

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Інститут комп'ютерних наук та інформаційних технологій
Кафедра програмного забезпечення



ЗВІТ

до лабораторної роботи № 7
на тему: «Статичні та динамічні бібліотеки. WINDOWS та LINUX»
з дисципліни: «Операційні системи»

Лектор:

ст. викладач кафедри ПЗ
Грицай О. Д.

Виконав:

ст. гр. ПЗ-22
Чаус О. М.

Прийняла:

ст. викладач кафедри ПЗ
Грицай О. Д.

« ____ » _____ 2022 р.

Σ= ____

Тема роботи: Статичні та динамічні бібліотеки. WINDOWS та LINUX.

Мета роботи: Ознайомитися з статичними та динамічними бібліотеками в операційних системах WINDOWS та LINUX. Навчитися реалізовувати статичні та динамічні бібліотеки.

Теоретичні відомості

Бібліотека – це файл, що містить один або декілька об'єктних файлів для вирішення близьких за тематикою завдань у розробці програмних продуктів. У бібліотеці можуть міститися об'єктні модулі, програмний код або дані, щоможуть використовуватись окремо або разом на різних етапах створення проекту(компілювання, лінування, завантаження чи виконання).

Бібліотека містить символний індекс, який складається з назв функцій і змінних, які містяться у бібліотеці. Це дозволяє прискорити процес лінування програми, оскільки пошук функцій і змінних в об'єктних файлах бібліотеки відбувається набагато швидше, ніж пошук в наборі вказаних об'єктних файлів. Тому використання бібліотеки дозволяє компактно зберігати усі необхідні об'єктні файли в одному місці, і при цьому значно підвищити швидкість компіляції. Бібліотеки прийнято розділяти відповідно до способу з'єднання на статичні та динамічні. Статичні бібліотеки зв'язуються з проектом перед завантаженням і є частиною бінарного файлу. Динамічні можуть бути зв'язані з проектом статично і динамічно. У різних операційних системах є свої особливості створення і використання бібліотек

Завдання для виконання лабораторної роботи

1. Реалізувати лабораторну роботу No5 (згідно варіанту) у вигляді статичної та динамічної бібліотеки в ОС WINDOWS.
2. Запустити створену динамічну бібліотеку з командної стрічки (cmd.exe) за допомогою rundll32.exe.
3. Створити окрему програму і реалізувати статичний зв'язок між програмою та бібліотекою із п. 1.4. Реалізувати можливість зробити потік від'єднаним.
4. Створити окрему програму і реалізувати динамічний зв'язок між програмою та бібліотекою із п. 1.6. Реалізувати синхронізацію потоків за допомогою вказаних методів (згідно варіанту)
5. Експортувати головну функцію бібліотеки під іншим іменем із п. 1. Результати виконання роботи оформити у звіт.
6. Реалізувати лабораторну роботу No6 у вигляді статичної та динамічної (поділюваної) бібліотеки в ОС LINUX.
7. Створити окрему програму і реалізувати статичний зв'язок між програмою та бібліотекою із п. 2
8. Створити окрему програму і реалізувати динамічний зв'язок між програмою та бібліотекою із п. 2.
9. Порівняти результати виконання програми та роботи бібліотек під ОС Windows та Linux.
10. Результати виконання роботи відобразити у звіті.

Хід роботи

1. WINDOWS

1.1. Статична бібліотека:

os_lab7_static_lib.h

```
#include <vector>
#include <Windows.h>
#include <string>

struct lines {
    int start;
    int end;
};

std::vector<std::string> split(const std::string str, const std::string
delim);
```

```
DWORD WINAPI semaphore(PVOID lpParam);
DWORD WINAPI critical_section(PVOID lpParam);
```

```
void create_threads();
```

os_lab7_static_lib.cpp

```
#include "pch.h"
#include "framework.h"
#include "os_lab7_staticlib.h"
#include <iostream>
#include <vector>
#include <Windows.h>
#include <fstream>
#include <chrono>

CRITICAL_SECTION cs;
std::ifstream file;
HANDLE my_semaphore;
std::vector<std::string> text;

std::vector<std::string> split(const std::string str, const std::string delim)
{
    std::vector<std::string> res;
    std::string buffer;
    for (int i = 0; i < str.size(); i++) {
        bool is_delim = false;
        for (int j = 0; j < delim.size(); j++) {
            if (str[i] == delim[j]) {
                if (buffer != "") {
                    res.push_back(buffer);
                    buffer = "";
                }
                is_delim = true;
                break;
            }
        }
        if (!is_delim)
            buffer += str[i];
    }
    return res;
}

DWORD WINAPI semaphore(PVOID lpParam) {
    lines* thread_lines = (lines*)(lpParam);
    std::string line;
    int start = thread_lines->start;
    int end = thread_lines->end;
    delete thread_lines;
    for (int i = start; i < end; i++) {
        line = text[i];
```

```

        std::vector<std::string> words = split(line, " ,.;:!?\"'\\n");
        std::string min_word = words[0];
        for (auto& word : words) {
            if (word.size() < min_word.size())
                min_word = word;
        }
        WaitForSingleObject(my_semaphore, INFINITE);
        std::cout << i << " : " << min_word << std::endl;
        ReleaseSemaphore(my_semaphore, 1, NULL);
        //Sleep(10);
    }
    return 0;
}

DWORD WINAPI critical_section(PVOID lpParam) {
    lines* thread_lines = (lines*)(lpParam);
    std::string line;
    int start = thread_lines->start;
    int end = thread_lines->end;
    delete thread_lines;
    for (int i = start; i < end; i++) {
        line = text[i];
        std::vector<std::string> words = split(line, " ,.;:!?\"'\\n");
        std::string min_word = words[0];
        for (auto& word : words) {
            if (word.size() < min_word.size())
                min_word = word;
        }
        EnterCriticalSection(&cs);
        std::cout << i << " : " << min_word << std::endl;
        LeaveCriticalSection(&cs);
        //Sleep(10);
    }
    return 0;
}

void create_threads() {
    InitializeCriticalSection(&cs);
    my_semaphore = CreateSemaphore(NULL, 1, 1, NULL);
    file.open("C:\\\\Users\\Oleh\\Downloads\\Telegram Desktop\\text1.txt");
    std::string line;
    while (std::getline(file, line)) {
        text.push_back(line);
    }
    file.close();
    int thread_num;
    int synchro;
    std::cout << "Please enter the number of threads (1-16): ";
    std::cin >> thread_num;
    std::unique_ptr<HANDLE[]> threads(new HANDLE[thread_num]);
    std::cout << "\\nPlease choose the type of synchronization.\\n\\t-\\tCritical
Section\\n2\\t-\\tSemaphore\\n";
    std::cin >> synchro;
    auto begin = std::chrono::high_resolution_clock::now();
    for (int i = 0; i < thread_num; i++) {
        std::cout << "\\nText size: " << text.size();
        int lines_per_thread = floor(text.size() / thread_num);
        std::cout << "\\nCalculated lines";
        lines* thread_lines = new lines;
        thread_lines->start = i * lines_per_thread;
        thread_lines->end = (i + 1) * lines_per_thread;
        if (i == thread_num - 1)
            thread_lines->end += text.size() % thread_num;
        std::pair<HANDLE, DWORD> this_thread;
        threads[i] = CreateThread(NULL, 0, (synchro == 1 ? critical_section :
semaphore), thread_lines, 0, NULL);
        std::cout << "\\nRun thread";
    }
}

```

```

        WaitForMultipleObjects(thread_num, threads.get(), TRUE, INFINITE);
        auto end = std::chrono::high_resolution_clock::now();
        std::cout << "\nExecution time: " <<
        (std::chrono::duration_cast<std::chrono::milliseconds>(end - begin)).count()
        << " ms.";
        for (int i = 0; i < thread_num; i++)
            CloseHandle(threads[i]);
        DeleteCriticalSection(&cs);
        CloseHandle(my_semaphore);
    }

```

static lib_client.cpp

```

#include "s_lib.h"

int main() {
    create_threads();
    return 0;
}

```

Вивід програми:

```

Microsoft Visual Studio Debug Console

Please enter the number of threads (1-16): 2

Please choose the type of synchronization.
1 - Critical Section
2 - Semaphore
2

Text size: 32568
Calculated lines
Run thread
Text size: 32568
Calculated lines
Run thread0 : a
16284 : at
1 : In
16285 : in
2 : a

```

1.2. Динамічна бібліотека

dyn_lib.h

```

#pragma once

#ifdef DYN_LIB_EXPORTS
#define DYN_LIB_API extern "C" __declspec(dllexport)
#else
#define DYN_LIB_API extern "C" __declspec(dllimport)
#endif

#include <iostream>
#include <vector>
#include <Windows.h>
#include <fstream>
#include <string>

struct lines {
    int start;
    int end;
};

CRITICAL_SECTION cs;
std::ifstream file;
HANDLE my_semaphore;
std::vector<std::string> text;

```

```
DWORD WINAPI semaphore(PVOID lpParam);
DWORD WINAPI critical_section(PVOID lpParam);
```

```
DYN_LIB_API void create_threads();
extern "C" void name_to_change();
```

dyn_lib.cpp

```
#include "pch.h"
#include "dyn_lib.h"
#include <iostream>
#include <vector>
#include <Windows.h>
#include <fstream>
#include <chrono>
#include <memory>

void print_hi() {
    std::cout << "Hi everyone, Im Yi Lon Ma!";
}

DWORD WINAPI semaphore(PVOID lpParam) {
    lines* thread_lines = (lines*)(lpParam);
    std::string line;
    int start = thread_lines->start;
    int end = thread_lines->end;
    delete thread_lines;
    for (int i = start; i < end; i++) {
        line = text[i];
        std::vector<std::string> words;
        std::string buffer;
        std::string delim = " ,.:!?\\";
        for (int i = 0; i < line.size(); i++) {
            bool is_delim = false;
            for (int j = 0; j < delim.size(); j++) {
                if (line[i] == delim[j]) {
                    if (buffer != "") {
                        words.push_back(buffer);
                        buffer = "";
                    }
                    is_delim = true;
                    break;
                }
            }
            if (!is_delim)
                buffer += line[i];
        }
        std::string min_word = words[0];
        for (auto& word : words) {
            if (word.size() < min_word.size())
                min_word = word;
        }
        WaitForSingleObject(my_semaphore, INFINITE);
        std::cout << i << " : " << min_word << std::endl;
        ReleaseSemaphore(my_semaphore, 1, NULL);
        //Sleep(10);
    }
    return 0;
}

DWORD WINAPI critical_section(PVOID lpParam) {
    lines* thread_lines = (lines*)(lpParam);
    std::string line;
    int start = thread_lines->start;
    int end = thread_lines->end;
    delete thread_lines;
    for (int i = start; i < end; i++) {
        line = text[i];
        std::vector<std::string> words;
```

```

        std::string buffer;
        std::string delim = " ,.:;!?\\";
        for (int i = 0; i < line.size(); i++) {
            bool is_delim = false;
            for (int j = 0; j < delim.size(); j++) {
                if (line[i] == delim[j]) {
                    if (buffer != "") {
                        words.push_back(buffer);
                        buffer = "";
                    }
                    is_delim = true;
                    break;
                }
            }
            if (!is_delim)
                buffer += line[i];
        }
        std::string min_word = words[0];
        for (auto& word : words) {
            if (word.size() < min_word.size())
                min_word = word;
        }
        EnterCriticalSection(&cs);
        std::cout << i << " : " << min_word << std::endl;
        LeaveCriticalSection(&cs);
        //Sleep(10);
    }
    return 0;
}

void create_threads() {
    InitializeCriticalSection(&cs);
    my_semaphore = CreateSemaphore(NULL, 1, 1, NULL);
    file.open("C:\\Users\\Oleh\\Downloads\\Telegram Desktop\\text1.txt");
    std::string line;
    while (std::getline(file, line)) {
        text.push_back(line);
    }
    file.close();
    int thread_num;
    int synchro;
    std::cout << "Please enter the number of threads (1-16): ";
    std::cin >> thread_num;
    auto threads = std::make_unique<HANDLE[]>(thread_num);
    std::cout << "\nPlease choose the type of synchronization.\n1\t-\tCritical
Section\n2\t-\tSemaphore\n";
    std::cin >> synchro;
    auto begin = std::chrono::high_resolution_clock::now();
    for (int i = 0; i < thread_num; i++) {
        std::cout << "\nText size: " << text.size();
        int lines_per_thread = floor(text.size() / thread_num);
        std::cout << "\nCalculated lines";
        lines* thread_lines = new lines;
        thread_lines->start = i * lines_per_thread;
        thread_lines->end = (i + 1) * lines_per_thread;
        if (i == thread_num - 1)
            thread_lines->end += text.size() % thread_num;
        std::pair<HANDLE, DWORD> this_thread;
        threads[i] = CreateThread(NULL, 0, (synchro == 1 ? critical_section :
semaphore), thread_lines, 0, NULL);
        std::cout << "\nRun thread";
    }
    WaitForMultipleObjects(thread_num, threads.get(), TRUE, INFINITE);
    auto end = std::chrono::high_resolution_clock::now();
    std::cout << "\nExecution time: " <<
(std::chrono::duration_cast<std::chrono::milliseconds>(end - begin)).count()
<< " ms.";
    for (int i = 0; i < thread_num; i++)

```

```

        CloseHandle(threads[i]);
        DeleteCriticalSection(&cs);
        CloseHandle(my_semaphore);
    }

extern "C" void name_to_change() {
    InitializeCriticalSection(&cs);
    my_semaphore = CreateSemaphore(NULL, 1, 1, NULL);
    file.open("C:\\Users\\Oleh\\Downloads\\Telegram Desktop\\text1.txt");
    std::string line;
    while (std::getline(file, line)) {
        text.push_back(line);
    }
    file.close();
    int thread_num;
    int synchro;
    std::cout << "Please enter the number of threads (1-16): ";
    std::cin >> thread_num;
    auto threads = std::make_unique<HANDLE[]>(thread_num);
    std::cout << "\nPlease choose the type of synchronization.\n1\t-\tCritical
Section\n2\t-\tSemaphore\n";
    std::cin >> synchro;
    auto begin = std::chrono::high_resolution_clock::now();
    for (int i = 0; i < thread_num; i++) {
        std::cout << "\nText size: " << text.size();
        int lines_per_thread = floor(text.size() / thread_num);
        std::cout << "\nCalculated lines";
        lines* thread_lines = new lines;
        thread_lines->start = i * lines_per_thread;
        thread_lines->end = (i + 1) * lines_per_thread;
        if (i == thread_num - 1)
            thread_lines->end += text.size() % thread_num;
        std::pair<HANDLE, DWORD> this_thread;
        threads[i] = CreateThread(NULL, 0, (synchro == 1 ? critical_section :
semaphore), thread_lines, 0, NULL);
        std::cout << "\nRun thread";
    }
    WaitForMultipleObjects(thread_num, threads.get(), TRUE, INFINITE);
    auto end = std::chrono::high_resolution_clock::now();
    std::cout << "\nExecution time: " <<
(std::chrono::duration_cast<std::chrono::milliseconds>(end - begin)).count()
<< " ms.";
    for (int i = 0; i < thread_num; i++)
        CloseHandle(threads[i]);
    DeleteCriticalSection(&cs);
    CloseHandle(my_semaphore);
}

```

dynamic_client/main.cpp

```

#include <windows.h>
#include <iostream>

typedef void(*func)();

int main() {
    HMODULE handle;
    handle =
LoadLibrary(L"C:\\Users\\Oleh\\source\\repos\\dyn_lib\\x64\\Debug\\dyn_lib.dll
");
    if (handle) {
        int answer;
        const char* func_name;
        std::cout << "Loaded library! :)\n";
        std::cout << "1 -\tuse declspec function\n2 -\tuse .def
function\n";
        std::cin >> answer;
        if (answer == 1)
            func_name = "create_threads";
    }
}

```



```

        else func_name = "changed_name";
        func call_func = (void(*)())GetProcAddress(handle,
"create_threads");
        call_func();
        FreeLibrary(handle);
    }
    else
        std::cout << "Failed to load library ;(\n";
}

static_client/main.cpp
#include <iostream>
#include "dyn_lib.h"

int main() {
    create_threads();
}

```

Вивід програми з динамічним зв'язуванням:

```

Microsoft Visual Studio Debug Console
Loaded library! :)
1 - use declspec function
2 - use .def function
2
Please enter the number of threads (1-16): 3

Please choose the type of synchronization.
1 - Critical Section
2 - Semaphore
1

Text size: 32568
Calculated lines
Run thread
Text size: 32568
Calculated lines

```

Вивід програми зі статичним зв'язуванням:

```

Microsoft Visual Studio Debug Console
Please enter the number of threads (1-16): 2

Please choose the type of synchronization.
1 - Critical Section
2 - Semaphore
2

Text size: 32568
Calculated lines
Run thread
Text size: 32568
Calculated lines
Run thread0 : a
16284 : at
1 : In
16285 : in
2 : a
16286 : ac
3 : et
16287 : a
4 : id

```

2. LINUX

2.1. Статична бібліотека

s_header.h

```
#include <vector>
#include <string>

struct lines {
    int start;
    int end;
};

extern "C" void print();
```

s_lib.cpp

```
#include <iostream>
#include <pthread.h>
#include <fstream>
#include "s_header.h"

std::vector<std::string> split(const std::string str, const std::string delim);

void print() {
    std::vector<std::string> text;
    std::ifstream file;
    file.open("/home/oleh/OS_lab66/text1.txt");
    std::string line;
    int i = 0;
    while(std::getline(file, line)) {
        text.push_back(line);
        i++;
    }
    file.close();
    for(int i = 0; i < text.size(); i++) {
        line = text[i];
        std::vector<std::string> words = split(line, " ,.:!&\"'\n");
        std::string min_word = words[0];
        for (auto & word : words)
            if(word.size() < min_word.size())
                min_word = word;
        std::cout << i << " : " << min_word << std::endl;
    }
}

std::vector<std::string> split(const std::string str, const std::string delim)
{
    std::vector<std::string> res;
    std::string buffer;
    for(int i = 0; i < str.size(); i++) {
        bool is_delim = false;
        for(int j = 0; j < delim.size(); j++) {
            if(str[i] == delim[j]) {
                if(buffer != "") {
                    res.push_back(buffer);
                    buffer = "";
                }
                is_delim = true;
            }
        }
        if(!is_delim)
            buffer += str[i];
    }
    return res;
}
```

main.cpp

```
#include "s_header.h"

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

Вивід програми:

```
oleh@oleh-VirtualBox:~/os_lab7/static_lib$ ./static_client
0:a
1:In
2:a
3:et
4:id
5:in
6:a
7:in
```

Команди, використані для компіляції:

```
oleh@oleh-VirtualBox:~/os_lab7$ g++ -c s_lib.cpp -o s_lib.o
oleh@oleh-VirtualBox:~/os_lab7$ ar -rc libstatic.a s_lib.o
oleh@oleh-VirtualBox:~/os_lab7$ ranlib libstatic.a
oleh@oleh-VirtualBox:~/os_lab7$ g++ main.cpp -L. -lstatic -o static_client
oleh@oleh-VirtualBox:~/os_lab7$ ./static_client
```

2.2. Динамічна бібліотека

d_header.h

```
extern "C" void print();
```

d_lib.cpp

```
#include <iostream>
#include <pthread.h>
#include <fstream>
#include <vector>
#include "d_header.h"
```

```
std::vector<std::string> split(const std::string str, const std::string delim);
```

```
void print() {
    std::vector<std::string> text;
    std::ifstream file;
    file.open("/home/oleh/OS_lab66/text1.txt");
    std::string line;
    int i = 0;
    while(std::getline(file, line)) {
        text.push_back(line);
        i++;
    }
    file.close();
    for(int i = 0; i < text.size(); i++) {
        line = text[i];
        std::vector<std::string> words = split(line, " ,.:!&\"'\n");
        std::string min_word = words[0];
        for (auto & word : words)
            if(word.size() < min_word.size())
                min_word = word;
        std::cout << i << ":" << min_word << std::endl;
    }
}
```

```
std::vector<std::string> split(const std::string str, const std::string delim)
{
    std::vector<std::string> res;
    std::string buffer;
    for(int i = 0; i < str.size(); i++) {
        bool is_delim = false;
```

```

        for(int j = 0; j < delim.size(); j++) {
            if(str[i] == delim[j]) {
                if(buffer != "") {
                    res.push_back(buffer);
                    buffer = "";
                }
                is_delim = true;
            }
        }
        if(!is_delim)
            buffer += str[i];
    }
    return res;
}

```

dynamic_linking.cpp

```
#include <dlfcn.h>
```

```

void *lib;
void (*func)();

```

```

int main() {
    lib = dlopen("/home/oleh/os_lab7/dynamic_lib/libdynamic.so", RTLD_LAZY);
    if (lib) {
        func = (void(*)())dlsym(lib, "print");
        func();
    }
    dlclose(lib);
}

```

Вивід програми:

```

oleh@oleh-VirtualBox:~/os_lab7/dynamic_lib$ ./dynamic_client
0:a
1:In
2:a
3:et
4:id
5:in
6:a

```

Команди, використані для компіляції:

```

g++ -fPIC -c d_lib.cpp
g++ -shared d_lib.o -o libdynamic.so
g++ dynamic_linking.cpp -o dynamic_client

```

static_linking.cpp

```
#include "d_header.h"
```

```
extern "C" void print();
```

```

int main() {
    print();
}

```

Вивід програми:

```
oleh@oleh-VirtualBox:~/os_lab7/dynamic_lib$ ./static_client
0:a
1:In
2:a
3:et
4:id
5:in
6:a
7:in
8:Et
9:a
10:A
```

Команди, використані для компіляції:

```
g++ -c static_linking.cpp
g++ static_linking.o -L. -ldynamic -o static_client
```

```
oleh@oleh-VirtualBox:~/os_lab7/dynamic_lib$ LD_LIBRARY_PATH=/home/oleh/os_lab7/dynamic_lib/libdynamic.so:${LD_LIBRARY_PATH}
oleh@oleh-VirtualBox:~/os_lab7/dynamic_lib$ export LD_LIBRARY_PATH
oleh@oleh-VirtualBox:~/os_lab7/dynamic_lib$ echo $LD_LIBRARY_PATH
/home/oleh/os_lab7/dynamic_lib/libdynamic.so:
oleh@oleh-VirtualBox:~/os_lab7/dynamic_lib$ ./static_client
```

Висновок: під час виконання лабораторної роботи ознайомився з статичними та динамічними бібліотеками в операційних системах WINDOWS та LINUX. Навчився реалізовувати статичні та динамічні бібліотеки.