МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образование «Полоцкий государственный университет»

Факультет информационных технологий Кафедра технологий программирования

# Лабораторная работа №9

по дисциплине: «Системное программирование»

на тему:

«Потоки в ОС UNIX: создание и синхронизация потоков, анализ быстродействия последовательного и параллельного алгоритма»

Вариант №12

Выполнил Студент группы 21-ИТ-1

Чиникайло А.П.

Проверил Преподаватель

Сергеев М.А.

Полоцк 2023

**Задания**

1. Ознакомиться с теоретическим материалом.

2. Разработать три многопоточные программы с использованием минимум двух потоков и различных средств синхронизации. Например: два потока записывают и читают информацию из одного файла; два потока увеличивают значение общей переменной; два потока с различной частотой считывают и записывают данные в общий буфер памяти.

3. Необходимо обеспечить синхронизированную работу потоков в критической секции с использованием:

мьютексов;

семафоров;

условных переменных.

4. Убедиться в результативности применения средств синхронизации потоков, сравнив результаты работы программ с использованием и без использования средств синхронизации.

5. Написать ту же программу используя последовательный алгоритм и сравнить его скорость работы с параллельным.

**Не синхронизированные:**

#include <iostream>

#include <thread>

#include <mutex>

#include <chrono>

#include <semaphore.h>

#include <condition\_variable>

using namespace std;

int value = 0;

void MutexFunc(int count) {

cout << "Mutex was taken. Thread ID: " << this\_thread::get\_id() << endl << endl;

for(int i = 0; i < 1000000000; i++)

{

value += count;

}

cout << "Global value was changed to - " << value << endl << endl;

cout << "Mutex was given. Thread ID: " << this\_thread::get\_id() << endl << endl;

}

void SemaphoreFunc(int count) {

cout << "Semaphore was taken. Thread ID: " << this\_thread::get\_id() << endl << endl;

for(int i = 0; i < 1000000000; i++)

{

value += count;

}

cout << "Global value was changed to - " << value << endl << endl;

cout << "Semaphore was given. Thread ID: " << this\_thread::get\_id() << endl << endl;

}

void PeremFunc(int count) {

cout << "Perem was taken. Thread ID: " << this\_thread::get\_id() << endl << endl;

for(int i = 0; i < 1000000000; i++)

{

value += count;

}

cout << "Global value was changed to - " << value << endl << endl;

cout << "Perem was given. Thread ID: " << this\_thread::get\_id() << endl << endl;

}

int main() {

cout << "Main: " << this\_thread::get\_id() << endl << endl;

auto start = chrono::high\_resolution\_clock::now();

thread thread1(MutexFunc, 5);

MutexFunc(4);

thread1.join();

cout << "Value - " << value << endl << endl;

auto end = chrono::high\_resolution\_clock::now();

chrono::duration<double> mutexDuration = end - start;

cout << "//////////////////////////////////////////////////////////////////////////////////////////////" << endl << endl;

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

start = chrono::high\_resolution\_clock::now();

thread thread2(SemaphoreFunc, 5);

SemaphoreFunc(4);

thread2.join();

cout << "Value - " << value << endl << endl;

end = chrono::high\_resolution\_clock::now();

chrono::duration<double> semaphoreDuration = end - start;

cout << "//////////////////////////////////////////////////////////////////////////////////////////////" << endl << endl;

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

start = chrono::high\_resolution\_clock::now();

thread thread3(PeremFunc, 5);

PeremFunc(4);

thread3.join();

cout << "Value - " << value << endl << endl;

end = chrono::high\_resolution\_clock::now();

chrono::duration<double> peremDuration = end - start;

cout << "Mutex time: " << mutexDuration.count() << " sec." << endl;

cout << "Semaphore time: " << semaphoreDuration.count() << " sec." << endl;

cout << "Perem time: " << peremDuration.count() << " sec." << endl << endl;

cout << "Total time: " << peremDuration.count() + semaphoreDuration.count() + peremDuration.count() << " sec." << endl;

return 0;

}

**Синхронизированный:**

#include <iostream>

#include <thread>

#include <mutex>

#include <chrono>

#include <semaphore.h>

#include <condition\_variable>

using namespace std;

mutex mtx;

sem\_t semaphore;

condition\_variable perem;

bool isFree = true;

long int value = 0;

void MutexFunc(int count) {

mtx.lock();

cout << "Mutex was taken. Thread ID: " << this\_thread::get\_id() << endl << endl;

for(int i = 0; i < 1000000000; i++)

{

value += count;

}

cout << "Global value was changed to - " << value << endl << endl;

mtx.unlock();

cout << "Mutex was given. Thread ID: " << this\_thread::get\_id() << endl << endl;

}

void SemaphoreFunc(int count) {

sem\_wait(&semaphore);

cout << "Semaphore was taken. Thread ID: " << this\_thread::get\_id() << endl << endl;

for(int i = 0; i < 1000000000; i++)

{

value += count;

}

cout << "Global value was changed to - " << value << endl << endl;

sem\_post(&semaphore);

cout << "Semaphore was given. Thread ID: " << this\_thread::get\_id() << endl << endl;

}

void PeremFunc(int count) {

unique\_lock<mutex> lock(mtx);

perem.wait(lock, [] { return isFree; });

cout << "Perem was taken. Thread ID: " << this\_thread::get\_id() << endl << endl;

for(int i = 0; i < 1000000000; i++)

{

value += count;

}

cout << "Global value was changed to - " << value << endl << endl;

cout << "Perem was given. Thread ID: " << this\_thread::get\_id() << endl << endl;

}

int main() {

cout << "Main: " << this\_thread::get\_id() << endl << endl;

auto start = chrono::high\_resolution\_clock::now();

thread thread1(MutexFunc, 5);

MutexFunc(4);

thread1.join();

cout << "Value - " << value << endl << endl;

auto end = chrono::high\_resolution\_clock::now();

chrono::duration<double> mutexDuration = end - start;

cout << "//////////////////////////////////////////////////////////////////////////////////////////////" << endl << endl;

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

start = chrono::high\_resolution\_clock::now();

sem\_init(&semaphore, 0, 1);

thread thread2(SemaphoreFunc, 5);

SemaphoreFunc(4);

thread2.join();

cout << "Value - " << value << endl << endl;

end = chrono::high\_resolution\_clock::now();

chrono::duration<double> semaphoreDuration = end - start;

sem\_destroy(&semaphore);

cout << "//////////////////////////////////////////////////////////////////////////////////////////////" << endl << endl;

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

start = chrono::high\_resolution\_clock::now();

thread thread3(PeremFunc, 5);

PeremFunc(4);

thread3.join();

{

lock\_guard<mutex> lock(mtx);

isFree = true;

}

perem.notify\_one();

cout << "Value - " << value << endl << endl;

end = chrono::high\_resolution\_clock::now();

chrono::duration<double> peremDuration = end - start;

cout << "Mutex time: " << mutexDuration.count() << " sec." << endl;

cout << "Semaphore time: " << semaphoreDuration.count() << " sec." << endl;

cout << "Perem time: " << peremDuration.count() << " sec." << endl << endl;

cout << "Total time: " << mutexDuration.count() + semaphoreDuration.count() + peremDuration.count() << " sec." << endl;

return 0;

}

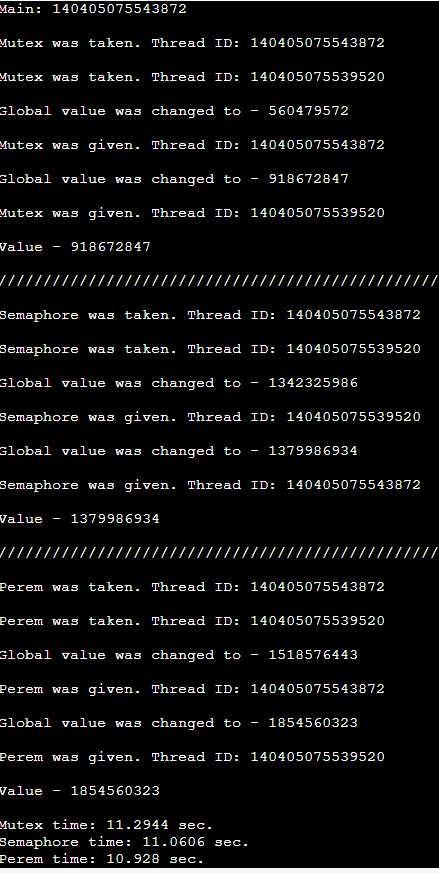
****

Рисунок 1 – Не синхронизированные потоки

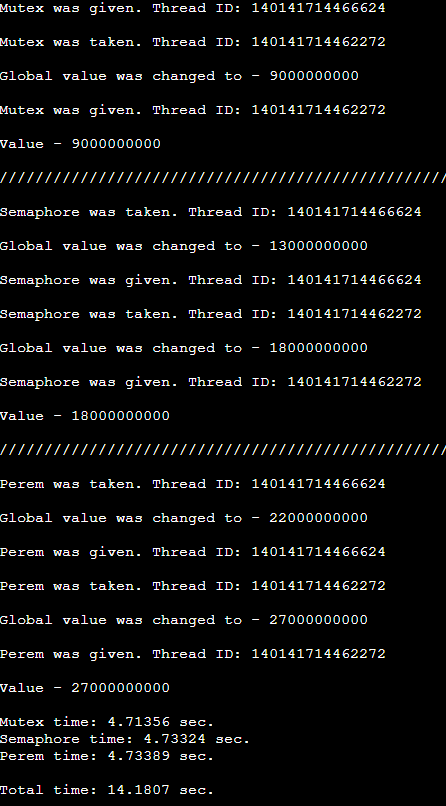
****

Рисунок 2 – Синхронизированные потоки

**Вывод:** в результате выполнения данной лабораторной работы были изучены способы синхронизации потоков, а так же обеспечена синхронизированная работа потоков в критической секции с использованием:

мьютексов;

семафоров;

условных переменных.