Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЕТ

к лабораторной работе №5

на тему

**УПРАВЛЕНИЕ ПОТОКАМИ, СРЕДСТВА СИНХРОНИЗАЦИИ.**

Студент А. В. Скворцов

Преподаватель Н. Ю. Гриценко

Минск 2024

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 4](#_Toc146752069)

[3 Результат выполнения 5](#_Toc146752070)

[Заключение 6](#_Toc146752071)

[Список использованных источников 7](#_Toc146752072)

[Приложение А (обязательное) Листинг кода 8](#_Toc146752073)

1. **ЦЕЛЬ РАБОТЫ**

Изучение подсистемы потоков (pthread), основных особенностей функционирования и управления, средств взаимодействия потоков.

Практическое проектирование, реализация и отладка программ с параллельными взаимодействующими (конкурирующими) потоками.

Реализация программы, которая сортирует большой массив данных, получаемый из файла в одном и нескольких потоках.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Поток выполнения (англ. thread — нить) — наименьшая единица обработки, исполнение которой может быть назначено ядром операционной системы. Несколько потоков выполнения могут существовать в рамках одного и того же процесса и совместно использовать ресурсы, такие как память, тогда как процессы не разделяют этих ресурсов. В частности, потоки выполнения разделяют инструкции процесса (его код) и его контекст (значения переменных, которые они имеют в любой момент времени) [1].

Множественные нити исполнения в одном процессе называют потоками и это базовая единица загрузки ЦПУ, состоящая из идентификатора потока, счетчика, регистров и стека. Потоки внутри одного процесса делят секции кода, данных, а также различные ресурсы: описатели открытых файлов, учетные данные процесса сигналы, значения umask, nice, таймеры и прочее.

У всех исполняемых процессов есть как минимум один поток исполнения. Некоторые процессы этим и ограничиваются в тех случаях, когда дополнительные нити исполнения не дают прироста производительности, но только усложняют программу. Однако таких программ с каждым днем становится относительно меньше.

Практически все современные ОС — включая Windows, Linux, Mac OS X, и Solaris — поддерживают управление потоками в режиме ядра. Однако потоки могут быть созданы не только в режиме ядра, но и в режиме пользователя. При использовании этого уровня ядро не знает о существовании потоков — все управление потоками реализуется приложением с помощью специальных библиотек. Пользовательские потоки по-разному отображаются на потоки в режиме ядра. Всего существует три модели, из которых 1:1 является наиболее часто используемой [2].

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В результате работы были разработаны два приложения, которые осуществляют сортировку большого массива данных, получаемого из файла в одном и нескольких потоках.

Первым было протестировано приложение, которое получает массив данных из файла и сортирует массив чисел в одном потоке и выведены результаты счисления (рисунок 1).

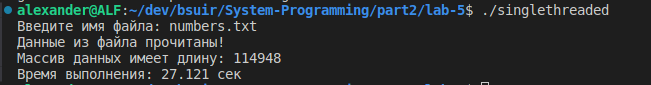


Рисунок 1 – Результаты сортировки одного потока

Вторым было протестировано приложение, сортирующее массив с помощью 20 потоков (рисунок 2).

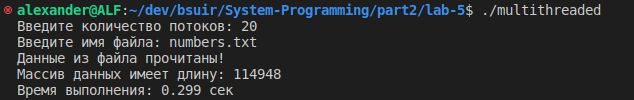


Рисунок 2 – Результаты сортировки 20 потоков

Нетрудно заметить, что на достаточно большом массиве данных разница довольно велика, количество затраченного времени различается почти в 100 раз.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были разработаны два приложения, которые осуществляют сортировку большого массива данных, получаемого из файла в одном и нескольких потоках. Был сделан вывод по результатам работы, сравнена производительность между обоими приложениями.

Также были изучены основные принципы работы многопоточности в Linux, изучены возможности взаимодействия между потоками.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Unix2018/Потоки [Электронный ресурс]. – Режим доступа: https://acm.bsu.by/wiki/Unix2018/Потоки. – Дата доступа: 26.03.2024.
2. Pthreads [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/326138/. – Дата доступа: 26.03.2024.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – sort\_multithread.c

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <sys/time.h>

#define MAX\_THREADS 8

#define MAX\_FILENAME\_LENGTH 100

typedef struct {

int\* array;

int start;

int end;

} SortParams;

void\* sort(void\* params) {

SortParams\* sp = (SortParams\*) params;

int\* array = sp->array;

int start = sp->start;

int end = sp->end;

// Сортировка фрагмента массива

for (int i = start; i <= end; i++) {

for (int j = i + 1; j <= end; j++) {

if (array[i] > array[j]) {

int temp = array[i];

array[i] = array[j];

array[j] = temp;

}

}

}

pthread\_exit(NULL);

}

int main() {

int num\_threads;

char filename[MAX\_FILENAME\_LENGTH];

int array\_capacity = 10;

int array\_size = 0;

int\* array = (int\*) malloc(array\_capacity \* sizeof(int));

printf("Введите количество потоков: ");

scanf("%d", &num\_threads);

printf("Введите имя файла: ");

scanf("%s", filename);

FILE\* file = fopen(filename, "r");

if (file == NULL) {

printf("Не удалось открыть файл\n");

return 1;

}

int value;

while (fscanf(file, "%d", &value) == 1) {

// Если фактический размер массива равен его емкости, увеличиваем емкость вдвое

if (array\_size == array\_capacity) {

array\_capacity \*= 2;

array = (int\*) realloc(array, array\_capacity \* sizeof(int));

}

array[array\_size] = value;

array\_size++;

}

fclose(file);

printf("Данные из файла прочитаны!\n");

printf("Массив данных имеет длину: %d\n", array\_size);

int fragment\_size = array\_size / num\_threads;

pthread\_t threads[MAX\_THREADS];

SortParams params[MAX\_THREADS];

struct timeval start\_time, end\_time;

gettimeofday(&start\_time, NULL);

for (int i = 0; i < num\_threads; i++) {

params[i].array = array;

params[i].start = i \* fragment\_size;

params[i].end = (i == num\_threads - 1) ? array\_size - 1 : (i + 1) \* fragment\_size - 1;

pthread\_create(&threads[i], NULL, sort, (void\*) &params[i]);

}

for (int i = 0; i < num\_threads; i++) {

pthread\_join(threads[i], NULL);

}

int\* sorted\_array = (int\*) malloc(array\_size \* sizeof(int));

int\* fragment\_indices = (int\*) calloc(num\_threads, sizeof(int));

for (int i = 0; i < array\_size; i++) {

int min\_value = 999999;

int min\_index = -1;

for (int j = 0; j < num\_threads; j++) {

int index = fragment\_indices[j];

if (index <= params[j].end && array[index] < min\_value) {

min\_value = array[index];

min\_index = j;

}

}

sorted\_array[i] = min\_value;

fragment\_indices[min\_index]++;

}

gettimeofday(&end\_time, NULL);

double execution\_time = (end\_time.tv\_sec - start\_time.tv\_sec) + (end\_time.tv\_usec - start\_time.tv\_usec) / 1000000.0;

printf("Время выполнения: %.3f сек\n", execution\_time);

free(array);

free(sorted\_array);

free(fragment\_indices);

return 0;

}