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»

Выполнил             Е. А. Киселёва

Проверил                          Н. Ю. Гриценко

Минск 2024

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_Toc157722973)

[2 Краткие теоретические сведения 4](#_Toc157722974)

[3 Результаты выполнения лабораторной работы 5](#_Toc157722975)

[Выводы 6](#_Toc157722976)

[Список использованных источников 7](#_Toc157722977)

[Приложение А (обязательное) Листинг исходного кода 8](#_Toc157722978)

# **1 ПОСТАНОВКА ЗАДАЧИ**

Целью выполнения данной лабораторной работы является изучение подсистемы потоков, основных особенностей функционирования   
и управления, средств взаимодействия потоков. Кроме этого, необходимо реализовать программу на языке программирования С, которая будет реализовывать параллельную обработку блока данных различными потоками с использованием семафоров.

# **2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Поток выполнения представляет собой наименьшую единицу обработки, которую операционная система может назначить на выполнение. В рамках одного процесса может существовать несколько потоков выполнения, которые совместно используют ресурсы, такие как память, в отличие от процессов, которые не разделяют эти ресурсы. Особенно потоки выполнения разделяют инструкции процесса и значения его переменных в любой момент времени. [1]

В качестве потоков в системе Unix исполняются «облегченные» процессы. Эти облегченные процессы делят общие ресурсы с другими потоками в рамках одного процесса.[2]

Каждый поток обладает своим уникальным целочисленным идентификатором, известным как TID, который служит для определения потока в системе. Для основного потока процесса значения PID и TID совпадают, по сути, PID основного потока выступает как TID. Для остальных потоков процесса PID такой же, как у основного потока, в то время как значения TID индивидуальны. В многопоточных приложениях доступны стандартные средства синхронизации и обмена данными, такие как каналы, сокеты, семафоры и разделяемая память. [3]

Для выполнения данной лабораторной работы были использованы следующие сведения и концепции:

1 Разделяемая память: для работы с разделяемой памятью была использована структура данных, к которой обращались процессы. В коде программы отсутствует явная работа с разделяемой памятью. Программа использует многопоточность для обработки блоков данных различными потоками, а семафоры используются для обеспечения синхронизации доступа к общим данным.

2 Семафоры: для контроля доступа к разделяемой памяти каждого процесса были использованы семафоры, а также функции malloc   
для выделения определенного размера памяти под семафор, sem\_init   
для инициализации семафора, sem\_wait для уменьшения значения семафора на единицу, sem\_post для увеличения значения семафора на единицу, sem\_destroy для уничтожения семафора, free для освобождения выделенной памяти под семафор.

3 Управление потоками: при помощи функции pthread\_create создавался новый поток. При помощи функции pthread\_join блокировалось выполнение главного потока до тех пор, пока не завершатся все созданные раннее потоки. При помощи функции pthread\_exit завершается выполнение потоков   
и освобождаются ресурсы, связанные с ними.

# **3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ             РАБОТЫ**

В ходе лабораторной работы была реализована программа, в которой происходит создание нескольких потоков, заполняющих случайными значениями определенный блок данных, после чего родительский поток подсчитывает сумму этих данных. Результат работы программы представлен на рисунке 3.1.

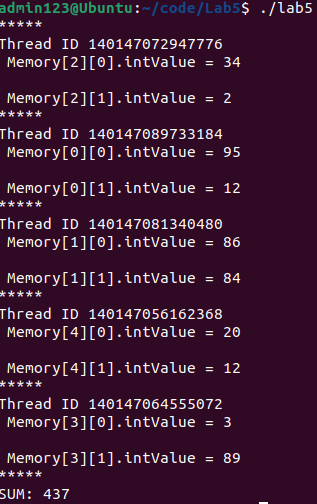


Рисунок 3.1 – Результат работы программы

Таким образом, в ходе лабораторной работы была реализована программа, реализующая заполнение блока данных случайными значениями и подсчет суммы этих значений.

# **ВЫВОДЫ**

В ходе лабораторной работы были изучены подсистема потоков, основные особенности функционирования и управления, средства взаимодействия потоков. Кроме этого, была реализована программа на языке программирования С, которая параллельно обрабатывает блок данных различными потоками, которые заполняют данный блок данных случайными числами, после чего родительский поток подсчитывает сумму этих значений.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Процессы и потоки [Электронный ресурс]. – Режим доступа: <https://acm.bsu.by/wiki/Unix2019b/>. – Дата доступа: 21.03.2024.

[2] Архитектура Unix. Процессы [Электронный ресурс]. – Режим доступа: https://acm.bsu.by/wiki/Unix2018/. – Дата доступа: 22.03.2024.

[3] Разделяемая память и семафоры [Электронный ресурс]. – Режим доступа: https://debianinstall.ru/razdelyaemaya-pamyat-semafory-i-ocheredi-soobshhenij-v-os-linux/. – Дата доступа: 23.03.2024.

## **ПРИЛОЖЕНИЕ А**

## **(обязательное)**

## **Листинг исходного кода**

Листинг 1 – Программный код lab5.c

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#include <time.h>

#define NUM\_BLOCKS 5

#define BLOCK\_SIZE 2

typedef struct {

int intValue;

} DataBlock;

sem\_t semaphores[NUM\_BLOCKS];

int sums[NUM\_BLOCKS];

void \*thread\_function(void \*args) {

int \*arg = (int \*)args;

int id = \*arg;

printf("\*\*\*\*\*\nThread ID %ld", pthread\_self());

srand(time(NULL) + id);

for (int j = 0; j < BLOCK\_SIZE; ++j) {

sem\_wait(&semaphores[id]);

int value = rand() % 100;

printf("\n Memory[%d][%d].intValue = %d\n", id, j, value);

sums[id] += value;

sem\_post(&semaphores[id]);

}

pthread\_exit(NULL);

}

int main() {

pthread\_t threads[NUM\_BLOCKS];

int thread\_args[NUM\_BLOCKS];

for (int i = 0; i < NUM\_BLOCKS; ++i) {

sem\_init(&semaphores[i], 1, 1);

thread\_args[i] = i;

sums[i] = 0;

}

for (int i = 0; i < NUM\_BLOCKS; ++i) {

pthread\_create(&threads[i], NULL, thread\_function, &thread\_args[i]);

}

for (int i = 0; i < NUM\_BLOCKS; ++i) {

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

}

int total\_sum = 0;

for (int i = 0; i < NUM\_BLOCKS; ++i) {

total\_sum += sums[i];

sem\_destroy(&semaphores[i]);

}

printf("\*\*\*\*\*\nSUM: %d\n", total\_sum);

return 0;

}