**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

Лабораторная работа №4

по курсу

«Операционные системы и сети»

Выполнили:

студенты группы ИКПИ-14

Хохлов Т.В.

Принял:

доцент кафедры ПИиВТ

Дагаев А.В

Санкт-Петербург

2023 г.

**Цель работы**

Целью работы является разработка многопроцессной программы на языке C++ с использованием библиотеки <Windows.h>.

# Описание программы

Программа является реализацией многопользовательской игры «15 палок». Программа состоит из двух файлов: player1.cpp и player2.cpp. После компиляции, программа запускается двумя параллельными процессами, синхронизированными выделенной памятью для передачи информации о текущем количестве палок и мьютексами для обеспечения правильности смены ходов.

Описание кода

**Основные компоненты:**

Файл **player1.cpp:**

**HANDLE sharedMemory**

**SharedData \*sharedData –** создание разделяемой памяти в виде структуры **SharedData** со свойствами **matchsticks (**количество оставшихся палок**), gameOver (**свойство-флаг завершения игры**).**

**HANDLE mutexPlayer1 –** создание мьютекса первого игрока.

**HANDLE mutexPlayer2 –** создание мьютекса второго игрока.

Файл **player2.cpp:**

**HANDLE sharedMemory**

**SharedData \*sharedData –** открытие разделяемой памяти, созданной процессом первого игрока в виде структуры **SharedData** со свойствами **matchsticks (**количество оставшихся палок**), gameOver (**свойство-флаг завершения игры**).**

**HANDLE mutexPlayer1 –** открытия мьютекса первого игрока.

**HANDLE mutexPlayer2 –** открытие мьютекса второго игрока.

Мьютексы используются для того, чтобы обеспечить правильную смену ходов и предотвратить одновременное изменение данных обеими сторонами.

**Принцип работы:**

1. **Ожидание подключения:**

Первым запускается процесс первого игрока – **Player1.cpp.**

**Player1.cpp** ожидает, пока **Player2.cpp** уведомит о своей готовности, блокируясь на ожидании мьютекса **mutexPlayer2**.

Затем, после запуска процесса второго игрока - **Player2.cpp,** второй игрок ожидает, пока **Player1.cpp** уведомит о своей готовности, блокируясь на ожидании мьютекса **mutexPlayer1**.

1. **Ход игрока:**

Player1.cpp делает свой ход (ввод игрока 1) и сигнализирует Player2.cpp, освобождая мьютекс mutexPlayer1.

Player2.cpp делает свой ход (ввод игрока 2) и сигнализирует Player1.cpp, освобождая мьютекс mutexPlayer2.

За ход игроки могут взять от 1 до 3 палок. Защита от дурака и неправильного ввода были реализованы.

1. **Проверка условия завершения игры:**

После каждого хода проверяется условие победы или поражения. Если условие выполнено, игра завершается, и соответствующему игроку выводится сообщение о победе.

1. **Завершение игры и освобождение ресурсов:**

По завершении игры оба процесса закрывают мьютексы и освобождают разделяемую память.

**Выводы**

В ходе проведённой лабораторной работы были получены следующие результаты:

1. Освоена библиотека **WINAPI** языка C++.

2. Создана программа с использованием параллельных процессов.

Исходный код программы

Player1.cpp:

#include <iostream>

#include <windows.h>

const char \*sharedMemoryName = "Local\\MatchsticksGameSharedMemory";

const char \*mutexNamePlayer1 = "Local\\Player1Mutex";

const char \*mutexNamePlayer2 = "Local\\Player2Mutex";

struct SharedData {

    int matchsticks;

    bool gameOver;

};

int main() {

    HANDLE sharedMemory = CreateFileMappingA(INVALID\_HANDLE\_VALUE, NULL, PAGE\_READWRITE, 0, sizeof(SharedData), sharedMemoryName);

    SharedData \*sharedData = static\_cast<SharedData\*>(MapViewOfFile(sharedMemory, FILE\_MAP\_READ | FILE\_MAP\_WRITE, 0, 0, sizeof(SharedData)));

    HANDLE mutexPlayer1 = CreateMutexA(NULL, FALSE, mutexNamePlayer1);

    HANDLE mutexPlayer2 = CreateMutexA(NULL, FALSE, mutexNamePlayer2);

    // Инициализация игры

    sharedData->matchsticks = 15;

    sharedData->gameOver = false;

    // Сигнализируем второму игроку о готовности

    ReleaseMutex(mutexPlayer1);

    // Ожидаем, пока второй игрок подключится

    std::cout << "Waiting for player2 to connect..." << std::endl;

    WaitForSingleObject(mutexPlayer2, INFINITE);

    // Начинаем игру

    while (true) {

        if (sharedData -> matchsticks == 0) {std::cout << "YOU WON!!!" << std::endl; break;}

        // Ваш код для хода первого игрока

        int player1Move;

        std::cout << "Player 1, enter your move (1-3 matchsticks): ";

        std::cin >> player1Move;

        // Проверка допустимости хода

        if (player1Move < 1 || player1Move > 3 || player1Move > sharedData->matchsticks) {

            std::cout << "Invalid move. Try again." << std::endl;

            player1Move = 0;

        }

        // Обновляем количество спичек

        sharedData->matchsticks -= player1Move;

        // Печать текущего состояния игры

        std::cout << "Matchsticks left: " << sharedData->matchsticks << std::endl;

        // Проверка условия победы/проигрыша

        if (sharedData->matchsticks == 0) {

            std::cout << "Player 2 wins!" << std::endl;

            sharedData->gameOver = true;

            break;

        }

        // Сигнализируем второму игроку о том, что ход первого игрока завершен

        ReleaseMutex(mutexPlayer1);

        // Ожидаем хода второго игрока

        std::cout << "Waiting for player2 to make a move..." << std::endl;

        WaitForSingleObject(mutexPlayer2, INFINITE);

        // Проверка условия победы/проигрыша

        if (sharedData->matchsticks == 0) {

            std::cout << "Player 2 wins!" << std::endl;

            sharedData->gameOver = true;

            break;

        }

    }

    // Закрываем мьютексы и разделяемую память

    CloseHandle(mutexPlayer1);

    CloseHandle(mutexPlayer2);

    UnmapViewOfFile(sharedData);

    CloseHandle(sharedMemory);

    return 0;

}

Player2.cpp:

#include <iostream>

#include <windows.h>

const char \*sharedMemoryName = "Local\\MatchsticksGameSharedMemory";

const char \*mutexNamePlayer1 = "Local\\Player1Mutex";

const char \*mutexNamePlayer2 = "Local\\Player2Mutex";

struct SharedData {

    int matchsticks;

    bool gameOver;

};

int main() {

    HANDLE sharedMemory = OpenFileMappingA(FILE\_MAP\_READ | FILE\_MAP\_WRITE, FALSE, sharedMemoryName);

    SharedData \*sharedData = static\_cast<SharedData\*>(MapViewOfFile(sharedMemory, FILE\_MAP\_READ | FILE\_MAP\_WRITE, 0, 0, sizeof(SharedData)));

    HANDLE mutexPlayer1 = OpenMutexA(SYNCHRONIZE, FALSE, mutexNamePlayer1);

    HANDLE mutexPlayer2 = OpenMutexA(SYNCHRONIZE, FALSE, mutexNamePlayer2);

    // Сигнализируем первому игроку о готовности

    ReleaseMutex(mutexPlayer2);

    // Ожидаем, пока первый игрок подключится

    std::cout << "Waiting for player1 to connect..." << std::endl;

    WaitForSingleObject(mutexPlayer1, INFINITE);

    // Начинаем игру

    while (true) {

        if (sharedData -> matchsticks == 0) {std::cout << "YOU WON!!!" << std::endl; break;}

        // Ваш код для хода второго игрока

        int player2Move;

        std::cout << "Player 2, enter your move (1-3 matchsticks): ";

        std::cin >> player2Move;

        // Проверка допустимости хода

        if (player2Move < 1 || player2Move > 3 || player2Move > sharedData->matchsticks) {

            std::cout << "Invalid move. Try again." << std::endl;

            player2Move = 0;

        }

        // Обновляем количество спичек

        sharedData->matchsticks -= player2Move;

        // Печать текущего состояния игры

        std::cout << "Matchsticks left: " << sharedData->matchsticks << std::endl;

        // Проверка условия победы/проигрыша

        if (sharedData->matchsticks == 0) {

            std::cout << "Player 1 wins!" << std::endl;

            sharedData->gameOver = true;

            break;

        }

        // Сигнализируем первому игроку о том, что ход второго игрока завершен

        ReleaseMutex(mutexPlayer2);

        // Ожидаем хода первого игрока

        std::cout << "Waiting for player1 to make a move..." << std::endl;

        WaitForSingleObject(mutexPlayer1, INFINITE);

        // Проверка условия победы/проигрыша

        if (sharedData->matchsticks == 0) {

            std::cout << "Player 1 wins!" << std::endl;

            sharedData->gameOver = true;

            break;

        }

    }

    // Закрываем мьютексы и разделяемую память

    CloseHandle(mutexPlayer1);

    CloseHandle(mutexPlayer2);

    UnmapViewOfFile(sharedData);

    CloseHandle(sharedMemory);

    return 0;

}