Звіт

до лабораторної роботи 5

з навчальної дисципліни «Операційні системи»

студента 3 курсу ФКНК групи МІ-32

Красікова Антона Ігоровича

“Inter-Process Communications (IPC) + Threads”

Умова завдання:

Взаємодія процесів. Паралелізм. Обмін повідомленнями через порт. Обчислити f(x) || g(x), використовуючи 2 допоміжні процеси: один обчислює f(x), а інший – g(x). Основна програма виконує ввод-вивід та операцію ||. Використати обмін повідомленнями між процесами через порт (Socket). Реалізувати варіант блокуючих операцій обміну повідомленнями, тобто з очікуванням обробки повідомлення і відповіді на повідомлення (і “зависанням” процесу на цей час). Функції f(x) та g(x) “нічого не знають друг про друга” і не можуть комунікувати між собою.

Виконання:

Для обчислення f(x) та g(x) використовується один вихідний файл та одна скомпільована програма func.cpp, якій у командному рядку вказується номер порту для прослуховування, та яку саме функцію обчислювати — f чи g. Програма підключається до вказаного порту і очікує вхідного з’єднання. Після цього вона отримує значення x, обчислює результат та надсилає його, після чого закриває сокет та завершує роботу (така поведінка дозволяється умовою завдання).

Головна програма за допомогою функції _spawnl запускає на виконання два екземпляри програми func з різними портами. Після цього вона запускає на виконання два потоки, які мають передати значення x та отримати результати обчислень. Для них викликається функція .detach().

Для синхронізації виводу потоками використовується м’ютекс; для отриманих результатів та flag отримання — атомарні змінні. Це дозволяє організувати основний потік як цикл, що очікує отримання результатів потоками, які при отриманні результату виставляють відповідне значення flag true.

Якщо отримано обидва результати, або хоча б один, який дорівнює true (тобто значення другого вже непотрібне: результат операції || все одно буде true), здійснюється відповідний вивід та вихід із програми, яка також «вбиває» процеси func, на той випадок, якщо вони ще не завершилися.

Кожні 10 секунд чекання програма запитує, що треба робити далі і відповідно обо завершує роботу, або продовжує її (у випадку «більше не нагадувати» надавши відповідне значення змінній-флагу).

Код програми func.cpp

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <thread>
#include <winsock2.h>
#pragma comment(lib,"ws2_32.lib")


using namespace std;

// Function f
int f(int x)
{
    this_thread::sleep_for(28s);
    return 421;
}
// Function g
int g(int x)
{
    this_thread::sleep_for(3s);
    return 0;
}
```

```
int main(int argc, char * argv[])
{
    if (argc < 3)
    {
        cerr << "Usage: funt port f|g\n";
        return 1;
    }
    // Command-line options
    unsigned short int port = (unsigned short int)atoi(argv[1]);
    int (*func)(int) = (argv[2][0] == 'f') ? f : g;

    SOCKET ms = INVALID_SOCKET, cs = INVALID_SOCKET;
    try {
        WORD wVersionRequested = MAKEWORD(1, 1);
        WSADATA wsaData;
        if (WSAStartup(wVersionRequested, &wsaData) != 0)
            throw runtime_error("WSAStartup");

        // Socket creation
        ms = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
        if (ms == INVALID_SOCKET) throw runtime_error("socket creation");

        // bind to port
        sockaddr_in name;
        memset(&name, 0, sizeof(name));
        name.sin_addr.s_addr = INADDR_ANY;
        name.sin_family = AF_INET;
        name.sin_port = htons(port);
        if (SOCKET_ERROR == bind(ms, (SOCKADDR*)&name, sizeof(name)))
            throw runtime_error("bind");

        if (listen(ms, 1) != 0) throw runtime_error("listen");

        // Wait for connect
        cs = accept(ms, NULL, NULL);
        if (cs == INVALID_SOCKET) throw runtime_error("accept");

        // Receive x and ...
        int x;
        int rc = recv(cs, (char*)&x, sizeof(x), 0);
        if (rc < 0) throw runtime_error("recv data");
        if (rc == 0) throw runtime_error("connection closed");;

        // ...caclulation...
        rc = func(x);

        // ...and send answer
        rc = send(cs, (char*)&rc, sizeof(rc), 0);
        if (rc != sizeof(rc)) throw("send data");

        closesocket(cs);
    }
    catch (exception& e)
    {
        cerr << "Error: " << e.what() << endl;
    }
    closesocket(ms);
    closesocket(cs);
    WSACleanup();
}
```

Код програми main.cpp:

```
#define  _CRT_SECURE_NO_WARNINGS
#define  _WINSOCK_DEPRECATED_NO_WARNINGS
#include <iostream>
#include <winsock2.h>
#include <thread>
#include <mutex>

#pragma comment(lib,"ws2_32.lib")

using namespace std;

mutex outmutex;  // Mutex for outputs

// flags "received" and received data
atomic<bool> received[2] = { false, false };
atomic<bool> result[2]   = { false, false };


// thread function
void getValue(int x, unsigned short int port, int index)
{
    // Create socket
    SOCKET s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (s == INVALID_SOCKET) throw runtime_error("socket creation");

    // Call
    sockaddr_in name;
    memset(&name, 0, sizeof(name));
    name.sin_addr.s_addr = inet_addr("127.0.0.1");
    name.sin_family = AF_INET;
    name.sin_port = htons(port);
    if (SOCKET_ERROR == connect(s, (SOCKADDR*)&name, sizeof(name)))
        throw runtime_error("connect");

    {
        lock_guard<mutex> lock(outmutex);
        cout << "Send " << x << " to port " << port << "\n";
    }
    // Send x
    if (send(s,(char*)&x,sizeof(x),0) != sizeof(x))
        throw runtime_error("send");

    // Receive result
    if (recv(s,(char*)&x,sizeof(x),0) != sizeof(x))
        throw runtime_error("recv");

    {
        lock_guard<mutex> lock(outmutex);
        cout << "Recv " << x << " from port " << port << "\n";
    }
    // Write result and flag
    result[index]   = bool(x);
    received[index] = true;
    closesocket(s);
}

int main(int argc, char * argv[])
{
    int x = 1, y = 2;
    // x and y can be passed on the command line
    if (argc > 1) x = atoi(argv[1]);
    if (argc > 2) y = atoi(argv[2]);

    // Run f(x) and g(x) processes
```

```
unsigned short int portF = 20120, portG = 20121;
char portS[20];
sprintf(portS,"%hd",portF);
_spawnl(_P_NOWAIT,"func.exe","func.exe",portS,"f",0);
sprintf(portS,"%hd",portG);
_spawnl(_P_NOWAIT,"func.exe","func.exe",portS,"g",0);

WORD wVersionRequested = MAKEWORD(1, 1);
WSADATA wsaData;
try {

    if (WSAStartup(wVersionRequested, &wsaData) != 0)
        throw runtime_error("WSAStartup");

    // Threads for f(x) and g(x)
    thread(getValue,x,portF,0).detach();
    thread(getValue,y,portG,1).detach();

    int Result = -1;       // -1 => no result yet
    bool quest = true;

    for(int i = 1;;++i)
    {
        // is received data enough?
        if (received[0] && received[1])
        {
            Result = result[0] || result[1];
        }
        else if (received[0] && result[0])
        {
            Result = 1;
        }
        else if (received[1] && result[1])
        {
            Result = 1;
        }

        if (Result != -1) // Yes, we have result
        {
            lock_guard<mutex> lock(outmutex);
            cout << "f(" << x << ") || g(" << y << ") == "
                << (Result ? "true\n" : "false\n");
            break;
        }

        // No, result not ready. Wait 1 s
        this_thread::sleep_for(1s);
        // Every 10 s if flag 'quest' is true
        if (i%10 == 0 && quest)
        {
            lock_guard<mutex> lock(outmutex);
            cout << "Too long wait. 1 - abort, "
                    "2 - retry, 3 - no more questions: ";
            int n;
            cin >> n;
            if (n == 3) quest = false;           // no more qurestions
            else if (n == 2) cout << "retry\n";  // retry
            else if (n == 1) throw runtime_error("user choose abort");
            else cout << "wrong choice, retry\n";
        }
    }

}
catch (exception& e)
```

```
    {
        cerr << "Error: " << e.what() << endl;
    }
    cout << "Bye!\n";
    WSACleanup();
    system("taskkill /IM func.exe /F >nul 2>&1");
}
```