Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра Систем Штучного Інтелекту

Звіт

до лабораторної роботи № 5

з дисципліни

Операційні системи

на тему:

“Робота з динамічними бібліотеками в ОС Windows”

Виконав: студент КН-217

Ратушняк Денис

Прийняв: доцент каф. СШІ

Кривенчук Ю. П.

Львів – 2022

Мета роботи: Ознайомитися з динамічно-зв’язувальними бібліотеками (Dynamic-link library) в ОС Windows. Навчитися реалізовувати динамічно-зв’язувальні бібліотеки.

Завдання:

- [Складність 1] Реалізувати лабораторну роботу №4 у вигляді динамічно-зв’язувальної бібліотеки. Запустити створену бібліотеку з командної стрічки (cmd.exe) за допомогою rundll32.exe
- [Складність 2] Створити окрему програму і реалізувати статичний зв'язок між програмою та бібліотекою.
- [Складність 3] Створити окрему програму і реалізувати динамічний зв'язок між програмою та бібліотекою.

Код програми

```
#include <bits/stdc++.h>
#include <windows.h>
#include <tchar.h>
//#include "lab5dllheaderv1.h"
using namespace std;
int main(){
    HMODULE h =
LoadLibrary("A:\\T\\3_term\\Operating_Systems\\5labDLLv1\\bin\\Debug\\5labDLLv1.dll");
    if(!h){
        cout << "Could not open the library";
        return 1;
    }
    FARPROC fn = GetProcAddress(h, "runTask");
    if(!fn){
        cout << "Could not locate the function";
        return 2;
    }
    fn();
    //runTask();
    return 0;
}
```

Код хедера

```
#include <tchar.h>
#include <windows.h>
using namespace std;

#ifdef LAB5DLLHEADERV1_EXPORTS
#define LAB5DLLHEADERV1__API __declspec(dllexport)
#else
#define LAB5DLLHEADERV1_API __declspec(dllimport)
#endif

extern "C" LAB5DLLHEADERV1_API void runTask();

extern "C" LAB5DLLHEADERV1_API void calc_points();

extern "C" LAB5DLLHEADERV1_API void read(vector< vector<long long> > &a, ifstream &input);

extern "C" LAB5DLLHEADERV1_API void print(vector< vector<long long> > &a, ofstream &output);

extern "C" LAB5DLLHEADERV1_API DWORD WINAPI MyThreadFunction(LPVOID lpParam);

extern "C" LAB5DLLHEADERV1_API void solve(int sx, int sy, int step, int type, int num);
```

Код .cpp файлу з якого була створена бібліотека

```
#include <bits/stdc++.h>

#include "lab5dllheaderv1.h"

typedef long long ll;
typedef long double ld;
typedef vector< vector<ll> > matrix;

///HERE YOU CAN CHANGE INPUT
const int TOTAL_THREADS = 100;
const int MAX_SEM_COUNT = 10;
string version = "10000_10000_10000";
ll type_of_task = 1;
ll low_priority_thread = 3;
ll high_priority_thread = 7;

///HERE YOU CAN CHANGE INPUT
ll done_threads = 0;
ll needThreads;
HANDLE hMutex;
HANDLE ghSemaphore;
HANDLE hCriticalSection;
const long long mod = 1e9+7;

typedef struct MyData
{
    int startx, starty, step, type;
    int threadNum;
} MYDATA, *PMYDATA;

//PMYDATA pDataArray[TOTAL_THREADS];
MYDATA pDataArray[TOTAL_THREADS];
DWORD   dwThreadIdArray[TOTAL_THREADS];
HANDLE  hThreadArray[TOTAL_THREADS];
HANDLE stackTh[64];

matrix A;
ll n,m,k,max_number;
ll add;
vector < pair<ll,ll> > diag[TOTAL_THREADS];

vector< pair< pair<ll,ll>, pair<ll,ll> > > points_for_diag;

void solve(int sx, int sy, int step, int type, int num)
{
    if(type == 1)
    {
        int i = sx;
        int j = sy;
        while(step--)
        {
            diag[num][i+j].first += A[i][j];
            diag[num][i+j].second ++;
            diag[num][i-j+add].first += A[i][j];
            diag[num][i-j+add].second ++;
            //cout << i << " " << j << endl;
            //cout << num << " " << step << endl;
            j++;
            if(j == m)
            {
```

```
                j = 0;
                i ++;
                if(i == n) break;
            }
        }
    }
    else
    {
        int now = sx * n + sy;

        while(now < n * m)
        {
            int i = now / m;
            int j = now % m;

            diag[num][i+j].first += A[i][j];
            diag[num][i+j].second ++;

            diag[num][i-j+add].first += A[i][j];
            diag[num][i-j+add].second ++;
            //cout << num << " " << now << " " << step << endl;
            now += step;
        }
    }
    ll ans = 0;
    for(int i = 1 ; i < diag[num].size(); ++i)
    {
        if(diag[num][i].first * diag[num][ans].second > diag[num][i].second * diag[num][ans].first) ans = i;
    }
    // for(int i = 0; i < diag[0].size(); ++i) cout << diag[0][i].first << " " << diag[0][i].second << endl;

    ld result = (ld)diag[0][ans].first / (ld)diag[0][ans].second;
    WaitForSingleObject(hMutex, INFINITE);
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    COORD pos = {0, 1};
    SetConsoleCursorPosition(hConsole, pos);
    cout << "Thread " << num << " MaxAverage = " << result;
    ReleaseMutex(hMutex);

}

ld threadTime[TOTAL_THREADS];

DWORD WINAPI MyThreadFunction(LPVOID lpParam)
{
    if(MAX_SEM_COUNT)
    {
        DWORD dwWaitResult;
        BOOL bContinue=TRUE;

        while(bContinue)
        {
            // Try to enter the semaphore gate.
            dwWaitResult = WaitForSingleObject(
                                ghSemaphore,     // handle to semaphore
                                0);              // zero-second time-out interval

            if(dwWaitResult == WAIT_OBJECT_0){

                // TODO: Perform task
                LARGE_INTEGER st, en, fq;
                QueryPerformanceFrequency(&fq);
```

```
            QueryPerformanceCounter(&st);

            PMYDATA pDataVar;

            pDataVar = (PMYDATA)lpParam;
            MYDATA DataVar = *pDataVar;
            //cout << "HERE IS " << " " << pDataVar->threadNum << " thread" << endl;
            solve(DataVar.startx, DataVar.starty, DataVar.step, DataVar.type, DataVar.threadNum);
            QueryPerformanceCounter(&en);

            ld currtime = (en.QuadPart-st.QuadPart)*1000.0/fq.QuadPart;
            threadTime[pDataVar->threadNum] = currtime;
            WaitForSingleObject(hMutex, INFINITE);
            done_threads ++;
            HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
            COORD pos = {0, 0};
            SetConsoleCursorPosition(hConsole, pos);
            cout << fixed << setprecision(3) << 100.0*(double)done_threads/(double)needThreads << '%';
            ReleaseMutex(hMutex);


            //printf("Thread %d: wait succeeded\n", GetCurrentThreadId());
            bContinue=FALSE;

            // Release the semaphore when task is finished

            if (!ReleaseSemaphore(
                        ghSemaphore,   // handle to semaphore
                        1,             // increase count by one
                        NULL) )        // not interested in previous count
            {
                printf("ReleaseSemaphore error: %d\n", 3);
            }
        }

        // The semaphore was nonsignaled, so a time-out occurred.
        else{
            //printf("Thread %d: wait timed out\n", GetCurrentThreadId());
            Sleep(3);
        }
    }
    return 0;
}

LARGE_INTEGER st, en, fq;
QueryPerformanceFrequency(&fq);
QueryPerformanceCounter(&st);

PMYDATA pDataVar;

pDataVar = (PMYDATA)lpParam;
MYDATA DataVar = *pDataVar;
//cout << "HERE IS " << " " << pDataVar->threadNum << " thread" << endl;
solve(DataVar.startx, DataVar.starty, DataVar.step, DataVar.type, DataVar.threadNum);
QueryPerformanceCounter(&en);

ld currtime = (en.QuadPart-st.QuadPart)*1000.0/fq.QuadPart;
threadTime[pDataVar->threadNum] = currtime;
WaitForSingleObject(hMutex, INFINITE);
done_threads ++;
HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
COORD pos = {0, 0};
```

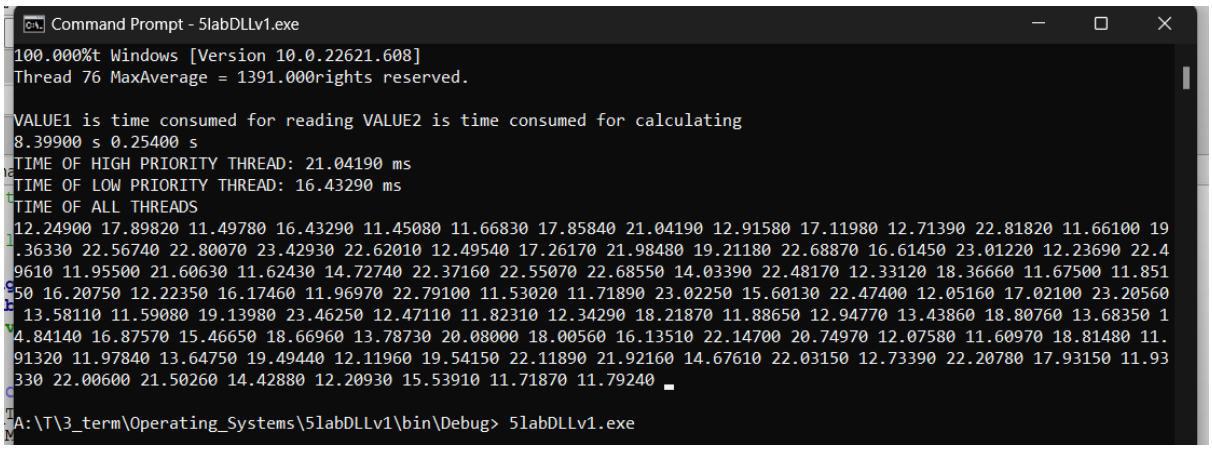
```
    SetConsoleCursorPosition(hConsole, pos);
    cout << fixed << setprecision(3) << 100.0*(double)done_threads/(double)needThreads << '%';
    ReleaseMutex(hMutex);
    return 0;
}

void print(matrix &a)
{
    for(int i = 0; i < a.size(); ++ i)
    {
        for(int j = 0; j < a[i].size(); ++ j)
        {
            cout << a[i][j] << " ";
        }
        cout << "\n";
    }
    cout << "\n";
}

void print(matrix &a, ofstream &output)
{
    for(int i = 0; i < a.size(); ++ i)
    {
        for(int j = 0; j < a[i].size(); ++ j)
        {
            output << a[i][j] << " ";
        }
        output << "\n";
    }
    output << "\n";
}

void read(matrix &a, ifstream &input)
{
    for(int i = 0; i < a.size(); ++ i)
    {
        for(int j = 0; j < a[i].size(); ++ j)
        {
            input >> a[i][j];
        }
    }
}


void calc_points()
{
    points_for_diag.resize(2 * m + 2 * n - 2);
    ll x1,y1,x2,y2;
    x1 = y1 = x2 = y2 = 0;

    for(int i = 0; i < (2 * m + 2 * n - 2)/2; ++i)
    {
        points_for_diag[i] = {{x1, y1}, {x2, y2}};
        if(x1 == n - 1) y1++;
        else x1++;

        if(y2 == m - 1) x2++;
        else y2++;
    }

    x1 = x2 = 0;
    y1 = y2 = m-1;
```

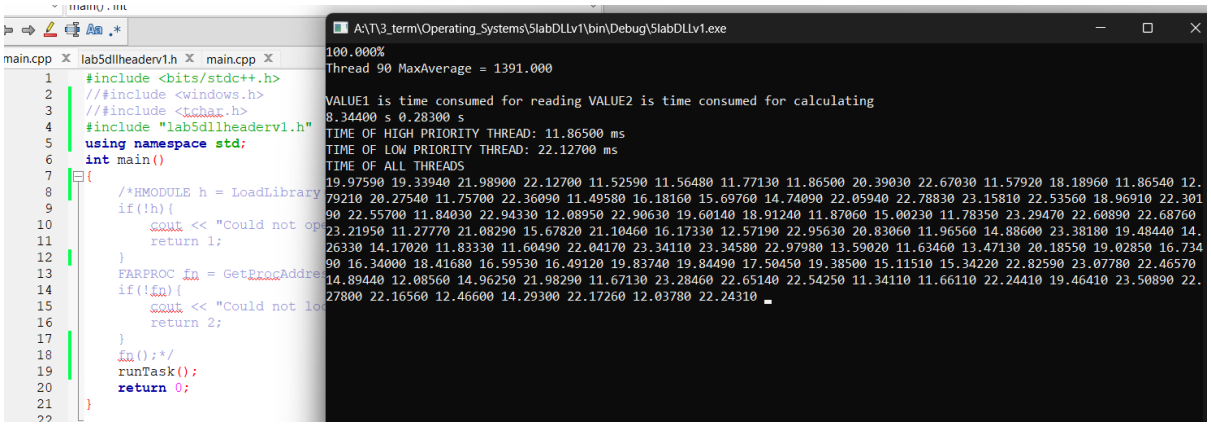
```
    for(int i = (2 * m + 2 * n - 2)/2; i < (2 * m + 2 * n - 2); ++i)
    {
        points_for_diag[i] = {{x1, y1}, {x2, y2}};
        if(y1 == 0) x1++;
        else y1--;

        if(x2 == n - 1) y2--;
        else x2++;
    }
}

void runTask()
{
    string test_path = "A:\\T\\3_term\\Operating_Systems\\5lab\\tests\\" + version + "_in.txt";

    low_priority_thread %= TOTAL_THREADS;
    high_priority_thread %= TOTAL_THREADS;
    if(low_priority_thread == high_priority_thread) low_priority_thread--;
    if(low_priority_thread == -1) low_priority_thread = 1;
    if(low_priority_thread == TOTAL_THREADS) low_priority_thread = 0;

    if(MAX_SEM_COUNT )ghSemaphore = CreateSemaphore(
                        NULL,           // default security attributes
                        MAX_SEM_COUNT,  // initial count
                        MAX_SEM_COUNT,  // maximum count
                        NULL);          // unnamed semaphore

    if(MAX_SEM_COUNT && ghSemaphore == NULL)
    {
        printf("CreateSemaphore error: %d\n", 1);
        return;
    }

    ifstream input(test_path);
    ld t0 = clock();
    input >> n >> m >> max_number;
    add = n + 2*m - 2;
    calc_points();
    A.resize(n);
    for(int i = 0; i < n; ++i) A[i].resize(m,0);

    read(A, input);
    //print(A);

    ll step1 = (n * m / TOTAL_THREADS);
    if(!step1) step1++;


    LARGE_INTEGER st, en, fq;
    QueryPerformanceFrequency(&fq);
    QueryPerformanceCounter(&st);

    ld t1 = clock();
    ll now1 = 0;
    ll now2 = 0;

    needThreads = min(ll(n * m), ll(TOTAL_THREADS));
    ll step2 = needThreads;

    hMutex = CreateMutex(NULL, FALSE, NULL);
    for(int i = 0; i < needThreads; ++i)
```

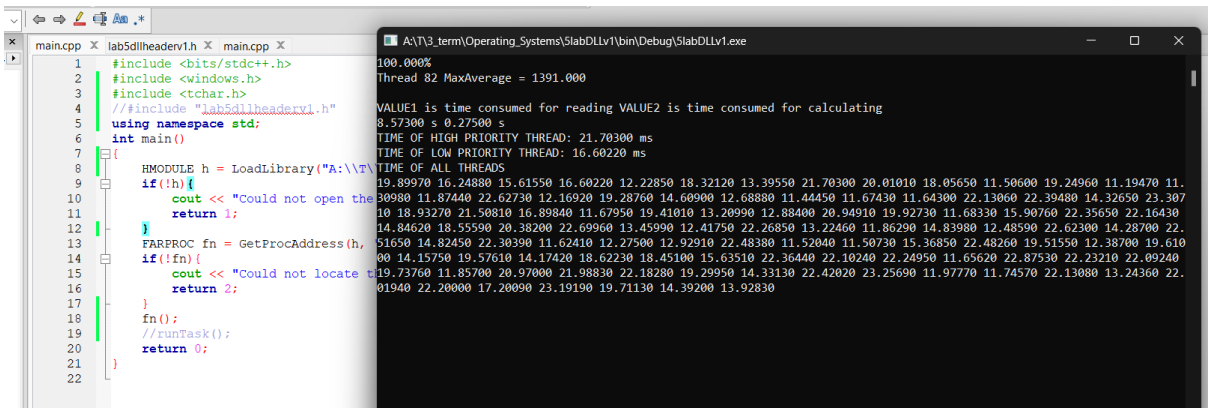
```
    {
        diag[i].resize(2 * n + 2 * m - 2, {0, 0});
        if(i%64 == 0) WaitForMultipleObjects(64, stackTh, TRUE, INFINITE);

        //pDataArray[i] = (PMYDATA) HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY,
sizeof(MYDATA));

        //if(pDataArray[i] == NULL) ExitProcess(2);

        pDataArray[i].threadNum = i;
        pDataArray[i].type = type_of_task;

        if(type_of_task == 1)
        {
            pDataArray[i].startx = now1 / m;
            pDataArray[i].starty = now1 % m;
            pDataArray[i].step = step1;
            now1 += step1;
        }
        else
        {
            pDataArray[i].startx = now2 / m;
            pDataArray[i].starty = now2 % m;
            pDataArray[i].step = step2;
            now2 ++;
        }

        hThreadArray[i] = CreateThread(
                                NULL,                   // default security attributes
                                0,                      // use default stack size
                                MyThreadFunction,       // thread function name
                                &pDataArray[i],         // argument to thread function
                                0,                      // use default creation flags
                                &dwThreadIdArray[i]);   // returns the thread identifier

        stackTh[i%64] = hThreadArray[i];

        if(hThreadArray[i] == NULL )
        {
            printf("CreateThread error: %d\n", 1);
            return;
        }

        if(i == high_priority_thread)
            SetThreadPriority(hThreadArray[i],THREAD_PRIORITY_HIGHEST);


        if(i == low_priority_thread)
            SetThreadPriority(hThreadArray[i],THREAD_PRIORITY_LOWEST);

    }

    WaitForMultipleObjects(min(64,(int)needThreads), stackTh, TRUE, INFINITE);

    for(int i = 0; i < needThreads; i++)
        if(hThreadArray[i] != INVALID_HANDLE_VALUE) CloseHandle(hThreadArray[i]);


    CloseHandle(hMutex);
    if(MAX_SEM_COUNT) CloseHandle(ghSemaphore);
```

```
    for(int i = 1; i < needThreads; ++i)
    {
        for(int j = 0; j < diag[0].size(); ++j)
        {
            diag[0][j].first += diag[i][j].first;
            diag[0][j].second += diag[i][j].second;
            //cout << i << " " << j << " " << diag[i][j].first << " " << diag[i][j].second << endl;
        }
    }

    ll ans = 0;
    for(int i = 1 ; i < diag[0].size(); ++i)
    {
        if(diag[0][i].first * diag[0][ans].second > diag[0][i].second * diag[0][ans].first) ans = i;
    }
    // for(int i = 0; i < diag[0].size(); ++i) cout << diag[0][i].first << " " << diag[0][i].second << endl;

    ld result = (ld)diag[0][ans].first / (ld)diag[0][ans].second;

    QueryPerformanceCounter(&en);
    ld t2 = clock();

    ld process_time = (en.QuadPart-st.QuadPart)*1000.0/fq.QuadPart;
    long long inttime = round(process_time * 100);
    long long dectime = inttime % 100;
    inttime /= 100;
    string result_path = "A:\\T\\3_term\\Operating_Systems\\5lab\\results_prob_right\\" +
to_string(TOTAL_THREADS);
    result_path += "threads_" + to_string(type_of_task) +"typeOfTask_" + to_string(MAX_SEM_COUNT)+
"maxSemCount_"    + to_string(inttime) + "." + to_string(dectime) + "ms_" + version + "_out.txt";
    ofstream output(result_path);

    //cout << process_time << " ms" << endl;

    output << "Number of diagonal where average element is the biggest = " << ans << " \n";
    output << fixed << setprecision(5) << "Average number = " << result << "\n";
    output << "2 points of this diagonal\n";
    output << points_for_diag[ans].first.first+1 << " " << points_for_diag[ans].first.second+1 << " " <<
points_for_diag[ans].second.first+1 << " " << points_for_diag[ans].second.second+1 << "\n";

    ld d1 = t1 - t0;
    ld d2 = t2 - t1;

    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    COORD pos = {0, 2};
    SetConsoleCursorPosition(hConsole, pos);

    cout << "\nVALUE1 is time consumed for reading VALUE2 is time consumed for calculating" << endl;
    cout << fixed << setprecision(5) << d1/CLOCKS_PER_SEC << " s " << d2/CLOCKS_PER_SEC << " s"
<< endl;
    cout << "TIME OF HIGH PRIORITY THREAD: " << threadTime[high_priority_thread] << " ms" << endl;
    cout << "TIME OF LOW PRIORITY THREAD: " << threadTime[low_priority_thread] << " ms" << endl;
    cout << "TIME OF ALL THREADS" << endl;
    for(int i = 0; i < TOTAL_THREADS; ++ i)
    {
        cout << threadTime[i] << " ";
    }
    int a;
    cin >> a;
    return;
}
```

```
int main(){
    runTask();
    return 0;
}
```







Висновок:

Я закріпив вміння та навички роботи з динамічно-зв’язувальними бібліотеками (Dynamic-link library) в ОС Windows. Навчитися реалізовувати динамічно-зв’язувальні бібліотеки. Створив свою динамічну бібліотеку, запустив головну функцію через консоль, та створив програму, у які підключив новостворенну бібліотеку статично, а потім динамічно.