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 И. В. Бобко

Преподаватель Н. Ю. Гриценко

Минск 2024

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 4](#_Toc146752069)

[3 Результат выполнения 6](#_Toc146752070)

[Заключение 6](#_Toc146752071)

[Список использованных источников 7](#_Toc146752072)

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

1. **ЦЕЛЬ РАБОТЫ**

Целью данной работы является изучение подсистемы потоков (pthread), основных особенностей функционирования и управления, средств взаимодействия потоков. Практическое проектирование, реализация и отладка программ с параллельными взаимодействующими (конкурирующими) потоками.

Написать программу, реализующую многопоточную сортировку для большого массива данных.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

В семействах ОС Windows – каждая программа запускает один процесс выполнения, в котором находится как минимум один поток (нить). В процессе может находиться множество потоков, между которыми делится процессорное время. Один процесс не может напрямую обратиться к памяти другого процесса, а потоки же разделяют одно адресное пространство одного процесса. То есть в Windows процесс – это совокупность потоков.

В Linux же немного по-другому. Сущность процесса такая же, как и в Windows – это исполняемая программа со своими данными. Но вот поток в Linux является отдельным процессом (можно встретить название как «легковесный процесс», LWP). Различие такое же – процесс отдельная программа со своей памятью, не может напрямую обратиться к памяти другого процесса, а вот поток, хоть и отдельный процесс, имеет доступ к памяти процесса-родителя. LWP процессы создаются с помощью системного вызова clone() с указанием определенных флагов [1].

Системная функция fork() порождает полную копию родительского процесса, и эта процедура является затратной как с точки зрения расходования процессорного времени, так и с точки зрения расходования оперативной памяти. В параллельном программировании зачастую достаточно использования частичной копии процесса. Такая частичная копия называется потоком, «легковесным процессом» или нитью (thread). Нить ­­– это поток команд, который может самостоятельно управляться планировщиком. В отличие от дочернего процесса нить создается в контексте головного процесса, в том же самом адресном пространстве. На однопроцессорных системах планировщик чередует выполнение нитей посредством квантования процессорного времени, а на многопроцессорных системах нити могут выполняться параллельно на различных процессорах. Потоки не требуют специального механизма взаимодействия между подзадачами, поскольку они имеют равноправный доступ к общей памяти родительского процесса [2].

Программа создает нить, используя функцию pthread\_create(), затем назначает уникальный идентификатор переменной типа pthread\_t. Головная программа вызывает вычислительную функцию, которая будет выполняться как нить. Функция pthread\_exit() используется для завершения «легковесного» процесса. Функция pthread\_join() является аналогом функции wait(), но в этом случае любая нить может связываться с любой другой нитью в процессе, так как для нитей нет жесткой связи родитель-потомок.

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В результате лабораторной работы был написана программа, реализующая многопоточную сортировку для массива данных (рисунок 1). Количество потоков и размер массива задается пользователем. Количество потоков ограничено, что бы не провоцировало перегрузку системы. Также программа выводит время, за которое была выполнена сортировка.

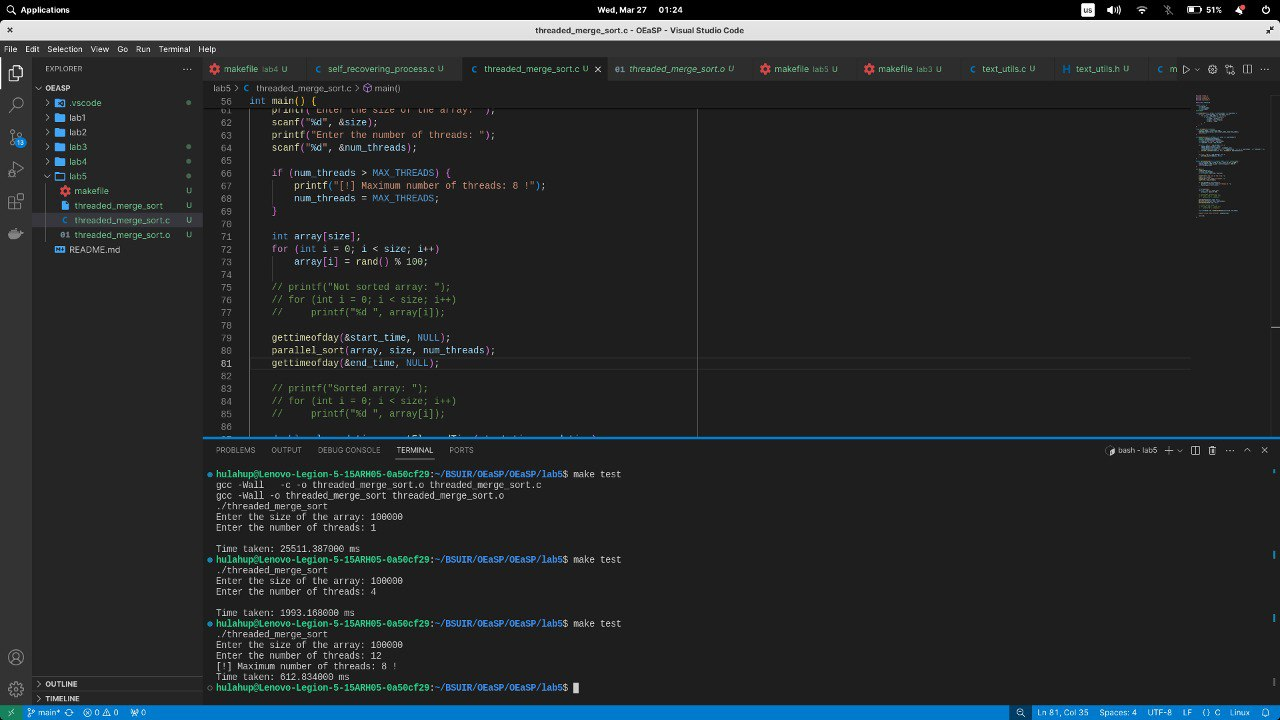


Рисунок 1 – Результаты выполнения программы

# ЗАКЛЮЧЕНИЕ

В ходе данной лабораторной работы были изучены подсистемы потоков (pthread), основные особенности функционирования и управления, средства взаимодействия потоков. Практическое проектирование, реализация и отладка программ с параллельными взаимодействующими (конкурирующими) потоками.

Была написана программа, реализующая многопоточную сортировку для большого массива данных. Количество потоков и размер массива задается пользователем. Количество потоков ограничено, что бы не провоцировало перегрузку системы. Также программа выводит время, за которое была выполнена сортировка.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Программирование C в Linux – потоки pthreads [Электронный ресурс]. – Режим доступа: https://tetraquark.ru/archives/47 – Дата доступа: 24.03.2024.
2. Многопоточность. Библиотека Pthread [Электронный ресурс]. – Режим доступа: http://rsuib.cc.rsu.ru/high\_performance\_computing/page08.html– Дата доступа: 24.03.2024.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Файл thread\_sort.c

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <sys/time.h>

#define MAX\_THREADS 8

typedef struct {

int \*array;

int start\_index;

int end\_index;

} pthrData;

void bubbleSort(int\* array, int start\_index, int end\_index) {

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

for (int j = end\_index; j > i; j--) {

if (array[j - 1] > array[j]) {

int temp = array[j - 1];

array[j - 1] = array[j];

array[j] = temp;

}

}

}

}

void merge(int \*array, int \*temp, int left, int mid, int right) {

int i = left, j = mid + 1, k = left;

while (i <= mid && j <= right) {

if (array[i] <= array[j])

temp[k++] = array[i++];

else

temp[k++] = array[j++];

}

while (i <= mid)

temp[k++] = array[i++];

while (j <= right)

temp[k++] = array[j++];

for (i = left; i <= right; ++i)

array[i] = temp[i];

}

void mergeSort(int \*array, int \*temp, int left, int right) {

if (left < right) {

int mid = (left + right) / 2;

mergeSort(array, temp, left, mid);

mergeSort(array, temp, mid + 1, right);

merge(array, temp, left, mid, right);

}

}

void\* threadSort(void\* arg) {

pthrData \*data = (pthrData\*) arg;

bubbleSort(data->array, data->start\_index, data->end\_index);

return NULL;

}

void parallel\_sort(int array[], int size, int num\_threads) {

pthread\_t threads[num\_threads];

pthrData thread\_data[num\_threads];

int segment\_size = size / num\_threads;

int remainder = size % num\_threads;

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

thread\_data[i].array = array;

thread\_data[i].start\_index = i \* segment\_size;

thread\_data[i].end\_index = (i + 1) \* segment\_size - 1 + (i == num\_threads - 1 ? remainder : 0);

pthread\_create(&threads[i], NULL, threadSort, &thread\_data[i]);

}

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

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

int \*temp = (int \*)malloc(size \* sizeof(int));

mergeSort(array, temp, 0, size - 1);

free(temp);

}

double getElapsedTime(struct timeval start, struct timeval end) {

double elapsed\_time = (end.tv\_sec - start.tv\_sec) \* 1000.0;

elapsed\_time += (end.tv\_usec - start.tv\_usec) / 1000.0;

return elapsed\_time;

}

int main() {

srand(time(NULL));

int size, num\_threads;

struct timeval start\_time, end\_time;

printf("Enter the size of the array: ");

scanf("%d", &size);

printf("Enter the number of threads: ");

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

if (num\_threads > MAX\_THREADS) {

printf("[!] Maximum number of threads: 8 !");

num\_threads = MAX\_THREADS;

}

int array[size];

for (int i = 0; i < size; i++)

array[i] = rand() % 100;

printf("Not sorted array: ");

for (int i = 0; i < size; i++)

printf("%d ", array[i]);

printf("\n");

gettimeofday(&start\_time, NULL);

parallel\_sort(array, size, num\_threads);

gettimeofday(&end\_time, NULL);

printf("Sorted array: ");

for (int i = 0; i < size; i++)

printf("%d ", array[i]);

printf("\n");

double elapsed\_time = getElapsedTime(start\_time, end\_time);

printf("\nTime taken: %f ms\n", elapsed\_time);

return 0;

}

Листинг 2 – Файл makefile

CC = gcc

CFLAGS = -Wall

OBJ = threaded\_sort.o

EXEC = threaded\_sort

$(EXEC): $(OBJ)

$(CC) $(CFLAGS) -o $@ $^

main.o:

$(CC) $(CFLAGS) -c threaded\_sort.c

clean:

rm -f $(OBJ) $(EXEC)

test: $(EXEC)

./$(EXEC)