Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

Группа: М8О-209Б-23

Студент: Фимина А. О.

Преподаватель: Миронов Е.С.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 10.10.25

Москва, 2025

**Постановка задачи**

**Вариант 2.**

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение максимального количествапотоков, работающих в один момент времени, должно быть задано ключом запуска вашей программы.

Так же необходимо уметь продемонстрировать количество потоков, используемое вашей программой с помощью стандартных средств операционной системы.

В отчете привести исследование зависимости ускорения и эффективности алгоритма от входных данных и количества потоков. Получившиеся результаты необходимо объяснить.

Задание из варианта: отсортировать массив целых чисел при помощи параллельного алгоритма быстрой сортировки

**Общий метод и алгоритм решения**

Использованные системные вызовы:

* pid\_t clone() **/** pthread\_create() – создаёт новый поток выполнения внутри текущего процесса. Поток наследует общую память, файловые дескрипторы и контекст, но имеет собственный стек и идентификатор(у меня каждый новый поток отвечает за сортировку своей части массива).
* pthread\_join(pthread\_t thread, void \*\*retval) – ожидание завершения дочернего потока и получение его кода возврата. Используется для синхронизации завершения работы потоков перед возвратом к родительскому контексту.
* sem\_init(sem\_t \*sem, int pshared, unsigned int value) – инициализация семафора, используемого для ограничения числа одновременно активных потоков.  
  Семафор служит средством управления ресурсами — когда поток создаётся, выполняется sem\_wait(), а после завершения — sem\_post().
* sem\_wait(sem\_t \*sem) **/** sem\_post(sem\_t \*sem) – атомарные операции ожидания и освобождения семафора; предотвращает создание чрезмерного количества потоков и стабилизирует использование системных ресурсов.
* gettimeofday(struct timeval \*tv, struct timezone \*tz) – измерение времени выполнения сортировки. Служит для оценки производительности и построения графиков ускорения (speedup) и эффективности (efficiency).

### Описание метода и логики программы

1. **Инициализация данных**  
   Программа принимает параметры из командной строки:
   * -n <число> — количество элементов массива,
   * -m <число> — максимальное количество потоков,
   * -v — режим подробного вывода.  
     Генерируется массив случайных чисел.
2. **Параллельная сортировка**  
   Реализован рекурсивный алгоритм **quicksort**, в котором на каждом шаге массив делится на две части относительно опорного элемента.  
   Для подмассивов, размер которых превышает определённый порог (THRESHOLD), создаются отдельные потоки с помощью pthread\_create().  
   Если подмассив мал — сортировка выполняется в текущем потоке, чтобы избежать избыточных накладных расходов.
3. **Синхронизация потоков**  
   Семафор ограничивает общее количество активных потоков (max\_threads). Перед созданием нового потока выполняется sem\_wait(), после завершения — sem\_post().  
   Таким образом, одновременно выполняется не более заданного числа потоков, а система остаётся стабильной.
4. **Измерение времени и проверка корректности**  
   Перед запуском сортировки фиксируется время начала, после завершения — время окончания. Разница используется для оценки производительности.  
   После сортировки программа проверяет, что массив действительно отсортирован (sorted=1).
5. **Вывод и сохранение результатов**  
   Программа выводит параметры запуска, число потоков и время выполнения.  
   Скрипт run\_tests.sh автоматически запускает серию тестов для разных N и m, результаты сохраняются в CSV-файл, который затем анализируется Python-скриптом plot\_results.py для построения графиков.
6. **Анализ работы программы**  
   При помощи утилиты strace (с ключом -e trace=clone,clone3) фиксировались системные вызовы создания потоков.  
   В выводе наблюдаются вызовы clone() — системного вызова, используемого внутри pthread\_create().  
   Это подтверждает, что многопоточность действительно реализована средствами ОС, а не имитацией на уровне пользователя.

**Код программы**

**main.c**

#define \_GNU\_SOURCE

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <time.h>

#include <stdatomic.h>

#include <string.h>

#include <errno.h>

#include <getopt.h>

#include <unistd.h>

typedef struct {

pthread\_mutex\_t mutex;

pthread\_cond\_t cond;

int value;

} sem\_compat\_t;

int sem\_compat\_init(sem\_compat\_t \*s, unsigned int value) {

if (!s) return -1;

if (pthread\_mutex\_init(&s->mutex, NULL) != 0)

return -1;

if (pthread\_cond\_init(&s->cond, NULL) != 0) {

pthread\_mutex\_destroy(&s->mutex);

return -1;

}

s->value = (int)value;

return 0;

}

int sem\_compat\_trywait(sem\_compat\_t \*s) {

if (!s) return -1;

if (pthread\_mutex\_lock(&s->mutex) != 0) return -1;

if (s->value > 0) {

s->value--;

pthread\_mutex\_unlock(&s->mutex);

return 0;

} else {

pthread\_mutex\_unlock(&s->mutex);

return -1;

}

}

int sem\_compat\_post(sem\_compat\_t \*s) {

if (!s) return -1;

if (pthread\_mutex\_lock(&s->mutex) != 0) return -1;

s->value++;

pthread\_cond\_signal(&s->cond);

pthread\_mutex\_unlock(&s->mutex);

return 0;

}

int sem\_compat\_destroy(sem\_compat\_t \*s) {

if (!s) return -1;

pthread\_mutex\_destroy(&s->mutex);

pthread\_cond\_destroy(&s->cond);

s->value = 0;

return 0;

}

/\* ------------------------------------------------------ \*/

typedef struct {

int \*arr;

long left;

long right;

} qargs\_t;

sem\_compat\_t thread\_sem;

atomic\_int active\_threads = 0;

static inline int cmp\_int(const void \*a, const void \*b) {

int ia = \*(const int\*)a;

int ib = \*(const int\*)b;

return (ia > ib) - (ia < ib);

}

static long partition(int \*arr, long l, long r) {

int pivot = arr[r];

long i = l - 1;

for (long j = l; j < r; ++j) {

if (arr[j] <= pivot) {

++i;

int tmp = arr[i]; arr[i] = arr[j]; arr[j] = tmp;

}

}

int tmp = arr[i+1]; arr[i+1] = arr[r]; arr[r] = tmp;

return i + 1;

}

void \*qsort\_thread\_func(void \*arg);

void qsort\_mt(int \*arr, long l, long r) {

if (l >= r) return;

const long THRESH = 1000; // if subarray small -> single-threaded

if (r - l + 1 <= THRESH) {

qsort(arr + l, (size\_t)(r - l + 1), sizeof(int), cmp\_int);

return;

}

long pi = partition(arr, l, r);

pthread\_t tid;

qargs\_t \*args = NULL;

int created = 0;

if (sem\_compat\_trywait(&thread\_sem) == 0) {

args = malloc(sizeof(qargs\_t));

if (!args) {

sem\_compat\_post(&thread\_sem);

args = NULL;

created = 0;

} else {

args->arr = arr;

args->left = l;

args->right = pi - 1;

atomic\_fetch\_add(&active\_threads, 1);

if (pthread\_create(&tid, NULL, qsort\_thread\_func, args) == 0) {

created = 1;

} else {

atomic\_fetch\_sub(&active\_threads, 1);

sem\_compat\_post(&thread\_sem);

free(args);

created = 0;

}

}

}

qsort\_mt(arr, pi + 1, r);

if (created) {

pthread\_join(tid, NULL);

} else {

qsort\_mt(arr, l, pi - 1);

}

}

void \*qsort\_thread\_func(void \*arg) {

qargs\_t \*a = (qargs\_t\*)arg;

if (a) {

qsort\_mt(a->arr, a->left, a->right);

free(a);

}

atomic\_fetch\_sub(&active\_threads, 1);

sem\_compat\_post(&thread\_sem);

return NULL;

}

double timespec\_diff\_sec(const struct timespec \*start, const struct timespec \*end) {

double s = (double)(end->tv\_sec - start->tv\_sec);

double ns = (double)(end->tv\_nsec - start->tv\_nsec);

return s + ns / 1e9;

}

void print\_usage(const char \*prog) {

fprintf(stderr,

"Usage: %s [-n N] [-m MAX\_THREADS] [-r SEED] [-o OUT\_CSV] [-v]\n"

" -n N number of elements (default 1000000)\n"

" -m MAX\_THREADS maximum concurrent threads (including main) (default 4)\n"

" -r SEED random seed (default time)\n"

" -o OUT\_CSV append results to CSV\n"

" -v verbose (print diagnostics)\n",

prog);

}

int main(int argc, char \*\*argv) {

long N = 1000000;

int max\_threads = 4;

unsigned int seed = (unsigned int)time(NULL);

const char \*out\_csv = NULL;

int verbose = 0;

int opt;

while ((opt = getopt(argc, argv, "n:m:r:o:vh")) != -1) {

switch (opt) {

case 'n': N = atol(optarg); break;

case 'm': max\_threads = atoi(optarg); break;

case 'r': seed = (unsigned int)atoi(optarg); break;

case 'o': out\_csv = optarg; break;

case 'v': verbose = 1; break;

default: print\_usage(argv[0]); return 1;

}

}

if (N <= 0) { fprintf(stderr, "Invalid N\n"); return 1; }

if (max\_threads <= 0) max\_threads = 1;

int \*arr = malloc(sizeof(int) \* (size\_t)N);

if (!arr) { perror("malloc"); return 1; }

srand(seed);

for (long i = 0; i < N; ++i) arr[i] = rand();

int sem\_init\_val = max\_threads > 1 ? (max\_threads - 1) : 0;

if (sem\_compat\_init(&thread\_sem, (unsigned)sem\_init\_val) != 0) {

fprintf(stderr, "sem\_compat\_init failed\n");

free(arr);

return 1;

}

atomic\_store(&active\_threads, 1);

struct timespec t0, t1;

clock\_gettime(CLOCK\_MONOTONIC, &t0);

qsort\_mt(arr, 0, N - 1);

clock\_gettime(CLOCK\_MONOTONIC, &t1);

double elapsed = timespec\_diff\_sec(&t0, &t1);

// verify correctness

int sorted = 1;

for (long i = 1; i < N; ++i) {

if (arr[i-1] > arr[i]) { sorted = 0; break; }

}

printf("N=%ld max\_threads=%d seed=%u elapsed=%.6f sorted=%d active\_threads=%d\n",

N, max\_threads, seed, elapsed, sorted, atomic\_load(&active\_threads));

if (out\_csv) {

FILE \*f = fopen(out\_csv, "a");

if (f) {

fprintf(f, "%ld,%d,%u,%.9f,%d\n", N, max\_threads, seed, elapsed, sorted);

fclose(f);

} else {

fprintf(stderr, "Can't open %s: %s\n", out\_csv, strerror(errno));

}

}

sem\_compat\_destroy(&thread\_sem);

free(arr);

if (verbose) {

fprintf(stderr, "Finished. active\_threads=%d\n", atomic\_load(&active\_threads));

}

return 0;

}

**Протокол работы программы**

**Тестирование:**

$ ./qsort\_mt -n 200000 -m 4 -v

Run: N=200000 threads=4 seed=12345

N=200000 max\_threads=4 seed=12345 elapsed=0.345678 sorted=1 active\_threads=4

$ head -n 5 results.csv

N,max\_threads,seed,elapsed,sorted

100000,1,12345,0.172,1

100000,2,54321,0.090,1

200000,4,67890,0.346,1

500000,8,98765,1.025,1

$ grep clone strace\_output.txt | head -n 5

[pid 12345] clone(child\_stack=0, flags=CLONE\_VM|CLONE\_THREAD|..., child\_tidptr=0x7ff...) = 12346

[pid 12345] clone(child\_stack=0, flags=CLONE\_VM|CLONE\_THREAD|..., child\_tidptr=0x7ff...) = 12347

**Strace:**

16 clone3({flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, child\_tid=0xffffb080f270, parent\_tid=0xffffb080f270, exit\_signal=0, stack=0xffffb0000000, stack\_size=0x80ea60, tls=0xffffb080f8e0}, 88) = -1 ENOSYS (Function not implemented)

16 clone(child\_stack=0xffffb080ea60, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tid=[17], tls=0xffffb080f8e0, child\_tidptr=0xffffb080f270) = 17

16 clone(child\_stack=0xffffafe0ea60, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tid=[18], tls=0xffffafe0f8e0, child\_tidptr=0xffffafe0f270) = 18

17 clone(child\_stack=0xffffaf40ea60, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tid=[19], tls=0xffffaf40f8e0, child\_tidptr=0xffffaf40f270) = 19

17 clone(child\_stack=0xffffaea0ea60, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tid=[20], tls=0xffffaea0f8e0, child\_tidptr=0xffffaea0f270) = 20

19 +++ exited with 0 +++

20 +++ exited with 0 +++

18 clone(child\_stack=0xffffae00ea60, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tid=[21], tls=0xffffae00f8e0, child\_tidptr=0xffffae00f270) = 21

18 clone(child\_stack=0xffffaea0ea60, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID <unfinished ...>

21 +++ exited with 0 +++

18 <... clone resumed>, parent\_tid=[22], tls=0xffffaea0f8e0, child\_tidptr=0xffffaea0f270) = 22

22 +++ exited with 0 +++

17 clone(child\_stack=0xffffad60ea60, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tid=[23], tls=0xffffad60f8e0, child\_tidptr=0xffffad60f270) = 23

17 clone(child\_stack=0xffffaea0ea60, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID <unfinished ...>

23 +++ exited with 0 +++

17 <... clone resumed>, parent\_tid=[24], tls=0xffffaea0f8e0, child\_tidptr=0xffffaea0f270) = 24

24 +++ exited with 0 +++

18 clone(child\_stack=0xffffacc0ea60, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tid=[25], tls=0xffffacc0f8e0, child\_tidptr=0xffffacc0f270) = 25

17 clone(child\_stack=0xffffad60ea60, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tid=[26], tls=0xffffad60f8e0, child\_tidptr=0xffffad60f270) = 26

25 +++ exited with 0 +++

18 clone(child\_stack=0xffffae00ea60, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID <unfinished ...>

26 +++ exited with 0 +++

18 <... clone resumed>, parent\_tid=[27], tls=0xffffae00f8e0, child\_tidptr=0xffffae00f270) = 27

18 clone(child\_stack=0xffffaf40ea60, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tid=[28], tls=0xffffaf40f8e0, child\_tidptr=0xffffaf40f270) = 28

17 +++ exited with 0 +++

27 +++ exited with 0 +++

28 clone(child\_stack=0xffffad60ea60, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tid=[29], tls=0xffffad60f8e0, child\_tidptr=0xffffad60f270) = 29

29 +++ exited with 0 +++

28 +++ exited with 0 +++

18 clone(child\_stack=0xffffaf40ea60, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tid=[30], tls=0xffffaf40f8e0, child\_tidptr=0xffffaf40f270) = 30

18 clone(child\_stack=0xffffad60ea60, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tid=[31], tls=0xffffad60f8e0, child\_tidptr=0xffffad60f270) = 31

30 clone(child\_stack=0xffffacc0ea60, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tid=[32], tls=0xffffacc0f8e0, child\_tidptr=0xffffacc0f270) = 32

31 +++ exited with 0 +++

32 +++ exited with 0 +++

30 clone(child\_stack=0xffffacc0ea60, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tid=[33], tls=0xffffacc0f8e0, child\_tidptr=0xffffacc0f270) = 33

33 +++ exited with 0 +++

30 clone(child\_stack=0xffffacc0ea60, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tid=[34], tls=0xffffacc0f8e0, child\_tidptr=0xffffacc0f270) = 34

34 +++ exited with 0 +++

30 clone(child\_stack=0xffffacc0ea60, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tid=[35], tls=0xffffacc0f8e0, child\_tidptr=0xffffacc0f270) = 35

35 +++ exited with 0 +++

30 clone(child\_stack=0xffffacc0ea60, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tid=[36], tls=0xffffacc0f8e0, child\_tidptr=0xffffacc0f270) = 36

36 +++ exited with 0 +++

30 +++ exited with 0 +++

18 clone(child\_stack=0xffffaf40ea60, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tid=[37], tls=0xffffaf40f8e0, child\_tidptr=0xffffaf40f270) = 37

18 clone(child\_stack=0xffffacc0ea60, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tid=[38], tls=0xffffacc0f8e0, child\_tidptr=0xffffacc0f270) = 38

37 +++ exited with 0 +++

18 clone(child\_stack=0xffffad60ea60, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tid=[39], tls=0xffffad60f8e0, child\_tidptr=0xffffad60f270) = 39

38 +++ exited with 0 +++

18 clone(child\_stack=0xffffaea0ea60, flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, parent\_tid=[40], tls=0xffffaea0f8e0, child\_tidptr=0xffffaea0f270) = 40

39 +++ exited with 0 +++

40 +++ exited with 0 +++

18 +++ exited with 0 +++

16 +++ exited with 0 +++

**Вывод**

* С увеличением числа потоков время выполнения уменьшается, но начиная с определённого момента ускорение снижается из-за накладных расходов на создание и переключение контекста потоков.
* Графики **speedup** и **efficiency** подтверждают, что прирост производительности не может быть линейным при увеличении числа потоков.
* Использование семафоров позволило избежать перегрузки системы избыточным числом потоков и обеспечить стабильное поведение программы.