

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-213Б-23

Студент: Гуляев А.П.

Преподаватель: Бахарев В.Д.

Оценка: \_\_\_\_\_

Дата: 7.11.24

Москва, 2024

# Постановка задачи

## Вариант 2.

Отсортировать массив целых чисел при помощи параллельного алгоритма быстрой сортировки.

## Общий метод и алгоритм решения

Использованные системные вызовы:

- `pid_t fork(void);` – создание дочернего процесса
- `pid_t wait(int)` - ожидание завершения дочерних процессов
- `key_t ftok (const char *, int)` – создание ключа System V IPC
- `int shmget(key_t, size_t, int)` – получение дескриптора (создание) разделяемого сегмента памяти
- `void *shmat(int, const void*, int)` – внесение разделяемого сегмента памяти в пространство имен процесса
- `int shmdt(const void*)` - удаление разделяемого сегмента памяти из пространства имен процесса
- `int shmctl(int, int, struct shmid_ds *)` - удаление разделяемого сегмента памяти
- `int pthread_create(pthread_t, const pthread_attr_t *, void *, void *)` - создание потока
- `int pthread_join(pthread_t, void **)` - ожидание завершения потока
- `int pthread_mutex_lock(pthread_mutex_t *)` - блокировка мьютекса
- `int pthread_mutex_unlock(pthread_mutex_t *)` - разблокировка мьютекса
- `extern void *mmap(void *, size_t, int, int, int, off_t)` - отражает файлы или устройства в памяти
- `extern int munmap(void *, size_t)` – снимает отражение

### Описание работы программы:

#### 1. Инициализация и обработка аргументов командной строки:

- Программа ожидает один аргумент командной строки — максимальное количество параллельных процессов (потоков).
- Если аргумент не передан, программа выводит сообщение об ошибке и завершает работу.

#### 2. Открытие файла и чтение данных:

- Открывается файл `test.bin` который отражается в оперативную память .

#### 3. Рекурсивная сортировка с поддержкой многопоточности:

- Создание первого потока с функцией `qsort_thread` начинает сортировку массива.
- Внутри функции `my_qsort` для части массива выбирается опорный элемент `seed`.
- Затем массив разделяется на две части: элементы, меньшие `seed`, и элементы, большие или равные `seed`.
- Если количество запущенных потоков не превышает максимальное количество потоков и размер подмассива достаточно велик, создается новый поток.
- Новый поток рекурсивно сортирует одну часть массива, а текущий — другую.
- Если лимит потоков достигнут или размер подмассива недостаточно велик, оба подмассива сортируются в рамках текущего потока.

#### 4. Ожидание завершения всех потоков:

- Каждый поток ожидает завершения созданных им потоков, используя `Pthread_join`.

#### 5. Проверка на отсортированность:

- По завершении сортировки массив проверяется на отсортированность.

#### 6. Запись отсортированных данных в файл:

- По завершении сортировки файл `test.bin`, обновляется в соответствии с отображением.

## Код программы

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

```
#include <fcntl.h>
```

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

```
#include <sys/wait.h>
```

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

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

```
#include <sys/mman.h>
```

```
#include <sys/time.h>
```

```
struct timeval tv1, tv2, dtv;
```

```

struct timezone tz;

void time_start() { gettimeofday(&tv1, &tz); }

long time_stop()
{ gettimeofday(&tv2, &tz);
  dtv.tv_sec= tv2.tv_sec -tv1.tv_sec;
  dtv.tv_usec=tv2.tv_usec-tv1.tv_usec;
  if(dtv.tv_usec<0) { dtv.tv_sec--; dtv.tv_usec+=1000000; }
  return dtv.tv_sec*1000+dtv.tv_usec/1000;
}

```

```

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

```

```

void lltostr(long long a, char **ptr) {
  if (a == 0) {
    **ptr = '0';
    (*ptr)[1] = 0;
    return;
  }
  if (a < 0) {
    **ptr = '-';
    ++(*ptr);
  }
  a = llabs(a);
  if (a > 0) {
    lltostr(a / 10, ptr);
    **ptr = '0' + (a % 10);
    ++(*ptr);
  }
  **ptr = 0;
  return;
}

```

```

void swap(long long *a, long long *b) {
    long long buff = *a;
    *a = *b;
    *b = buff;
}

```

```

void check_if_sorted(long long *array, long long n) {
    long long *ptr = array;
    for (long long i = 1; i < n; ++i) {
        if (*ptr > ptr[1]) {
            char msg[] = "not sorted\n";
            write(STDOUT_FILENO, msg, sizeof(msg) - 1);
            return;
        }
        ++ptr;
    }
    char msg[] = "sorted\n";
    write(STDOUT_FILENO, msg, sizeof(msg) - 1);
    return;
}

```

```

void sort(long long **left, long long **right, long long **seed, long long *begin, long long *end) {
    while (*left < *right) {
        if ((*left >= **seed) && (**right < **seed)) {
            swap(*left, *right);
        }
        while (**left < **seed) {
            ++(*left);
        }
    }
}

```

```

    }
    while (*right >= begin && **right >= **seed) {
        --(*right);
    }
}
if (**left > **seed) {
    swap(*left, *seed);
}
*seed = *left;

while (**left == **seed) {
    ++(*left);
}

++(*right);

return;
}

```

```

typedef struct {
    long long *array;
    long long *begin;
    long long *end;
    int *running_threads_count;
    int max_threads_count;
} qsort_args_t;

```

```

void *qsort_thread(void *args) {
    qsort_args_t *qsort_args = (qsort_args_t *)args;
    int result = my_qsort(qsort_args->array, qsort_args->begin, qsort_args->end,
        qsort_args->running_threads_count, qsort_args->max_threads_count);
}

```

```
free(args);  
return (void *) (long) result;  
}
```

```
int my_qsort(long long *array, long long *begin, long long *end, int *running_threads_count, int  
max_threads_count) {  
    if (end - begin < 2) {  
        return 0;  
    }
```

```
    long long *seed = end - 1;  
    long long *left = begin;  
    long long *right = end - 2;
```

```
    sort(&left, &right, &seed, begin, end);
```

```
    if (((end - left > 1e5)) && (*(running_threads_count) < max_threads_count)) {  
        if (pthread_mutex_lock(&mutex) != 0) {  
            char msg[] = "error locking mutex\n";  
            write(STDOUT_FILENO, msg, sizeof(msg) - 1);  
            return 1;  
        }
```

```
        (*(running_threads_count))++;
```

```
        if (pthread_mutex_unlock(&mutex) != 0) {  
            char msg[] = "error unlocking mutex\n";  
            write(STDOUT_FILENO, msg, sizeof(msg) - 1);  
            return 1;  
        }
```

```
        pthread_t thread;
```

```

qsort_args_t *args = malloc(sizeof(qsort_args_t));
if (args == NULL) {
    char msg[] = "error allocating\n";
    write(STDOUT_FILENO, msg, sizeof(msg) - 1);
    return 1;
}
args->array = array;
args->begin = left;
args->end = end;
args->running_threads_count = running_threads_count;
args->max_threads_count = max_threads_count;

if (pthread_create(&thread, NULL, qsort_thread, args) != 0) {
    char msg[] = "error creating thread\n";
    write(STDOUT_FILENO, msg, sizeof(msg) - 1);
    free(args);
    return 1;
}

my_qsort(array, begin, right, running_threads_count, max_threads_count);

pthread_join(thread, NULL);

if (pthread_mutex_lock(&mutex) != 0) {
    char msg[] = "error locking mutex\n";
    write(STDOUT_FILENO, msg, sizeof(msg) - 1);
    return 1;
}

(*(running_threads_count)--);

if (pthread_mutex_unlock(&mutex) != 0) {

```



```

        char msg[] = "error unlocking mutex\n";
        write(STDOUT_FILENO, msg, sizeof(msg) - 1);
        return 1;
    }
} else {
    my_qsort(array, left, end, running_threads_count, max_threads_count);
    my_qsort(array, begin, right, running_threads_count, max_threads_count);
}

return 0;
}

```

```

int main(int argc, char* argv[]) {
    if (argc != 2) {
        char msg[] = "wrong number of args\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        return -1;
    }
}

```

```

int max_threads_count = atoi(argv[1]);

```

```

int running_threads_count = 1;

```

```

long long n = (long long)(1e8);

```

```

int fd;

```

```

if ((fd = open("test.bin", O_RDWR)) < 0) {
    char msg[] = "file error\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);

    return -1;
}

```

```
}
```

```
// long long n = 6;
```

```
// long long array[6] = {1LL, 5LL, 9LL, 2LL, 8LL, 13LL};
```

```
long long *array;
```

```
if ((array = mmap(NULL, n * sizeof(long long), PROT_READ|PROT_WRITE, MAP_PRIVATE,  
fd, 0)) == (void *)(-1)) {
```

```
    char msg[] = "mmap error\n";
```

```
    write(STDERR_FILENO, msg, sizeof(msg));
```

```
    close(fd);
```

```
    return -1;
```

```
}
```

```
time_start();
```

```
write(STDOUT_FILENO, "started\n", sizeof("started\n") - 1);
```

```
pthread_t thread1;
```

```
qsort_args_t *args = malloc(sizeof(qsort_args_t));
```

```
args->array = array;
```

```
args->begin = array;
```

```
args->end = array + n;
```

```
args->max_threads_count = max_threads_count;
```

```
args->running_threads_count = &running_threads_count;
```

```
if (pthread_create(&thread1, NULL, qsort_thread, args) != 0) {
```

```
    char msg[] = "pthread_create error\n";
```

```
    write(STDERR_FILENO, msg, sizeof("pthread_create error\n") - 1);
```

```
    free(args);
```

```
    munmap(array, n * sizeof(long long));
```

```

    close(fd);
    return -1;
}

pthread_join(thread1, NULL);

long long time = (long long)time_stop();

write(STDOUT_FILENO, "completed in ", sizeof("completed in ") - 1);

char *strtime = (char *)malloc(25 * sizeof(char));
if (strtime == NULL) {
    char msg[] = "error allocating memory\n";
    write(STDOUT_FILENO, msg, sizeof(msg) - 1);

    munmap(array, n * sizeof(long long));
    close(fd);

    return 1;
}
char *ptr = strtime;
lltostr(time, &ptr);
write(STDOUT_FILENO, strtime, sizeof(strtime) - 1);
write(STDOUT_FILENO, "ms\n", sizeof("ms\n") - 1);
free(strtime);

check_if_sorted(array, n);
//msync(array, n * sizeof(long long), MS_SYNC);

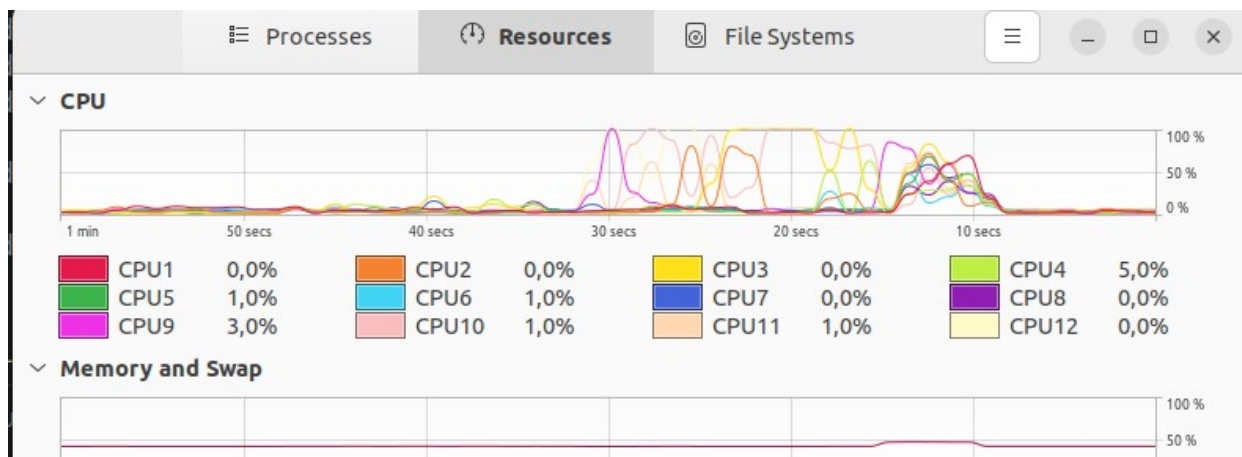
munmap(array, n * sizeof(long long));
close(fd);

```

```
return 0;  
}
```

## Протокол работы программы

```
PROBLEMS  DEBUG CONSOLE  OUTPUT  TERMINAL  PORTS  
  
● npabwa@npabwa-Vivobook-ASUSLaptop-M6500QH-M6500QH:~/Desktop/MAI_OS/lab02/src$ make o n=1  
./generate.sh  
generated  
./sol.out 1  
started  
completed in 18016ms  
sorted  
● npabwa@npabwa-Vivobook-ASUSLaptop-M6500QH-M6500QH:~/Desktop/MAI_OS/lab02/src$ make o n=12  
./generate.sh  
generated  
./sol.out 12  
started  
completed in 5092ms  
sorted  
○ npabwa@npabwa-Vivobook-ASUSLaptop-M6500QH-M6500QH:~/Desktop/MAI_OS/lab02/src$ █
```



### Тестирование:

Написал скрипт, генерирующий случайный бинарный файл, который читается программой как последовательность чисел long long int.

#### generate.sh:

```
#!/bin/bash  
  
head -c 4G </dev/urandom >test.bin  
  
echo "generated"
```

### Таблица зависимости времени работы от количества потоков:

(тестировал на случайно сгенерированных данных, 1e8 чисел, процессор 6 ядер 12 потоков)

Кол-во потоков	1	2	4	8	12
Время исполнения, с	17.6	9.9	10.4	5.3	5.7
Время исполнения, с	17.9	16.5	9.2	6.7	5.8
Время исполнения, с	17.4	11.2	8.3	7.4	5.0
Время исполнения, с	17.5	11.4	9.7	6.6	5.2
Время исполнения, с	17.8	10.5	9.7	6.9	5.2
Среднее время исполнения, с	17.7	11.9	9.5	6.6	5.4
Коэффициент производительности	1.0	1.49	1.86	2.68	3.28

**Вывод утилиты starce находится в директории с этим отчётом в силу своего размера (7000 строк).**

### Вывод

Реализовал алгоритм многопоточной быстрой сортировки.

При увеличении количества потоков в 2 раза, время исполнения уменьшается в ~1.4 раза (зависит от входных данных)