Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и
прикладной математики
Кафедра вычислительной математики и
программирования

Лабораторная работа №3 по курсу
"Операционные системы"

УПРАВЛЕНИЕ ПОТОКАМИ

Студент: Сеимов Максим Сергеевич
Группа: М8О-208Б-21
Вариант: 1
Преподаватель: Соколов Андрей Алексеевич
Оценка: _____

Дата: _____

Подпись: _____

Москва, 2023

1 Постановка задачи

Цель работы

Целью лабораторной работы №3 является приобретение практических навыков в:

- Управлении потоками в ОС
- Обеспечении синхронизации между потоками

Задание

Необходимо составить и отладить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработке использовать стандартные средства создания потоков операционной системы Unix. Ограничение потоков должно быть задано ключом запуска вашей программы.

Вариант 1

В варианте №1 нужно отсортировать массив целых чисел при помощи битонной сортировки.

Общие сведения о программе

Программа представлена одним исполняемым файлом **lab3**, компилируемым из файла **main.c**. Использованы заголовочные файлы **sys/types.h, sys/stat.h, fcntl.h, unistd.h, stdlib.h, stdio.h, string.h, limits.h, pthread.h, time.h, stdbool.h** В программе использованы следующие системные вызовы:

- **pthread_create** - Создание нового потока
- **pthread_join** - Ожидание завершения потока в главном потоке
- **exit** - Завершение выполнения процесса и возвращение статуса
- **open** - Открытие или создание файла
- **close** - Закрытие файла

Также в папке **src** есть файл **generator.c**, в котором написана программа, генерирующая файл с нужным количеством случайных чисел.

Общий метод и алгоритм решения

Общие сведения о битонной сортировке можно получить из статьи на *wikipedia.org*. В ней представлена следующая картинка, объясняющая принцип алгоритма:

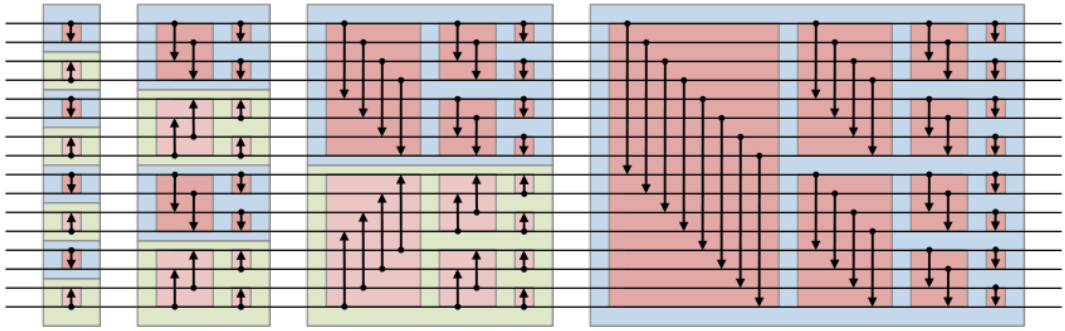

Рис. 1: Визуализация алгоритма битонной сортировки

На ней каждая стрелка обозначает компаратор, переставляющий два элемента так, чтобы стрелка указывала в сторону большего. Одна итерация сортировки на рисунке выглядит как вертикальная линия из красных прямоугольников (в каждом из которых стрелки указывают в одну сторону). Как можно заметить, в такой итерации к каждому элементу происходит всего одно обращение. Всего таких итераций для массива размером N будет $\sum_{i=1}^{\log_2 N} i$. Для каждой такой итерации будем создавать нужное число потоков, разделять компараторы между ними (синхронизация не нужна в силу того, что компараторы в рамках итерации не пересекаются) и менять местами элементы если нужно, после чего завершать потоки.

Основные файлы программы

main.c:

```c
#include <fcntl.h>
#include <limits.h>
#include <pthread.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <time.h>
#include <unistd.h>

int flag = 0;

typedef struct {
    int thread_number;
    int thread_quantity;
    int level;
    int chunk_size;
    int *data;
    int data_size;
} params;

#define FICTIOUS INT_MAX
#define STR_LEN 32

void *thread_function(void *param) {
    params args = *(params *)param;

    int num = args.thread_number;
```

```
    int *a = args.data;
    int size = args.data_size;
    int total = args.thread_quantity;
    int chunk_size = args.chunk_size;
    int level = args.level;
    flag = 1;

    while (num < size / 2) {
        int i = (num % chunk_size) + 2 * chunk_size * (num / chunk_size);
        int j = i + chunk_size;
        if ((i / (1 << level)) % 2 == 0) {
            if (a[i] > a[j]) {
                int temp = a[j];
                a[j] = a[i];
                a[i] = temp;
            }
        }
        if ((i / (1 << level)) % 2 == 1) {
            if (a[i] < a[j]) {
                int temp = a[j];
                a[j] = a[i];
                a[i] = temp;
            }
        }
        num += total;
    }
}

unsigned nearest_power_of_2(unsigned x) {
    int n = 1;
    int t = 0;
    while (n < x) {
        n <<= 1;
        t++;
    }
    return 1 << t;
}

size_t strlen_new(char *s) {
    for (size_t i = 0; i < strlen(s); ++i) {
        if (s[i] == '\n')
            return i;
    }
    return strlen(s);
}

char *get_name(const char *text) {
    printf("%s\n", text);

    char *fname_buf = calloc(sizeof(char), STR_LEN);
    if (read(0, fname_buf, STR_LEN) < 0) {
        perror("Read error");
        exit(EXIT_FAILURE);
    }

    char *fname = malloc(sizeof(char) * strlen_new(fname_buf));
    strncpy(fname, fname_buf, strlen_new(fname_buf));
    free(fname_buf);

    return fname;
```

```
}

int read_int(int fd, int *f) {
    char *buf = calloc(sizeof(char), STR_LEN);
    char c = 0;
    short i = 0;

    if (read(fd, &c, 1) < 0) {
        exit(3);
    }

    if ((c == ' ') || (c == '\n') || (c == '\0'))
        return -1;

    while (c != ' ' && c != '\n') {
        buf[i++] = c;
        read(fd, &c, 1);
    }

    *f = atoi(buf);

    if (c == '\n') {
        return 0;
    }

    return 1;
}

char *itoa(int x) {
    int n = 1;
    int a = abs(x);
    while (a >= 10) {
        a /= 10;
        n++;
    }
    char *result;
    if (x < 0) {
        result = calloc(sizeof(char), n + 1);
        a = abs(x);
        for (int i = n; i > 0; --i) {
            result[i] = a % 10 + 48;
            a /= 10;
        }
        result[0] = '-';
    } else {
        result = calloc(sizeof(char), n);
        a = x;
        for (int i = n - 1; i >= 0; --i) {
            result[i] = a % 10 + 48;
            a /= 10;
        }
    }
    return result;
}

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

    if (argc != 2) {
        char *err = "Error: number of threads expected\n";
        write(2, err, strlen(err));
```

```
151         exit(EXIT_FAILURE);
152     }
153
154     char *array_name = get_name("Enter name of file with array to sort: ");
155     char *output_name = get_name("Enter name of file for result: ");
156
157     int filedes;
158
159     if ((filedes = open(array_name, O_RDONLY)) < 0) {
160         perror(array_name);
161         exit(EXIT_FAILURE);
162     }
163
164     int n;
165     read_int(filedes, &n);
166
167     int m = nearest_power_of_2(n);
168
169     int *a = calloc(sizeof(int), m);
170     for (int i = 0; i < m; ++i) {
171         if (i < n) {
172             read_int(filedes, &a[i]);
173         } else {
174             a[i] = FICTIOUS;
175         }
176     }
177     printf("\n");
178
179     int thread_number = atoi(argv[1]);
180
181     pthread_t thread_id[thread_number];
182
183     int max_level = 0;
184     int i = 1;
185     while (i < m) {
186         i *= 2;
187         max_level++;
188     }
189
190     clock_t start, end;
191     start = clock();
192
193     for (int level = 1; level <= max_level; ++level) {
194         for (int chunk_size = 1 << (level - 1); chunk_size > 0; chunk_size /= 2) {
195             for (int i = 0; i < thread_number; ++i) {
196                 params t = {i, thread_number, level, chunk_size, a, m};
197                 int status = pthread_create(&thread_id[i], NULL, thread_function, &t);
198                 while (flag != 1)
199                     ;
200                 flag = 0;
201                 if (status != 0) {
202                     perror("Thread create error");
203                     exit(EXIT_FAILURE);
204                 }
205             }
206             for (int i = 0; i < thread_number; ++i) {
207                 pthread_join(thread_id[i], NULL);
208             }
209         }
210     }
```

```
211
212     end = clock();
213     close(filedes);
214     if ((filedes = open(output_name, O_CREAT | O_WRONLY | O_TRUNC, S_IRWXU)) < 0) {
215         perror(output_name);
216         exit(EXIT_FAILURE);
217     }
218
219     a = realloc(a, n * sizeof(int));
220
221     for (int i = 0; i < n; ++i) {
222         char *numb = itoa(a[i]);
223         write(filedes, numb, strlen(numb));
224         char space = ' ';
225         write(filedes, &space, 1);
226     }
227
228     close(filedes);
229
230     printf("%lf\n", (float)(end - start) / (CLOCKS_PER_SEC));
231 }
```

Пример работы

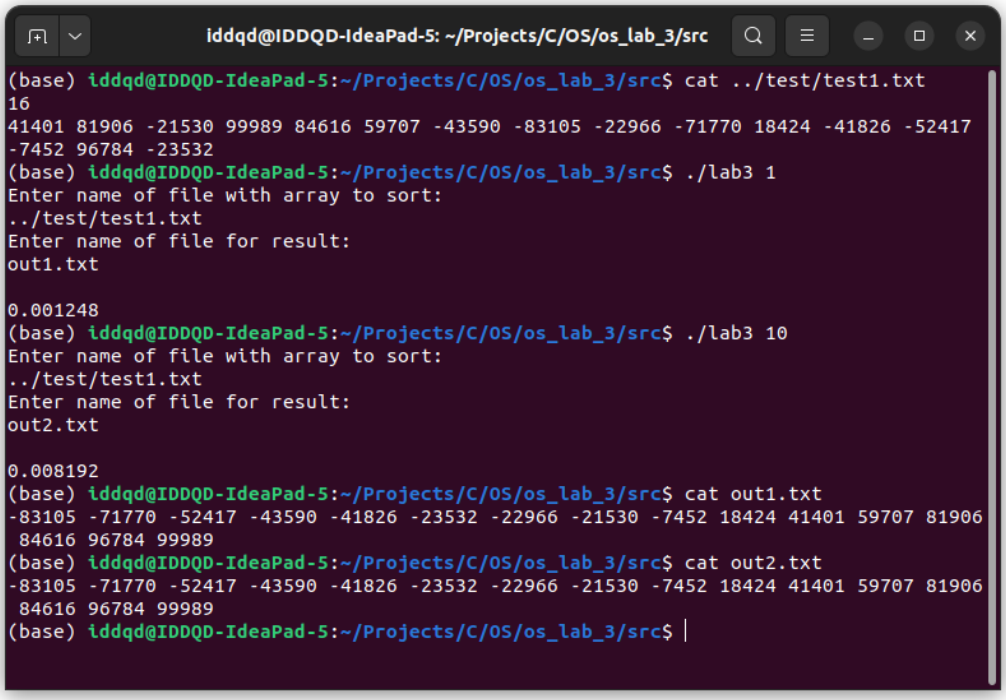

Рис. 2: Запуск программы на одном тесте с разным числом потоков

Анализ производительности

Ниже приведён график сравнения времени работы программы на одном и том же тесте для массива из 10000 элементов с разным числом потоков.

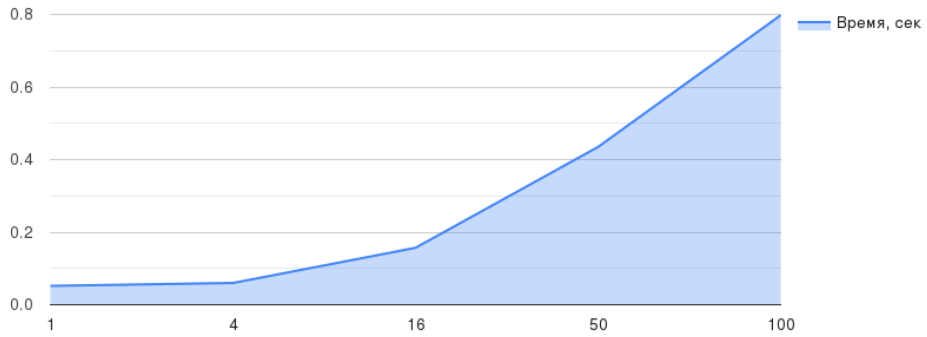

Рис. 3: Запуск программы на одном тесте с разным числом потоков

Как можно заметить, при небольшом увеличении числа потоков время выполнения почти не меняется, тогда как при значительном увеличении оно существенно снижается. Вызвано это тем, что в выбранной реализации происходит слишком много операциий создания и завершения потока, которые замедляют общее время выполнения.

Вывод

В ходе работы я научился создавать потоки средствами операционной системы Unix и работать с ними.