# Комп'ютерний практикум №5
# Можливості синхронізації багатозадачності в середовищі Windows

**Виконав:**
Студент 2 курсу ФТІ
групи ФІ-92
Поночевний Назар Юрійович

**Мета:** Реалізувати програмну синхронізацію потоків відповідно до завдання.

**Завдання 10 (54):**
Написати багатопоточний застосунок, який моделює роботу готелю. Потоки-клієнти роблять запит на бронювання. Потоки-адміністратори готелю перевіряють наявність місць. Якщо вільний номер є, потік-клієнт займає на K<N ночей, якщо відповідного місця нема, клієнту пропонують більш дорогий номер. Якщо клієнт відмовляється, Потік-клієнт залишає готель.

1. Створити головний процес. Створити 2 дочірні процеси, віддати наказ на обробку та отримати і відобразити результат обробки.
2. Першому дочірньому процесу: Створити ресурси для роботи потоків, наприклад файли для обробки та реалізувати обмін даними по іменованим каналам, якщо необхідно.
3. Другому дочірньому процесу: утворити Потоки для роботи з файлом. Обробка файлу реалізується потоками другого дочірнього процесу, що синхронізуються через засоби синхронізації.
4. Використовуючи компілятор C++, реалізувати програму в середовищі Windows синхронізації згідно варіанту. Під час роботи програми треба весь час виводити інформацію про те як працюють потоки (у файл, або реалізувати графічну візуалізацію). Якщо у Вашому завданні мова йде про розподіл ресурсів між потоками треба фіксувати звільнення і зайняття ресурсів потоками. При використанні механізму подій та семафорів треба фіксувати переходи у вільні стани (події) та переходи у сигнальний стан (таймери).

Код (Main Process):

```
#include <tchar.h>
#include <windows.h>
#include <stdio.h>
#include <strsafe.h>

#define BUFSIZE 4096
```

```
DWORD WINAPI InstanceThread(LPVOID);
VOID GetAnswerToRequest(LPTSTR, LPTSTR, LPDWORD);

DWORD MainProcessId = GetCurrentProcessId();

void CreateChild1Process() {
    TCHAR applicationName[] = TEXT("\"D:\\Microsoft Visual
Studio\\Workspace\\SysProga\\Lab5_1\\Debug\\Lab5_1.exe\" \"D:\\Microsoft
Visual Studio\\Workspace\\SysProga\\Lab5_1\\one.txt\" \"D:\\Microsoft
Visual Studio\\Workspace\\SysProga\\Lab5_1\\two.txt\"");
    STARTUPINFO si;
    PROCESS_INFORMATION pi;

    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory(&pi, sizeof(pi));

    // Start the child process.
    if (!CreateProcess(NULL,
        applicationName,
        NULL,
        NULL,
        FALSE,
        0,
        NULL,
        NULL,
        &si,
        &pi)
        )
    {
        printf("CreateProcess failed (%d).\n", GetLastError());
        return;
    }

    // Close process and thread handles.
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
}

void CreateChild2Process() {
    TCHAR applicationName[] = TEXT("\"D:\\Microsoft Visual
Studio\\Workspace\\SysProga\\Lab5_2\\Debug\\Lab5_2.exe\" \"D:\\Microsoft
Visual Studio\\Workspace\\SysProga\\Lab5_1\\two.txt\"");
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
```

```
    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory(&pi, sizeof(pi));

    // Start the child process.
    if (!CreateProcess(NULL,
        applicationName,
        NULL,
        NULL,
        FALSE,
        0,
        NULL,
        NULL,
        &si,
        &pi)
        )
    {
        printf("CreateProcess failed (%d).\n", GetLastError());
        return;
    }

    // Close process and thread handles.
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
}

int _tmain(VOID)
{
    BOOL   fConnected = FALSE;
    DWORD  dwThreadId = 0;
    HANDLE hPipe = INVALID_HANDLE_VALUE, hThread = NULL;
    LPCTSTR lpszPipename = TEXT("\\\\.\\pipe\\mynamedpipe");

    printf("(%d) Parent process running.... \n\n", MainProcessId);

    CreateChild1Process();
    CreateChild2Process();


    _tprintf(TEXT("\n(%d) Pipe Server: Main thread awaiting client
connection on %s\n\n"), MainProcessId, lpszPipename);

    for (;;)
    {
        hPipe = CreateNamedPipe(
            lpszPipename,
```

```
            PIPE_ACCESS_DUPLEX,
            PIPE_TYPE_MESSAGE |
            PIPE_READMODE_MESSAGE |
            PIPE_WAIT,
            PIPE_UNLIMITED_INSTANCES,
            BUFSIZE,
            BUFSIZE,
            0,
            NULL);

        if (hPipe == INVALID_HANDLE_VALUE)
        {
            _tprintf(TEXT("CreateNamedPipe failed, GLE=%d.\n"),
GetLastError());
            return -1;
        }

        fConnected = ConnectNamedPipe(hPipe, NULL) ?
            TRUE : (GetLastError() == ERROR_PIPE_CONNECTED);

        if (fConnected)
        {
            printf("\n\n(%d) Client connected, creating a processing
thread.\n", MainProcessId);

            // Create a thread for this client.
            hThread = CreateThread(
                NULL,
                0,
                InstanceThread,
                (LPVOID)hPipe,
                0,
                &dwThreadId);

            if (hThread == NULL)
            {
                _tprintf(TEXT("CreateThread failed, GLE=%d.\n"),
GetLastError());
                return -1;
            }
            else CloseHandle(hThread);
        }
        else
            CloseHandle(hPipe);
    }
```

```
    return 0;
}

DWORD WINAPI InstanceThread(LPVOID lpvParam)
{
    HANDLE hHeap = GetProcessHeap();
    TCHAR* pchRequest = (TCHAR*)HeapAlloc(hHeap, 0, BUFSIZE *
sizeof(TCHAR));
    TCHAR* pchReply = (TCHAR*)HeapAlloc(hHeap, 0, BUFSIZE *
sizeof(TCHAR));

    DWORD cbBytesRead = 0, cbReplyBytes = 0, cbWritten = 0;
    BOOL fSuccess = FALSE;
    HANDLE hPipe = NULL;

    if (lpvParam == NULL)
    {
        printf("\nERROR - Pipe Server Failure:\n");
        printf("   InstanceThread got an unexpected NULL value in
lpvParam.\n");
        printf("   InstanceThread exitting.\n");
        if (pchReply != NULL) HeapFree(hHeap, 0, pchReply);
        if (pchRequest != NULL) HeapFree(hHeap, 0, pchRequest);
        return (DWORD)-1;
    }

    if (pchRequest == NULL)
    {
        printf("\nERROR - Pipe Server Failure:\n");
        printf("   InstanceThread got an unexpected NULL heap
allocation.\n");
        printf("   InstanceThread exitting.\n");
        if (pchReply != NULL) HeapFree(hHeap, 0, pchReply);
        return (DWORD)-1;
    }

    if (pchReply == NULL)
    {
        printf("\nERROR - Pipe Server Failure:\n");
        printf("   InstanceThread got an unexpected NULL heap
allocation.\n");
        printf("   InstanceThread exitting.\n");
        if (pchRequest != NULL) HeapFree(hHeap, 0, pchRequest);
        return (DWORD)-1;
    }
```

```
    hPipe = (HANDLE)lpvParam;

    while (1)
    {
        fSuccess = ReadFile(
            hPipe,
            pchRequest,
            BUFSIZE * sizeof(TCHAR),
            &cbBytesRead,
            NULL);

        if (!fSuccess || cbBytesRead == 0)
        {
            if (GetLastError() == ERROR_BROKEN_PIPE)
            {
                _tprintf(TEXT("\n(%d) InstanceThread: client
disconnected.\n"), MainProcessId);
            }
            else
            {
                _tprintf(TEXT("InstanceThread ReadFile failed,
GLE=%d.\n"), GetLastError());
            }
            break;
        }

        GetAnswerToRequest(pchRequest, pchReply, &cbReplyBytes);
    }

    FlushFileBuffers(hPipe);
    DisconnectNamedPipe(hPipe);
    CloseHandle(hPipe);

    HeapFree(hHeap, 0, pchRequest);
    HeapFree(hHeap, 0, pchReply);

    return 1;
}

VOID GetAnswerToRequest(LPTSTR pchRequest,
    LPTSTR pchReply,
    LPDWORD pchBytes)
{
    _tprintf(TEXT("\n(%d) Client Request String:\n    \"%s\"\n"),
MainProcessId, pchRequest);
```

```
    // Check the outgoing message to make sure it's not too long for the
buffer.
    if (FAILED(StringCchCopy(pchReply, BUFSIZE, TEXT("default answer
from server"))))
    {
        *pchBytes = 0;
        pchReply[0] = 0;
        printf("StringCchCopy failed, no outgoing message.\n");
        return;
    }
    *pchBytes = (lstrlen(pchReply) + 1) * sizeof(TCHAR);
}
```

Код (Child 1 Process):

```
#include <tchar.h>
#include <windows.h>
#include <stdio.h>
#include <strsafe.h>

#define BUFSIZE 4096

DWORD Child1ProcessId = GetCurrentProcessId();

int _tmain(int argc, TCHAR* argv[])
{
    HANDLE hFile;
    HANDLE hAppend;
    DWORD  dwBytesRead, dwBytesWritten, dwPos;
    BYTE   buff[BUFSIZE];

    if (argc != 3)
    {
        _tprintf(TEXT("Usage: %s <data file> <copy file>\n"), argv[0]);
        return 1;
    }

    printf("(%d) Child1 process running.... \n", Child1ProcessId);

    // Create an Event.
    HANDLE ghWriteEvent = CreateEvent(
        NULL,
        TRUE,
        FALSE,
```

```
        TEXT("WriteEvent")
    );

    if (ghWriteEvent == NULL)
    {
        printf("CreateEvent failed (%d)\n", GetLastError());
        return 1;
    }

    printf("(%d) Sleep for 5 secs... \n", Child1ProcessId);
    Sleep(5000);

    // Connect to Named Pipe.
    HANDLE hPipe;
    TCHAR  chBuf[BUFSIZE];
    BOOL   fSuccess = FALSE;
    DWORD  cbRead, cbToWrite, cbWritten, dwMode;
    LPCTSTR lpszPipename = TEXT("\\\\.\\pipe\\mynamedpipe");

    while (1)
    {
        hPipe = CreateFile(
            lpszPipename,
            GENERIC_READ |
            GENERIC_WRITE,
            0,
            NULL,
            OPEN_EXISTING,
            0,
            NULL);

        if (hPipe != INVALID_HANDLE_VALUE)
            break;

        if (GetLastError() != ERROR_PIPE_BUSY)
        {
            _tprintf(TEXT("(%d) Could not open pipe. GLE=%d\n"),
Child1ProcessId, GetLastError());
            return -1;
        }

        if (!WaitNamedPipe(lpszPipename, 20000))
        {
            printf("(%d) Could not open pipe: 20 second wait timed
out.", Child1ProcessId);
            return -1;
```

```
        }
    }

    dwMode = PIPE_READMODE_MESSAGE;
    fSuccess = SetNamedPipeHandleState(
        hPipe,
        &dwMode,
        NULL,
        NULL);

    if (!fSuccess)
    {
        _tprintf(TEXT("SetNamedPipeHandleState failed. GLE=%d\n"),
GetLastError());
        return -1;
    }

    // Open the existing file.

    hFile = CreateFile(argv[1],
        GENERIC_READ,
        0,
        NULL,
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL,
        NULL);

    if (hFile == INVALID_HANDLE_VALUE)
    {
        printf("Could not open One.txt.");
        return 1;
    }

    // Create a new file.

    hAppend = CreateFile(argv[2],
        FILE_WRITE_DATA,
        FILE_SHARE_READ,
        NULL,
        CREATE_ALWAYS,
        FILE_ATTRIBUTE_NORMAL,
        NULL);

    if (hAppend == INVALID_HANDLE_VALUE)
    {
        printf("Could not open Two.txt.");
```

```
        return 1;
    }

    while (ReadFile(hFile, buff, sizeof(buff), &dwBytesRead, NULL)
        && dwBytesRead > 0)
    {
        dwPos = SetFilePointer(hAppend, 0, NULL, FILE_END);
        LockFile(hAppend, dwPos, 0, dwBytesRead, 0);
        WriteFile(hAppend, buff, dwBytesRead, &dwBytesWritten, NULL);
        UnlockFile(hAppend, dwPos, 0, dwBytesRead, 0);
    }

    // Close both files.

    CloseHandle(hFile);
    CloseHandle(hAppend);

    // Set ghWriteEvent to signaled.

    if (!SetEvent(ghWriteEvent))
    {
        printf("SetEvent failed (%d)\n", GetLastError());
        return 1;
    }
    CloseHandle(ghWriteEvent);

    // Send a message to the pipe server.

    TCHAR lpvMessage[MAX_PATH];
    StringCchPrintf(lpvMessage, MAX_PATH, TEXT("(%d) Created file %s"),
Child1ProcessId, argv[2]);

    cbToWrite = (lstrlen(lpvMessage) + 1) * sizeof(TCHAR);

    fSuccess = WriteFile(
        hPipe,
        lpvMessage,
        cbToWrite,
        &cbWritten,
        NULL);

    if (!fSuccess)
    {
        _tprintf(TEXT("(%d) WriteFile to pipe failed. GLE=%d\n"),
Child1ProcessId, GetLastError());
        return -1;
```

```
    }

    // Close Pipe.
    CloseHandle(hPipe);

    return 0;
}
```

Код (Child 2 Process):

```
#include <tchar.h>
#include <windows.h>
#include <stdio.h>
#include <strsafe.h>
#include <atlstr.h>
#include <vector>
#include <string>
#include <sstream>
#include <stdlib.h>

using namespace std;

#define BUFSIZE 4096
#define MAX_ADMINS 5
#define MAX_THREADS 50
#define MAX_ROOMS 15

typedef long long int mInt;

struct Request {
    string clientName = "";
    mInt desiredPrice = 0;
    mInt vacStart = 0;
    mInt vacDuration = 1;
};
struct Response {
    string adminName = "";
    mInt status = -1;
    mInt rNumber = 0;
    mInt rPrice = 0;
};

class Room {
public:
```

```
    Room(mInt num = 0, mInt fare = 10, string strSch = "");
    ~Room();
    bool isFree(mInt start, mInt duration = 1);
    void book(mInt start, mInt duration = 1);
    mInt number;
    mInt price;
protected:
    vector<mInt> sheet;
    mInt sheetLen = 0;
    HANDLE ghRoomMutex;
};
Room::Room(mInt num, mInt fare, string strSch) {
    ghRoomMutex = CreateMutex(
        NULL,
        FALSE,
        NULL);
    if (ghRoomMutex == NULL)
    {
        printf("CreateMutex error: %d\n", GetLastError());
    }
    number = num;
    price = fare;
    if (strSch.size() > 0) {
        string buf;
        stringstream ss(strSch);
        vector<string> strs;
        while (ss >> buf)
            strs.push_back(buf);
        if (strs.size() > 0) {
            sheetLen = stoi(strs[0]);
            mInt strs_len = sheetLen * 2;
            strs.erase(strs.begin());
            for (mInt i = 0; i < strs_len; i++) {
                sheet.push_back(stoi(strs[i]));
            }
        }
    }
}
Room::~Room() {
    CloseHandle(ghRoomMutex);
}
bool Room::isFree(mInt start, mInt duration) {
    DWORD dwWaitResult = WaitForSingleObject(
        ghRoomMutex,
        INFINITE);
    if (sheetLen > 0) {
```

```
        mInt loopLen = sheetLen * 2;
        for (mInt i = 0; i < loopLen; i += 2) {
            if (start + duration > sheet[i] && start < sheet[i] +
sheet[i + 1])
                return FALSE;
        }
    }
    return TRUE;
    ReleaseMutex(ghRoomMutex);
}
void Room::book(mInt start, mInt duration) {
    DWORD dwWaitResult = WaitForSingleObject(
        ghRoomMutex,
        INFINITE);
    sheetLen++;
    sheet.push_back(start);
    sheet.push_back(duration);
    ReleaseMutex(ghRoomMutex);
}

Room rooms[MAX_ROOMS];

class Admin {
public:
    string name;
    Admin(string n);
    Response check(Request req);
    bool book(string clientName, mInt number, mInt start, mInt duration
= 1);
};
Admin::Admin(string n) {
    name = n;
}
Response Admin::check(Request req) {
    Response res;
    printf("[Admin %s -> Client %s] Searching room for %lld price from
%lld during %lld period... \n", name.c_str(), req.clientName.c_str(),
req.desiredPrice, req.vacStart, req.vacDuration);
    for (mInt i = 0; i < MAX_ROOMS; i++) {
        Sleep(500);
        Room& room = rooms[i];
        if (room.isFree(req.vacStart, req.vacDuration)) {
            if (room.price > req.desiredPrice) {
                printf("[Admin %s -> Client %s] Room %lld is free, but
has greater price: %lld. Are you agree? \n", name.c_str(),
req.clientName.c_str(), room.number, room.price);
```

```
                res = { name, 0, room.number, room.price };
                return res;
            }
            printf("[Admin %s -> Client %s] Room %lld is free and has
affortable price: %lld. Are you agree? \n", name.c_str(),
req.clientName.c_str(), room.number, room.price);
            res = { name, 1, room.number, room.price };
            return res;
        }
        printf("Room %lld is already occupied for this period. \n",
room.number);
    }
    res.adminName = name;
    res.status = -1;
    return res;
}
bool Admin::book(string clientName, mInt number, mInt start, mInt
duration) {
    for (mInt i = 0; i < MAX_ROOMS; i++) {
        Sleep(500);
        Room& room = rooms[i];
        if (room.number == number) {
            if (room.isFree(start, duration)) {
                room.book(start, duration);
                printf("[Admin %s -> Client %s] Room %lld is
successfully booked from %lld during %lld period. \n", name.c_str(),
clientName.c_str(), room.number, start, duration);
                return TRUE;
            }
            printf("Room %lld is already occupied for this period. \n",
room.number);
            return FALSE;
        }
    }
    printf("Room %lld not found. \n", number);
    return FALSE;
}

class Client {
public:
    string name;
    Client(string n);
    bool request(Admin& admin);
    void showInfo();
protected:
    mInt vdesiredPrice = 0;
```

```
    mInt vMoney = 0;
    mInt vStart = 0;
    mInt vDuration = 0;
    mInt vBookedRoom = 0;
};
Client::Client(string n) {
    name = n;
    vMoney = rand() % 1100;
    vdesiredPrice = rand() % vMoney;
    vStart = rand() % 90 + 1;
    vDuration = rand() % 20 + 1;
}
bool Client::request(Admin& admin) {
    Request req = { name, vdesiredPrice, vStart, vDuration };
    printf("\n[Client %s -> Admin %s] Requesting room for %lld price
from %lld during %lld period... \n", name.c_str(), admin.name.c_str(),
req.desiredPrice, req.vacStart, req.vacDuration);
    Response res = admin.check(req);
    if (res.status == 1) {
        vMoney -= res.rPrice;
        vBookedRoom = res.rNumber;
        printf("[Client %s -> Admin %s] Yes, awesome\n", name.c_str(),
res.adminName.c_str());
        return admin.book(name, res.rNumber, vStart, vDuration);
    }
    if (res.status == 0 && res.rPrice <= vMoney) {
        vMoney -= res.rPrice;
        vBookedRoom = res.rNumber;
        printf("[Client %s -> Admin %s] Okay, fortunatly, I have enough
money\n", name.c_str(), res.adminName.c_str());
        return admin.book(name, res.rNumber, vStart, vDuration);
    }
    printf("[Client %s -> Admin %s] No, it is too expensive for me. I
leaving your hotel!\n", name.c_str(), res.adminName.c_str());
    return FALSE;
}
void Client::showInfo() {
    printf("Client %s: %lld for room, %lld at all, start %lld, duration
%lld, room %lld \n", name.c_str(), vdesiredPrice, vMoney, vStart,
vDuration, vBookedRoom);
}

DWORD Child2ProcessId = GetCurrentProcessId();
TCHAR sTargetFilePath[MAX_PATH];

HANDLE ghWriteEvent;
```

```
DWORD WINAPI AdminThreadFunction(LPVOID lpParam);
DWORD WINAPI ClientThreadFunction(LPVOID lpParam);

struct AdminThreadParams {
    Admin admin = Admin("");
    HANDLE ghAdminMutex;
};

struct ClientThreadParams {
    Client client = Client("");
    mInt threadSleepTime = 0;
    mInt adminId = 0;
};

AdminThreadParams   threadAdmins[MAX_ADMINS];

HANDLE hPipe;
TCHAR  chBuf[BUFSIZE];
BOOL   fSuccess = FALSE;
DWORD  cbRead, cbToWrite, cbWritten, dwMode;
LPCTSTR lpszPipename = TEXT("\\\\.\\pipe\\mynamedpipe");

DWORD WINAPI ClientThreadFunction(LPVOID lpParam)
{
    ClientThreadParams* params = (ClientThreadParams*)lpParam;
    Client& client = params->client;
    mInt& threadSleepTime = params->threadSleepTime;
    mInt& adminId = params->adminId;
    Admin& admin = threadAdmins[adminId].admin;
    HANDLE& ghAdminMutex = threadAdmins[adminId].ghAdminMutex;

    BOOL rSuccess;
    DWORD ClientThreadId = GetCurrentThreadId();
    printf("(%d - %d) Client thread created and waiting for %lld
ms...\n", Child2ProcessId, ClientThreadId, threadSleepTime);
    Sleep(threadSleepTime);

    DWORD dwWaitResult = WaitForSingleObject(
        ghAdminMutex,
        INFINITE);
    rSuccess = client.request(admin);
    ReleaseMutex(ghAdminMutex);

    if (rSuccess)
    {
        _tprintf(TEXT("(%d - %d) Room successfully booked\n"),
```

```
Child2ProcessId, ClientThreadId);
    }
    else
        _tprintf(TEXT("(%d - %d) Room cannot be booked\n"),
Child2ProcessId, ClientThreadId);

    return 0;
}

DWORD WINAPI AdminThreadFunction(LPVOID lpParam)
{
    AdminThreadParams* params = (AdminThreadParams*)lpParam;
    Admin& admin = params->admin;

    Request req;
    Response res;
    DWORD AdminThreadId = GetCurrentThreadId();
    printf("(%d - %d) Admin thread created and waiting for
requests...\n", Child2ProcessId, AdminThreadId);

    while (TRUE) {
        // Start block
        res = admin.check(req);
        if (res.status) {
            admin.book();
            _tprintf(TEXT("(%d - %d) Room successfully booked\n"),
Child2ProcessId, AdminThreadId);
        }
        else
            _tprintf(TEXT("(%d - %d) Room cannot be booked\n"),
Child2ProcessId, AdminThreadId);
        // End block
    };
    return 0;
}

int _tmain(int argc, TCHAR* argv[])
{
    if (argc != 2)
    {
        _tprintf(TEXT("Usage: %s <target file>\n"), argv[0]);
        return 1;
    }

    StringCchCopy(sTargetFilePath, MAX_PATH, argv[1]);
```

```
    printf("(%d) Child2 process running.... \n", Child2ProcessId);

    // Open an Event.
    ghWriteEvent = OpenEvent(
        EVENT_ALL_ACCESS,
        TRUE,
        TEXT("WriteEvent")
    );

    if (ghWriteEvent == NULL)
    {
        printf("OpenEvent failed (%d)\n", GetLastError());
        return 1;
    }

    printf("(%d) Waiting for event from Child1 process... \n",
Child2ProcessId);
    WaitForSingleObject(ghWriteEvent, 10000);

    // Connect to Named Pipe.
    while (1)
    {
        hPipe = CreateFile(
            lpszPipename,
            GENERIC_READ |
            GENERIC_WRITE,
            0,
            NULL,
            OPEN_EXISTING,
            0,
            NULL);

        if (hPipe != INVALID_HANDLE_VALUE)
            break;

        if (GetLastError() != ERROR_PIPE_BUSY)
        {
            _tprintf(TEXT("(%d) Could not open pipe. GLE=%d\n"),
Child2ProcessId, GetLastError());
            return -1;
        }

        if (!WaitNamedPipe(lpszPipename, 20000))
        {
            printf("(%d) Could not open pipe: 20 second wait timed
out.", Child2ProcessId);
```

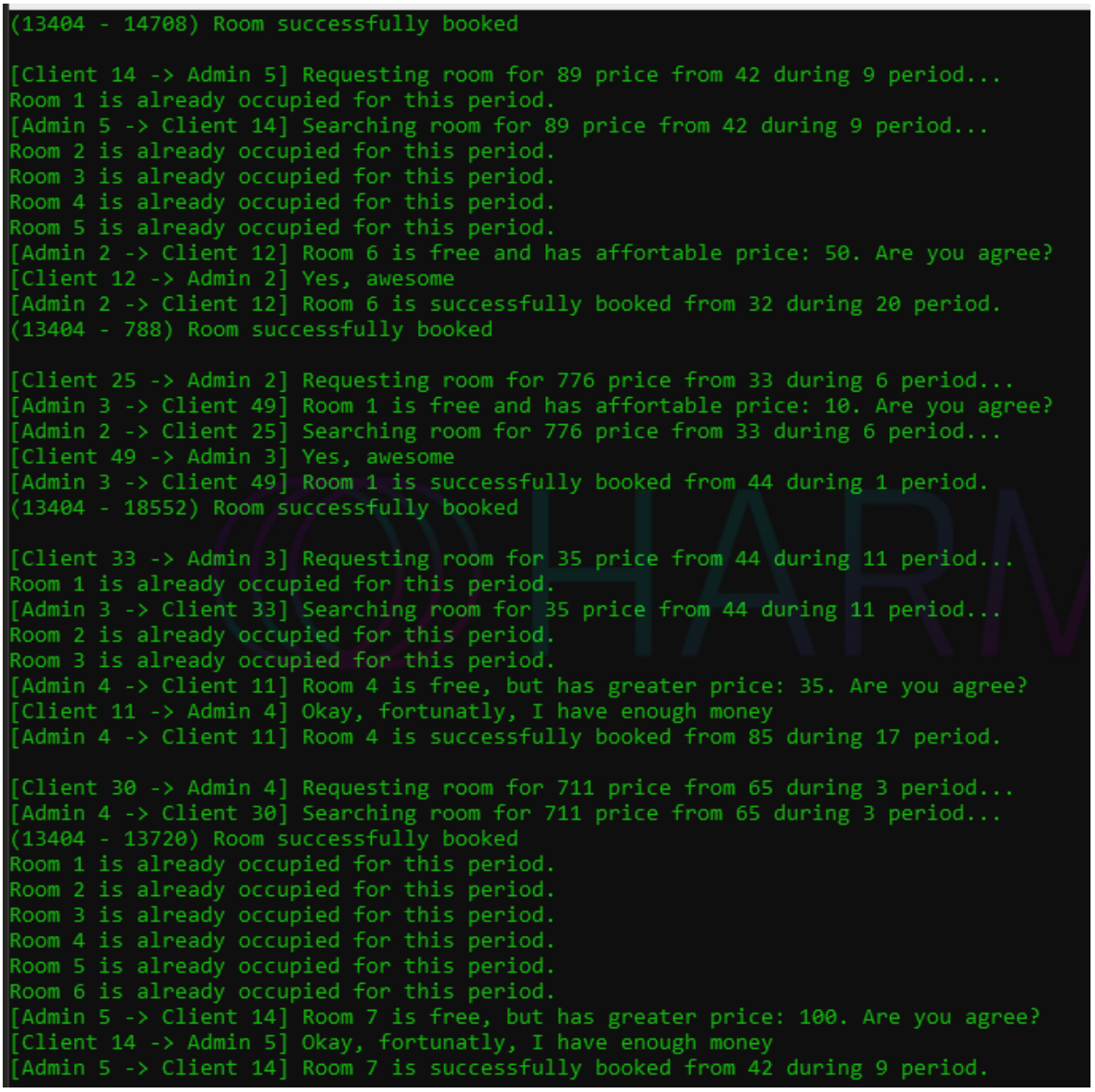
```
            return -1;
        }
    }

    dwMode = PIPE_READMODE_MESSAGE;
    fSuccess = SetNamedPipeHandleState(
        hPipe,
        &dwMode,
        NULL,
        NULL);

    if (!fSuccess)
    {
        _tprintf(TEXT("SetNamedPipeHandleState failed. GLE=%d\n"),
GetLastError());
        return -1;
    }

    // Create Admin Threads.
    DWORD               dwAdmThreadIdArray[MAX_ADMINS];
    HANDLE              hAdmThreadArray[MAX_ADMINS];

    for (mInt i = 0; i < MAX_ADMINS; i++) {
        threadAdmins[i] = { Admin(to_string(i + 1)), CreateMutex(NULL,
FALSE, NULL) };
        hAdmThreadArray[i] = CreateThread(
            NULL,
            0,
            AdminThreadFunction,
            &threadAdmins[i],
            0,
            &dwAdmThreadIdArray[i]);
        if (hAdmThreadArray[i] == NULL)
        {
            _tprintf(TEXT("Admin CreateThread Error"));
            ExitProcess(3);
        }
    }

    // Create Client Threads.
    DWORD               dwThreadIdArray[MAX_THREADS];
    HANDLE              hThreadArray[MAX_THREADS];
    ClientThreadParams  threadClients[MAX_THREADS];

    for (mInt i = 0; i < MAX_THREADS; i++) {
        threadClients[i] = { Client(to_string(i + 1)), rand() % 10000 +
```

```
1000, rand() % MAX_ADMINS };
        hThreadArray[i] = CreateThread(
            NULL,
            0,
            ClientThreadFunction,
            &threadClients[i],
            0,
            &dwThreadIdArray[i]);
        if (hThreadArray[i] == NULL)
        {
            _tprintf(TEXT("CreateThread Error"));
            ExitProcess(3);
        }
    }

    WaitForMultipleObjects(MAX_THREADS, hThreadArray, TRUE, INFINITE);
    WaitForMultipleObjects(MAX_ADMINS, hAdmThreadArray, TRUE, INFINITE);

    // Close all thread handles and free memory allocations.
    CloseHandle(ghWriteEvent);
    for (mInt i = 0; i < MAX_ADMINS; i++)
        CloseHandle(threadAdmins[i].ghAdminMutex);
    for (mInt i = 0; i < MAX_THREADS; i++)
        CloseHandle(hThreadArray[i]);
    for (mInt i = 0; i < MAX_ADMINS; i++)
        CloseHandle(hAdmThreadArray[i]);

    // Close Pipe.
    CloseHandle(hPipe);

    return 0;
}
```

Скріншоти:

```
D:\Microsoft Visual Studio\Workspace\SysProga\Lab5>Debug\Lab5.exe
(10288) Parent process running....

(3032) Child1 process running....
(3032) Sleep for 5 secs...

(10288) Pipe Server: Main thread awaiting client connection on \\.\pipe\mynamedpipe

(13404) Child2 process running....
(13404) Waiting for event from Child1 process...


(10288) Client connected, creating a processing thread.


(10288) Client connected, creating a processing thread.
(13404 - 11780) Admin thread created and waiting for requests...
(13404 - 15776) Admin thread created and waiting for requests...
(13404 - 6588) Admin thread created and waiting for requests...

(10288) Client Request String:
    "(3032) Created file D:\Microsoft Visual Studio\Workspace\SysProga\Lab5_1\two.txt"

(10288) InstanceThread: client disconnected.
(13404 - 8312) Admin thread created and waiting for requests...
(13404 - 10256) Admin thread created and waiting for requests...
(13404 - 11624) Client thread created and waiting for 1600 ms...
(13404 - 17144) Client thread created and waiting for 7224 ms...
(13404 - 6744) Client thread created and waiting for 4195 ms...
(13404 - 18768) Client thread created and waiting for 10314 ms...
(13404 - 6416) Client thread created and waiting for 1580 ms...
(13404 - 17004) Client thread created and waiting for 9009 ms...
(13404 - 7712) Client thread created and waiting for 7038 ms...
(13404 - 1252) Client thread created and waiting for 10815 ms...
(13404 - 19056) Client thread created and waiting for 5272 ms...
(13404 - 1756) Client thread created and waiting for 9875 ms...
(13404 - 13720) Client thread created and waiting for 2881 ms...
(13404 - 788) Client thread created and waiting for 4557 ms...
(13404 - 14540) Client thread created and waiting for 3600 ms...
(13404 - 19072) Client thread created and waiting for 4401 ms...
(13404 - 10904) Client thread created and waiting for 5182 ms...
(13404 - 17160) Client thread created and waiting for 10832 ms...
```

```
[Admin 1 -> Client 27] Searching room for 62 price from 20 during 2 period...
[Admin 1 -> Client 27] Room 1 is free and has affortable price: 10. Are you agree?
[Client 27 -> Admin 1] Yes, awesome
[Admin 1 -> Client 27] Room 1 is successfully booked from 20 during 2 period.
(13404 - 11796) Room successfully booked
[Admin 4 -> Client 24] Room 1 is free and has affortable price: 10. Are you agree?
[Client 24 -> Admin 4] Yes, awesome
[Admin 4 -> Client 24] Room 1 is successfully booked from 84 during 16 period.
(13404 - 15892) Room successfully booked

[Client 19 -> Admin 4] Requesting room for 261 price from 86 during 1 period...
[Admin 5 -> Client 1] Room 1 is free, but has greater price: 10. Are you agree?
[Admin 4 -> Client 19] Searching room for 261 price from 86 during 1 period...
[Client 1 -> Admin 5] No, it is too expensive for me. I leaving your hotel!
(13404 - 11624) Room cannot be booked

[Client 44 -> Admin 5] Requesting room for 75 price from 2 during 5 period...
[Admin 5 -> Client 44] Searching room for 75 price from 2 during 5 period...

[Client 41 -> Admin 1] Requesting room for 485 price from 87 during 8 period...
[Admin 1 -> Client 41] Searching room for 485 price from 87 during 8 period...
Room 1 is already occupied for this period.
[Admin 1 -> Client 41] Room 2 is free and has affortable price: 10. Are you agree?
[Client 41 -> Admin 1] Yes, awesome
[Admin 1 -> Client 41] Room 2 is successfully booked from 87 during 8 period.
(13404 - 11744) Room successfully booked

[Client 42 -> Admin 1] Requesting room for 114 price from 86 during 5 period...
[Admin 1 -> Client 42] Searching room for 114 price from 86 during 5 period...
Room 1 is already occupied for this period.
Room 2 is already occupied for this period.
[Admin 1 -> Client 42] Room 3 is free and has affortable price: 10. Are you agree?
[Client 42 -> Admin 1] Yes, awesome
[Admin 1 -> Client 42] Room 3 is successfully booked from 86 during 5 period.
(13404 - 9220) Room successfully booked

[Client 39 -> Admin 1] Requesting room for 240 price from 15 during 1 period...
[Admin 1 -> Client 39] Searching room for 240 price from 15 during 1 period...
[Admin 1 -> Client 39] Room 1 is free and has affortable price: 10. Are you agree?
[Client 39 -> Admin 1] Yes, awesome
[Admin 1 -> Client 39] Room 1 is successfully booked from 15 during 1 period.
(13404 - 1632) Room successfully booked
Room 1 is already occupied for this period.
```




Повний код можна знайти у GitHub-репозиторії:
https://github.com/NazarPonochevnyi/Programming-Labs/blob/master/System%20Programming/Lab5/lab5.cpp