заранее не известен – носит случайный характер

- Планировщик ОС динамически распределяет процессорное время учитывая текущую загруженность процессорных ядер, нагрузку (потоки, процессы) создают пользователи, поведение которых носит случайных характер

- Состояние гонки данных (race condition, data race) трудно обнаруживается в программах и воспроизводится в тестах (Гейзенбаг – heisenbug)

Состояние гонки данных (data race)

Динамические анализаторы кода

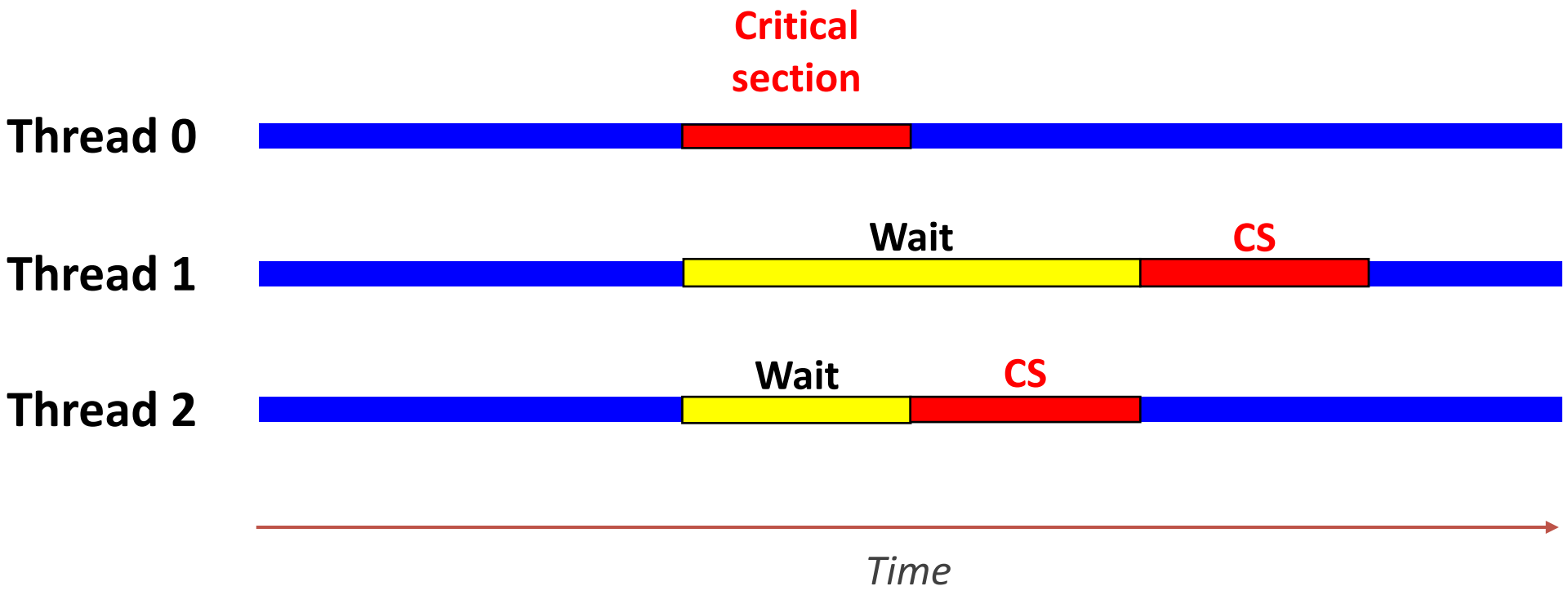
- Valgrind Helgrind, DRD
- ThreadSanitizer – a data race detector for C/C++ and Go (gcc 4.8, clang)
- Intel Thread Checker
- Oracle Studio Thread Analyzer
- Java ThreadSanitizer
- Java Chord

Статические анализаторы кода

- PVS-Studio (Viva64)

Понятие критической секции (critical section)

- **Критическая секция** (critical section) — это участок исполняемого кода, который в любой момент времени выполняется только одним потоком



Мьютексы (mutex)

- **Мьютекс (mutex) –** примитив синхронизации для организации в программах критических секций – взаимного исключения (mutual exclusion) выполнения заданного участка кода в один момент времени более одним потоком

```
double s = 0.0;  // разделяемая переменная
pthread_mutex_t mutex_sum = PTHREAD_MUTEX_INITIALIZER;

pthread_mutex_lock(&mutex_sum);
s = s + val;
pthread_mutex_unlock(&mutex_sum);
```

- **lock/acquire** – захват мьютекса осуществляется только одним потоком, остальные ожидают его освобождения
- **unlock/release** – освобождение мьютекса

Интегрирование методом средних прямоугольников

```
double s = 0.0;
pthread_mutex_t mutex_sum = PTHREAD_MUTEX_INITIALIZER;

void *integrate(void *arg)
{
    struct thread_data *p = (struct thread_data *)arg;
    double h = (b - a) / n;

    int points_per_proc = n / p->nthreads;
    int lb = p->threadno * points_per_proc;
    int ub = (p->threadno == p->nthreads - 1) ? (n - 1) : (lb + points_per_proc - 1);

    double locs = 0.0;
    for (int i = lb; i < ub; i++)
        locs += func(a + h * (i + 0.5));

    // Update global sum
    locs *= h;
    pthread_mutex_lock(&mutex_sum);
    s += locs;   // Критическая секция
    pthread_mutex_unlock(&mutex_sum);
    return NULL;
}
```

```
./integrate
Numerical integration (2 threads): [-4.000000, 4.000000], n = 10000000
Result Pi: 3.141589720795
Elapsed time (sec.): 0.151318

$ ./integrate
Numerical integration (2 threads): [-4.000000, 4.000000], n = 10000000
Result Pi: 3.141589720795
Elapsed time (sec.): 0.152941
```

Интегрирование методом средних прямоугольников

```
double s = 0.0;
pthread_mutex_t mutex_sum = PTHREAD_MUTEX_INITIALIZER;

void *integrate(void *arg)
{
    struct thread_data *p = (struct thread_data *)arg;
    double h = (b - a) / n;

    int points_per_proc = n / p->nthreads;
    int lb = p->threadno * points_per_proc;
    int ub = (p->threadno == p->nthreads - 1) ? (n - 1) : (lb + points_per_proc - 1);

    double locs = 0.0;
    for (int i = lb; i < ub; i++)
        locs += func(a + h * (i + 0.5));

    // Update global sum
    locs *= h;
    pthread_mutex_lock(&mutex_sum);
    s += locs;   // Критическая секция
    pthread_mutex_unlock(&mutex_sum);
    return NULL;
}
```

```
./integrate
Numerical integration (2 threads): [-4.000000, 4.000000], n = 10000000
Result Pi: 3.141589720795
Elapsed time (sec.): 0.151318

$ ./integrate
Numerical integration (2 threads): [-4.000000, 4.000000], n = 10000000
Result Pi: 3.141589720795
Elapsed time (sec.): 0.152941
```

Информация о системе: процессоры

```
$ cat /proc/cpuinfo
processor : 0
cpu family        : 6
model             : 23
model name        : Intel(R) Xeon(R) CPU           E5420  @ 2.50GHz
stepping  : 10
microcode : 0xa0b
cpu MHz           : 1998.000
cache size        : 6144 KB
physical id       : 0
siblings  : 4
core id           : 0
cpu cores : 4
apicid            : 0
initial apicid    : 0
fpu               : yes
fpu_exception     : yes
cpuid level       : 13
wp                : yes
flags             : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx
fxsr sse sse2 ss ht tm pbe syscall nx lm constant_tsc arch_perfmon pebs bts rep_good nopl aperfmperf pni dtes64
monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr pdcm dca sse4_1 xsave lahf_lm dtherm tpr_shadow vnmi flexpriority
bogomips  : 4987.59
clflush size      : 64
cache_alignment   : 64
address sizes     : 38 bits physical, 48 bits virtual
```

Информация о системе: процессоры

```
$ ls /sys/devices/system/cpu

drwxr-xr-x 9 root root    0 Feb  9 13:34 cpu0
drwxr-xr-x 8 root root    0 Feb  9 13:34 cpu1
drwxr-xr-x 8 root root    0 Feb  9 13:34 cpu2
drwxr-xr-x 8 root root    0 Feb  9 13:34 cpu3
drwxr-xr-x 8 root root    0 Feb  9 13:34 cpu4
drwxr-xr-x 8 root root    0 Feb  9 13:34 cpu5
drwxr-xr-x 8 root root    0 Feb  9 13:34 cpu6
drwxr-xr-x 8 root root    0 Feb  9 13:34 cpu7
drwxr-xr-x 3 root root    0 Feb 12 18:23 cpufreq
drwxr-xr-x 2 root root    0 Feb 12 18:23 cpuidle
-r--r--r-- 1 root root 4096 Feb 12 18:23 kernel_max
drwxr-xr-x 2 root root    0 Feb 12 18:23 microcode
-r--r--r-- 1 root root 4096 Feb 12 18:23 modalias
-r--r--r-- 1 root root 4096 Feb 12 18:23 offline
-r--r--r-- 1 root root 4096 Feb  9 13:34 online
-r--r--r-- 1 root root 4096 Feb  9 18:13 possible
drwxr-xr-x 2 root root    0 Feb 12 18:23 power
-r--r--r-- 1 root root 4096 Feb 12 18:23 present
--w------- 1 root root 4096 Feb 12 18:23 probe
--w------- 1 root root 4096 Feb 12 18:23 release
-rw-r--r-- 1 root root 4096 Feb  9 13:34 uevent
```

Информация о системе: процессоры

```
$ dmesg | grep CPU

[    0.000000] smpboot: Allowing 8 CPUs, 0 hotplug CPUs
[    0.000000] setup_percpu: NR_CPUS:1024 nr_cpumask_bits:1024 nr_cpu_ids:8 nr_node_ids:1
[    0.000000] PERCPU: Embedded 29 pages/cpu @ffff88025fc00000 s86784 r8192 d23808 u262144
[    0.000000] SLUB: HWalign=64, Order=0-3, MinObjects=0, CPUs=8, Nodes=1
[    0.000000]  RCU restricting CPUs from NR_CPUS=1024 to nr_cpu_ids=8.
[    0.010784] CPU: Physical Processor ID: 0
[    0.010785] CPU: Processor Core ID: 0
[    0.010788] mce: CPU supports 6 MCE banks
[    0.010796] CPU0: Thermal monitoring enabled (TM2)
[    0.033284] smpboot: CPU0: Intel(R) Xeon(R) CPU           E5420  @ 2.50GHz (fam: 06, model: 17,
stepping: 0a)
[    0.047160] NMI watchdog: enabled on all CPUs, permanently consumes one hw-PMU counter.
[    0.180019] Brought up 8 CPUs
[    3.382722] microcode: CPU0 sig=0x1067a, pf=0x40, revision=0xa0b
[    3.873633] microcode: CPU1 sig=0x1067a, pf=0x40, revision=0xa0b
[    3.873695] microcode: CPU2 sig=0x1067a, pf=0x40, revision=0xa0b
[    3.873747] microcode: CPU3 sig=0x1067a, pf=0x40, revision=0xa0b
[    3.873783] microcode: CPU4 sig=0x1067a, pf=0x40, revision=0xa0b
[    3.873823] microcode: CPU5 sig=0x1067a, pf=0x40, revision=0xa0b
[    3.873865] microcode: CPU6 sig=0x1067a, pf=0x40, revision=0xa0b
[    3.873922] microcode: CPU7 sig=0x1067a, pf=0x40, revision=0xa0b
```

Ресурсы системы

```
$  lstopo
Machine (7980MB)
  Socket L#0
    L2 L#0 (6144KB)
      L1d L#0 (32KB) + L1i L#0 (32KB) + Core L#0 + PU L#0 (P#0)
      L1d L#1 (32KB) + L1i L#1 (32KB) + Core L#1 + PU L#1 (P#4)
    L2 L#1 (6144KB)
      L1d L#2 (32KB) + L1i L#2 (32KB) + Core L#2 + PU L#2 (P#1)
      L1d L#3 (32KB) + L1i L#3 (32KB) + Core L#3 + PU L#3 (P#5)
  Socket L#1
    L2 L#2 (6144KB)
      L1d L#4 (32KB) + L1i L#4 (32KB) + Core L#4 + PU L#4 (P#2)
      L1d L#5 (32KB) + L1i L#5 (32KB) + Core L#5 + PU L#5 (P#6)
    L2 L#3 (6144KB)
      L1d L#6 (32KB) + L1i L#6 (32KB) + Core L#6 + PU L#6 (P#3)
      L1d L#7 (32KB) + L1i L#7 (32KB) + Core L#7 + PU L#7 (P#7)
  HostBridge L#0
    PCIBridge
      PCIBridge
        PCIBridge
          PCIBridge
            PCI 8086:107d
              Net L#0 "p3p1"
          PCIBridge
            PCI 8086:1096
              Net L#1 "em1"
            PCI 8086:1096
              Net L#2 "em2"
    PCIBridge
      PCI 1002:515e
    ...
```

Ресурсы системы: память

```
$  free
             total       used       free     shared    buffers     cached
Mem:      16429124   15327724    1101400       1056     361236   12686560
-/+ buffers/cache:    2279928   14149196
Swap:     18108436    1279484   16828952

$ cat /sys/devices/system/node/node0/meminfo
Node 0 MemTotal:       16429124 kB
Node 0 MemFree:         1097292 kB
Node 0 MemUsed:        15331832 kB
Node 0 Active:          7235284 kB
Node 0 Inactive:        7125736 kB
Node 0 Active(anon):     200724 kB
Node 0 Inactive(anon):  1113492 kB
Node 0 Active(file):    7034560 kB
Node 0 Inactive(file):  6012244 kB
Node 0 Unevictable:           0 kB
Node 0 Mlocked:               0 kB
Node 0 Dirty:               252 kB
Node 0 Writeback:             0 kB
Node 0 FilePages:      14127436 kB
Node 0 Mapped:           712380 kB
Node 0 AnonPages:        262700 kB
Node 0 Shmem:              1056 kB
Node 0 KernelStack:        4544 kB
Node 0 PageTables:        58760 kB
Node 0 NFS_Unstable:          0 kB
Node 0 Bounce:                0 kB
Node 0 WritebackTmp:          0 kB
Node 0 Slab:             758652 kB
Node 0 SReclaimable:     707824 kB
Node 0 SUnreclaim:        50828 kB
Node 0 AnonHugePages:     34816 kB
```

Литература

1. Blaise Barney. **POSIX Threads Programming** // https://computing.llnl.gov/tutorials/pthreads/
2. POSIX thread (pthread) libraries // http://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html
3. Эхтер Ш., Робертс Дж. **Многоядерное программирование**. – СПб.: Питер, 2010. – 316 с.
4. Эндрюс Г.Р. **Основы многопоточного, параллельного и распределенного программирования**. – М.: Вильямс, 2003. – 512 с.
5. Darryl Gove. **Multicore Application Programming: for Windows, Linux, and Oracle Solaris**. – Addison-Wesley, 2010. – 480 p.
6. Maurice Herlihy, Nir Shavit. **The Art of Multiprocessor Programming**. – Morgan Kaufmann, 2008. – 528 p.
7. Richard H. Carver, Kuo-Chung Tai. **Modern Multithreading : Implementing, Testing, and Debugging Multithreaded Java and C++/Pthreads/Win32 Programs**. – Wiley-Interscience, 2005. – 480 p.
8. Anthony Williams. **C++ Concurrency in Action: Practical Multithreading**. – Manning Publications, 2012. – 528 p.
9. Träff J.L. **Introduction to Parallel Computing** // http://www.par.tuwien.ac.at/teach/WS12/ParComp.html