Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №5**

По дисциплине «Аппаратное обеспечение интеллектуальных систем»

Тема: «Разработка многопоточного приложения на основе библиотеки thread»

**Выполнил:**

Студент 2 курса

Группы ИИ-21

Кирилович А. А.

**Проверил:**

Михно Е.В.

Брест 2023

**Цель работы:** Изучить основные принципы работы с библиотекой thread. Получить навыки разработки многопоточных приложений на языке C++.

**Ход работы:**

**Вариант 7**

Написать на C++ программу, которая реализует многопоточную обработку данных согласно варианту. Отобразить на экране процесс и результат обработки данных, а так же отладочную информацию, связанную с передачей управления между потоками и параллельной обработкой данных.

Игра в 21. Одна нить генерирует случайные числа от 2 до 11; вторая и третья нити пытаются собрать последовательность суммой в 21 или меньше, получая последовательно числа от первой нити. Четвертая нить записывает в файл результат работы каждой нити (последовательность сгенерированных первой нитью чисел, набор чисел во второй и третьей нити, победителя). Побеждает в игре тот, кто набрал 21 или наиболее близкое значение, если у обоих перебор, то ничья. Провести не менее 10000 игр и вывести итоговый результат.

**Результат программы:**

#include <iostream>

#include <thread>

#include <fstream>

#include <vector>

#include <string>

#include <Windows.h>

#include <numeric>

using VEC1 = std::vector<int>;

HANDLE sem\_r, sem1, sem2, sem\_p;

void random\_numbers(int num\_games, VEC1& nums, bool& done1, bool& done2)

{

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

{

while (!done1)

{

nums.push\_back(rand() % 10 + 2);

ReleaseSemaphore(sem1, 1, NULL);

WaitForSingleObject(sem\_r, INFINITE);

}

while (!done2)

{

nums.push\_back(rand() % 10 + 2);

ReleaseSemaphore(sem2, 1, NULL);

WaitForSingleObject(sem\_r, INFINITE);

}

ReleaseSemaphore(sem\_p, 1, NULL);

WaitForSingleObject(sem\_r, INFINITE);

done1 = false, done2 = false;

}

}

void player1(int num\_games, VEC1& nums, VEC1& seq, bool& done)

{

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

{

int n, sum = 0;

while (true)

{

WaitForSingleObject(sem1, INFINITE);

n = nums.back();

ReleaseSemaphore(sem\_r, 1, NULL);

if (sum > 21 || (sum > 10 && rand() % 2)) break;

sum += n;

seq.push\_back(n);

}

done = true;

}

}

void player2(int num\_games, VEC1& nums, VEC1& seq, bool& done)

{

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

{

int n, sum = 0;

while (true)

{

WaitForSingleObject(sem2, INFINITE);

n = nums.back();

ReleaseSemaphore(sem\_r, 1, NULL);

if (sum > 21 || (sum > 10 && rand() % 2)) break;

sum += n;

seq.push\_back(n);

}

done = true;

}

}

void write\_vector\_to\_file(const VEC1& vec, std::ofstream& file)

{

for (auto it = vec.begin(); it != vec.end(); ++it)

{

file << \*it;

if (it != vec.end() - 1) file << ",";

}

file << "\n";

}

void print\_results(int num\_games, VEC1& rand\_nums, VEC1& plr1list, VEC1& plr2list, VEC1& results)

{

std::ofstream file("results.txt");

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

{

WaitForSingleObject(sem\_p, INFINITE);

write\_vector\_to\_file(rand\_nums, file);

write\_vector\_to\_file(plr1list, file);

write\_vector\_to\_file(plr2list, file);

int sum1 = std::accumulate(plr1list.begin(), plr1list.end(), 0);

int sum2 = std::accumulate(plr2list.begin(), plr2list.end(), 0);

if ((sum1 > 21 && sum2 > 21) || sum1 == sum2)

{

file << 0 << std::endl;

results[0]++;

}

else if ((sum1 <= 21 && sum2 > 21) || ((21 - sum1 < 21 - sum2) && sum1 <= 21 && sum2 <= 21))

{

file << 1 << std::endl;

results[1]++;

}

else

{

file << 2 << std::endl;

results[2]++;

}

rand\_nums.clear();

plr1list.clear();

plr2list.clear();

ReleaseSemaphore(sem\_r, 1, NULL);

}

file.close();

}

int main()

{

srand(time(NULL));

int num\_games = 10000;

VEC1 rand\_nums, plr1list, plr2list, results = {0, 0, 0};

bool done1 = false, done2 = false;

sem\_r = CreateSemaphore(NULL, 0, 1, NULL);

sem1 = CreateSemaphore(NULL, 0, 1, NULL);

sem2 = CreateSemaphore(NULL, 0, 1, NULL);

sem\_p = CreateSemaphore(NULL, 0, 1, NULL);

std::thread t1(random\_numbers, num\_games, std::ref(rand\_nums), std::ref(done1), std::ref(done2));

std::thread t2(player1, num\_games, std::ref(rand\_nums), std::ref(plr1list), std::ref(done1));

std::thread t3(player2, num\_games, std::ref(rand\_nums), std::ref(plr2list), std::ref(done2));

std::thread t4(print\_results, num\_games, std::ref(rand\_nums), std::ref(plr1list), std::ref(plr2list), std::ref(results));

t1.join();

t2.join();

t3.join();

t4.join();

CloseHandle(sem\_r);

CloseHandle(sem1);

CloseHandle(sem2);

CloseHandle(sem\_p);

for (int i = 0; i < results.size(); i++) std::cout << i << " : " << results[i] << std::endl;

}

**Вывод:** Изучил основные принципы работы с библиотекой thread. Получил навыки разработки многопоточных приложений на языке C++.