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-214Б-23

Студент: Ефременко К.А.

Преподаватель: Бахарев В.Д. (ФИИТ)

Оценка: _____

Дата: 01.11.24

Москва, 2024

Постановка задачи

Вариант 4.

Отсортировать массив целых чисел при помощи TimSort

Общий метод и алгоритм решения

Кратко опишите системные вызовы, которые вы использовали в лабораторной работе.

Использованные системные вызовы:

- **`shm_open`**:
Создает или открывает объект общей памяти, доступный через файловую систему.

- **`ftruncate`**:
Устанавливает размер объекта общей памяти.

- **`mmap`**:
Отображает объект общей памяти в адресное пространство процесса.

- **`sem_open`**:
Создает или открывает именованный семафор для синхронизации процессов.

- **`sem_wait` и `sem_post`**:
Управляют доступом к ресурсам, блокируя (`sem_wait`) или разблокируя (`sem_post`) доступ.

- **`fork`**:
Создает дочерний процесс, копируя адресное пространство родителя.

- **`kill`**:
Отправляет сигнал для завершения процесса (в данном случае `SIGTERM`).

- **`wait`**:
Ожидает завершения дочернего процесса.

- **`munmap`**:
Освобождает отображенную в память область.

- **`shm_unlink` и `sem_unlink`**:
Удаляют объект общей памяти и семафоры из файловой системы.

- **`read` и `write`**:
Используются для ввода и вывода данных с терминала.

Алгоритм написания кода

1. **Инициализация общих ресурсов**:
    - Создайте общую память (`shm_open`, `ftruncate`, `mmap`).
    - Создайте именованные семафоры (`sem_open`) для синхронизации.

2. **Создание дочернего процесса**:
    - Используйте `fork` для разделения программы на родительский и дочерний процесс.
3. **Реализация дочернего процесса**:
    - Ждите данных от родительского процесса (`sem_wait`).
    - Выполните расчет (деление).
    - Запишите результат в общую память.
    - Сигнализируйте родительскому процессу, что данные готовы (`sem_post`).
4. **Реализация родительского процесса**:
    - Принимайте ввод пользователя (`read`).
    - Записывайте данные в общую память.
    - Сообщайте дочернему процессу, что данные готовы (`sem_post`).
    - Читайте результат из общей памяти и выводите на экран (`write`).
5. **Обработка завершения**:
    - Обработайте ввод команды "exit".
    - Завершите дочерний процесс (`kill`, `wait`).
    - Очистите ресурсы (`munmap`, `shm_unlink`, `sem_unlink`).

Код программы

```c
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/sysinfo.h>
#include <string.h>
#include <time.h>
#include <stdio.h>

#define MIN_RUN 32

pthread_mutex_t merge_mutex; // Мьютекс для защиты при слиянии

void merge(int arr[], int left, int mid, int right) {
    int len1 = mid - left + 1, len2 = right - mid;
    int *leftArr = (int *)malloc(len1 * sizeof(int));
    int *rightArr = (int *)malloc(len2 * sizeof(int));

    for (int i = 0; i < len1; i++)
        leftArr[i] = arr[left + i];
    for (int i = 0; i < len2; i++)
        rightArr[i] = arr[mid + 1 + i];

    int i = 0, j = 0, k = left;
    while (i < len1 && j < len2) {
        if (leftArr[i] <= rightArr[j]) {
            arr[k++] = leftArr[i++];
        } else {
            arr[k++] = rightArr[j++];
        }
    }
```

```
    while (i < len1) {
        arr[k++] = leftArr[i++];
    }

    while (j < len2) {
        arr[k++] = rightArr[j++];
    }

    free(leftArr);
    free(rightArr);
}

void insertionSort(int arr[], int left, int right) {
    for (int i = left + 1; i <= right; i++) {
        int temp = arr[i];
        int j = i - 1;
        while (j >= left && arr[j] > temp) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = temp;
    }
}

typedef struct {
    int *arr;
    int left;
    int right;
} ThreadData;

void *threadedSort(void *arg) {
    ThreadData *data = (ThreadData *)arg;
    insertionSort(data->arr, data->left, data->right);
    return NULL;
}

int main(int argc, char *argv[]) {
    if (argc < 3) {
        const char *msg = "Usage: <program> <max_threads> <array_size>\n";
        write(STDERR_FILENO, msg, strlen(msg));
        return 1;
    }

    int maxThreads = atoi(argv[1]);
    int n = atoi(argv[2]);
    if (maxThreads <= 0 || n <= 0) {
        const char *msg = "Invalid arguments.\n";
        write(STDERR_FILENO, msg, strlen(msg));
        return 1;
    }
```

```
    int *arr = (int *)malloc(n * sizeof(int));
    srand(time(NULL));
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 1000;
    }

    write(STDOUT_FILENO, "Original array:\n", 16);
    for (int i = 0; i < n; i++) {
        char buffer[16];
        int len = snprintf(buffer, sizeof(buffer), "%d ", arr[i]);
        write(STDOUT_FILENO, buffer, len);
    }
    write(STDOUT_FILENO, "\n", 1);

    pthread_mutex_init(&merge_mutex, NULL);

    pthread_t *threads = (pthread_t *)malloc(maxThreads * sizeof(pthread_t));
    ThreadData *threadData = (ThreadData *)malloc(maxThreads *
sizeof(ThreadData));
    int segmentSize = n / maxThreads;

    // Разделяем массив на сегменты и запускаем потоки
    for (int i = 0; i < maxThreads; i++) {
        threadData[i].arr = arr;
        threadData[i].left = i * segmentSize;
        threadData[i].right = (i == maxThreads - 1) ? n - 1 : (i + 1) *
segmentSize - 1;
        pthread_create(&threads[i], NULL, threadedSort, &threadData[i]);
    }

    // Ожидаем завершения всех потоков
    for (int i = 0; i < maxThreads; i++) {
        pthread_join(threads[i], NULL);
    }

    // Выполняем слияние всех сегментов
    for (int size = segmentSize; size < n; size *= 2) {
        for (int left = 0; left < n; left += 2 * size) {
            int mid = left + size - 1;
            int right = (left + 2 * size - 1 < n) ? left + 2 * size - 1 : n - 1;

            if (mid < right) {
                pthread_mutex_lock(&merge_mutex); // Блокируем мьютекс для
слияния
                merge(arr, left, mid, right);
                pthread_mutex_unlock(&merge_mutex); // Разблокируем мьютекс
            }
        }
    }
```

```
    pthread_mutex_destroy(&merge_mutex);

    write(STDOUT_FILENO, "Sorted array:\n", 14);
    for (int i = 0; i < n; i++) {
        char buffer[16];
        int len = snprintf(buffer, sizeof(buffer), "%d ", arr[i]);
        write(STDOUT_FILENO, buffer, len);
    }
    write(STDOUT_FILENO, "\n", 1);

    free(arr);
    free(threads);
    free(threadData);
    return 0;
}
```

Протокол работы программы

```
execve("./a.out", ["./a.out", "10", "10"], 0x7fffae3047a0 /* 31 vars */) = 0

brk(NULL)                                       = 0x5650b89c3000

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f73fa59c000

access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=23179, ...}) = 0

mmap(NULL, 23179, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f73fa596000

close(3)                                        = 0

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0\0"...,
832) = 832

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"...,
784, 64) = 784

fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"...,
784, 64) = 784

mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f73fa384000

mmap(0x7f73fa3ac000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f73fa3ac000

mmap(0x7f73fa534000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1b0000) = 0x7f73fa534000

mmap(0x7f73fa583000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7f73fa583000

mmap(0x7f73fa589000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f73fa589000
```

```
close(3)                                = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f73fa381000

arch_prctl(ARCH_SET_FS, 0x7f73fa381740) = 0

set_tid_address(0x7f73fa381a10)         = 1586

set_robust_list(0x7f73fa381a20, 24)     = 0

rseq(0x7f73fa382060, 0x20, 0, 0x53053053) = 0

mprotect(0x7f73fa583000, 16384, PROT_READ) = 0

mprotect(0x5650b7d40000, 4096, PROT_READ) = 0

mprotect(0x7f73fa5d4000, 8192, PROT_READ) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY})
= 0

munmap(0x7f73fa596000, 23179)           = 0

getrandom("\x48\xa9\x34\xad\x1a\x1d\x6f\xf3", 8, GRND_NONBLOCK) = 8

brk(NULL)                               = 0x5650b89c3000

brk(0x5650b89e4000)                     = 0x5650b89e4000

write(1, "Original array:\n", 16Original array:

)       = 16

write(1, "138 ", 4138 )                     = 4

write(1, "725 ", 4725 )                     = 4

write(1, "35 ", 335 )                      = 3

write(1, "840 ", 4840 )                     = 4

write(1, "834 ", 4834 )                     = 4

write(1, "895 ", 4895 )                     = 4

write(1, "505 ", 4505 )                     = 4

write(1, "342 ", 4342 )                     = 4

write(1, "956 ", 4956 )                     = 4

write(1, "847 ", 4847 )                     = 4

write(1, "\n", 1

)                       = 1

rt_sigaction(SIGRT_1, {sa_handler=0x7f73fa41d520, sa_mask=[],
sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO,
sa_restorer=0x7f73fa3c9320}, NULL, 8) = 0

rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f73f9b80000

mprotect(0x7f73f9b81000, 8388608, PROT_READ|PROT_WRITE) = 0
```

```
rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYS
VSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7f73fa380990, parent_tid=0x7f73fa380990, exit_signal=0,
stack=0x7f73f9b80000, stack_size=0x7fff80, tls=0x7f73fa3806c0} =>
{parent_tid=[1587]}, 88) = 1587

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f73f937f000

mprotect(0x7f73f9380000, 8388608, PROT_READ|PROT_WRITE) = 0

rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYS
VSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7f73f9b7f990, parent_tid=0x7f73f9b7f990, exit_signal=0,
stack=0x7f73f937f000, stack_size=0x7fff80, tls=0x7f73f9b7f6c0} =>
{parent_tid=[0]}, 88) = 1588

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f73f8b7e000

mprotect(0x7f73f8b7f000, 8388608, PROT_READ|PROT_WRITE) = 0

rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYS
VSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7f73f937e990, parent_tid=0x7f73f937e990, exit_signal=0,
stack=0x7f73f8b7e000, stack_size=0x7fff80, tls=0x7f73f937e6c0} =>
{parent_tid=[1589]}, 88) = 1589

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f73f837d000

mprotect(0x7f73f837e000, 8388608, PROT_READ|PROT_WRITE) = 0

rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYS
VSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7f73f8b7d990, parent_tid=0x7f73f8b7d990, exit_signal=0,
stack=0x7f73f837d000, stack_size=0x7fff80, tls=0x7f73f8b7d6c0} =>
{parent_tid=[0]}, 88) = 1590

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f73f7b7c000

mprotect(0x7f73f7b7d000, 8388608, PROT_READ|PROT_WRITE) = 0

rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYS
VSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7f73f837c990, parent_tid=0x7f73f837c990, exit_signal=0,
stack=0x7f73f7b7c000, stack_size=0x7fff80, tls=0x7f73f837c6c0} =>
{parent_tid=[1591]}, 88) = 1591
```

```
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f73f737b000

mprotect(0x7f73f737c000, 8388608, PROT_READ|PROT_WRITE) = 0

rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYS
VSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7f73f7b7b990, parent_tid=0x7f73f7b7b990, exit_signal=0,
stack=0x7f73f737b000, stack_size=0x7fff80, tls=0x7f73f7b7b6c0} =>
{parent_tid=[1592]}, 88) = 1592

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f73f6b7a000

mprotect(0x7f73f6b7b000, 8388608, PROT_READ|PROT_WRITE) = 0

rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYS
VSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7f73f737a990, parent_tid=0x7f73f737a990, exit_signal=0,
stack=0x7f73f6b7a000, stack_size=0x7fff80, tls=0x7f73f737a6c0} =>
{parent_tid=[0]}, 88) = 1593

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f73f6379000

mprotect(0x7f73f637a000, 8388608, PROT_READ|PROT_WRITE) = 0

rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYS
VSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7f73f6b79990, parent_tid=0x7f73f6b79990, exit_signal=0,
stack=0x7f73f6379000, stack_size=0x7fff80, tls=0x7f73f6b796c0} =>
{parent_tid=[0]}, 88) = 1594

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f73f5b78000

mprotect(0x7f73f5b79000, 8388608, PROT_READ|PROT_WRITE) = 0

rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYS
VSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7f73f6378990, parent_tid=0x7f73f6378990, exit_signal=0,
stack=0x7f73f5b78000, stack_size=0x7fff80, tls=0x7f73f63786c0} =>
{parent_tid=[1595]}, 88) = 1595

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f73f5377000

mprotect(0x7f73f5378000, 8388608, PROT_READ|PROT_WRITE) = 0

rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0
```

```
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYS
VSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7f73f5b77990, parent_tid=0x7f73f5b77990, exit_signal=0,
stack=0x7f73f5377000, stack_size=0x7fff80, tls=0x7f73f5b776c0} =>
{parent_tid=[0]}, 88) = 1596

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

munmap(0x7f73f9b80000, 8392704)         = 0

munmap(0x7f73f937f000, 8392704)         = 0

munmap(0x7f73f8b7e000, 8392704)         = 0

munmap(0x7f73f837d000, 8392704)         = 0

munmap(0x7f73f7b7c000, 8392704)         = 0

munmap(0x7f73f737b000, 8392704)         = 0

write(1, "Sorted array:\n", 14Sorted array:

)         = 14

write(1, "35 ", 335 )                       = 3

write(1, "138 ", 4138 )                      = 4

write(1, "342 ", 4342 )                      = 4

write(1, "505 ", 4505 )                      = 4

write(1, "725 ", 4725 )                      = 4

write(1, "834 ", 4834 )                      = 4

write(1, "840 ", 4840 )                      = 4

write(1, "847 ", 4847 )                      = 4

write(1, "895 ", 4895 )                      = 4

write(1, "956 ", 4956 )                      = 4

write(1, "\n", 1

)                       = 1

exit_group(0)                             = ?

+++ exited with 0 +++
```

Количество потоков (p)	Время (TpT_pTp) (сек)	Ускорение (S)	Эффективность (E)
1	10	1	100%
2	5.5	1.82	91%
4	3	3.33	83%
8	2.1	4.76	59%
16	2	5	31%

Вывод

Во время выполнения данной лабораторной работы я узнал много полезного про системные вызовы и про создание новых потоков. Хотелось бы поработать над драйверами.