Федеральное агентство связи

Сибирский государственный университет телекоммуникаций и

информатики

Кафедра прикладной математики и кибернетики

Отчёт по лабораторной работе №7

Выполнил: студент группы ИП–811 Адов А.С.

Проверил: Малков Е.А.

Новосибирск 2020

***Лабораторная 7а***

***Цель***: приобрести навыки разработки приложений с использованием dll и освоить интерфейс PSAPI.

***Задание 1***: написать программу prog1>>, с динамической загрузкой lib1>>.dll.

***Задание 2***: написать программу mon1>> с использованием функций PSAPI EnumProcesses, OpenProcess, EnumProcessModules, GetModuleBaseName и GetModuleFileName, определяющую все модули процесса и места расположения, соответствующих им файлов.

***Задание 3***: запустить prog1>>, периодически загружая и выгружая библиотеку lib1>>.dll; используя программу mon1>> отследить наличие модуля lib1>> в процессе prog1>>.

***Замечание***: используйте в качестве шаблона программу, разобранную на лекции 6.

***Лабораторная 7b***

***Цель***: закрепить сведения о структуре исполнямых файлов, получить навыки работы с интерфейсом IMAGEHLP.

***Задание 1***: получить список экспортируемых функций библиотеки kernel32.dll.

***Задание 2***: получить список импортируемых функций notepad.exe.

**Lab7b**:

***lab7b.c***

#include <windows.h>

#include <psapi.h>

#include <imagehlp.h>

#include <stdlib.h>

#include <stdio.h>

//#define TRIGGER\_SYSTEM\_COMMAND

//#define TRIGGER\_USE\_MONITORING

#define ERROR\_QUANTITY\_ARGUMENT 0b00000001

#define ERROR\_SECOND\_ARGUMENT 0b00000010

#define ERROR\_THIRD\_ARGUMENT 0b00000100

#define ERROR\_UNKNOWN\_MAP\_AND\_LOAD 0b00001000

#define ERROR\_ALLOCATION\_MEMORY 0b00010000

#define ERROR\_ENUM\_PROCESS\_MODULS 0b00100000

#define ERROR\_FOUND\_PROCESS 0b10000000

#ifdef TRIGGER\_SYSTEM\_COMMAND

#define ERROR\_OPEN\_FILE\_TEMP\_TXT 0b01000000

#define SIZE\_STRING\_TO\_SYSTEM\_COMMAND 31

#define CAPACITY\_DIGIT\_TO\_DECIMAL 10

#define TEST\_SIZE\_PROCESS\_NAME 20

#define FILE\_NAME\_FOR\_TEMP\_TXT "temp.txt"

#define READ\_ONLY "r"

#define FIRST\_CODE\_NUMBER\_TO\_INTEGER\_SYMBOL 0x30

#define LAST\_CODE\_NUMBER\_TO\_INTEGER\_SYMBOL 0x39

#define CONVERT\_CODE\_NUMBER\_FROM\_SYMBOL 0x0F

#else

#define ERROR\_ENUM\_PROCESSES 0b01000000

#endif

#define GET\_EXPORT\_FUNCTION 'E'

#define GET\_IMPORT\_FUNCTION 'I'

#define CORRECT\_QUANTITY\_ARGUMENT 3

#ifdef TRIGGER\_USE\_MONITORING

int Copy\_String\_In\_Char\_Array(char\* process\_name[], char copy\_string[], int\* size\_process\_name)

{

for(short int i = 0; copy\_string[i] != '\0'; i++) //counting quantity element in copt string

{

(\*size\_process\_name)++; //increment future size process name

}

\*process\_name = (char\*)calloc(\*size\_process\_name,sizeof(char)); //allocation need memory

if(\*process\_name == NULL)

{

return ERROR\_ALLOCATION\_MEMORY;

}

for(short int i = 0; i < \*size\_process\_name; i++)

{

(\*process\_name)[i] = copy\_string[i]; //copying in char array to char array

}

return 0;

}

int Get\_Pid\_Need\_Procces\_By\_Name(int\* pID, char process\_name[], int size\_process\_name)

{

#ifdef TRIGGER\_SYSTEM\_COMMAND

FILE\* file = NULL;

char\* system\_command = NULL;

char symbol = '0';

system\_command = (char\*)calloc((SIZE\_STRING\_TO\_SYSTEM\_COMMAND+size\_process\_name),sizeof(char));

if(system\_command == NULL)

{

return ERROR\_ALLOCATION\_MEMORY;

}

sprintf(system\_command,"tasklist | findstr %s > temp.txt", process\_name);

system(system\_command);

free(system\_command);

system\_command = NULL;

file = fopen(FILE\_NAME\_FOR\_TEMP\_TXT,READ\_ONLY);

if(file == NULL)

{

return ERROR\_OPEN\_FILE\_TEMP\_TXT;

}

do

{

symbol = getc(file);

} while((symbol != ' ') && (!feof(file)));

do

{

symbol = getc(file);

} while((symbol < FIRST\_CODE\_NUMBER\_TO\_INTEGER\_SYMBOL) || (symbol > LAST\_CODE\_NUMBER\_TO\_INTEGER\_SYMBOL) || feof(file));

do

{

(\*pID) = ((\*pID) + (symbol & CONVERT\_CODE\_NUMBER\_FROM\_SYMBOL));

(\*pID) = ((\*pID) \* CAPACITY\_DIGIT\_TO\_DECIMAL);

symbol = getc(file);

}while((symbol >= FIRST\_CODE\_NUMBER\_TO\_INTEGER\_SYMBOL) && (symbol <= LAST\_CODE\_NUMBER\_TO\_INTEGER\_SYMBOL) && !feof(file));

(\*pID) = ((\*pID) / CAPACITY\_DIGIT\_TO\_DECIMAL);

fclose(file);

#else

char temp\_process\_name[1000]; //temp array for process name

DWORD buffer\_process[1024]; //temp array for list process id

DWORD size\_buffer\_process = 0; // significant element in buffer process

HANDLE pHndl; //handle for process

int index\_equally = 0; //need index process

int size\_temp\_process = 0; //significant element in char array process name

if(EnumProcesses(buffer\_process, sizeof(buffer\_process), &size\_buffer\_process) == 0) //get process id for every process

{

return ERROR\_ENUM\_PROCESSES;

}

size\_buffer\_process = (size\_buffer\_process / sizeof(DWORD)); //calculate how many process identifiers were returned.

for(short int i = 0; i < (short int)size\_buffer\_process; i++) //print the name and process identifier for each process.

{

pHndl = OpenProcess(PROCESS\_QUERY\_INFORMATION | PROCESS\_VM\_READ, FALSE, buffer\_process[i]); //get handle for process by processID

GetProcessImageFileNameA(pHndl, temp\_process\_name, 1000); //get full path for process by handle on process

size\_temp\_process = 0; //reset significant element in char array process name

for(short int j = 0; ((j < 1000) && (temp\_process\_name[j] != '\0')); j++) // get significant element in char array process name

{

size\_temp\_process++;//increment significant element in char array process name

}

for(short int k = (size\_temp\_process - 1), j = (size\_process\_name - 1); ((k >-1) && (j >-1)); j--, k--)

{

if(process\_name[j] != temp\_process\_name[k]) //if last symbol in process not equally

{

index\_equally = -1; //reset index

break; //end cycle

}

else

{

index\_equally = i; //remember index for equally process names

}

}

if(index\_equally != -1)

{

\*pID = buffer\_process[i];//appoint process id for equally process names

break;

}

}

#endif

return 0;

}

#endif

int Check\_Argc(int argc)

{

if(argc == CORRECT\_QUANTITY\_ARGUMENT)

{

return 0;

}

printf("Error: incorect quantity argument in programm - need three\n");

return ERROR\_QUANTITY\_ARGUMENT;

}

int Get\_Export\_Function(PCSTR library\_name)

{

LOADED\_IMAGE LoadedImage;

PUCHAR BaseAddress;

DWORD RVAExpDir, VAExpAddress;

IMAGE\_EXPORT\_DIRECTORY\* ExpTable;

char\* sName;

DWORD nNames;

char\* pName;

char\*\* pNames;

if(!MapAndLoad(library\_name, NULL, &LoadedImage, TRUE, TRUE)) //load pe-file

{

printf("Need file dont find\n");

printf("Maybe incorrect second argument, please check this\n");

printf("Example correct second argument imagehlp.dll");

return ERROR\_UNKNOWN\_MAP\_AND\_LOAD;

}

if(LoadedImage.FileHeader->OptionalHeader.DataDirectory[IMAGE\_DIRECTORY\_ENTRY\_IMPORT].Size)//check size export directory

{//if not zero

BaseAddress = LoadedImage.MappedAddress; //get base adress loaded module

printf("0x%lx - Base Address\n", (unsigned long)BaseAddress);

RVAExpDir = LoadedImage.FileHeader->OptionalHeader.DataDirectory[IMAGE\_DIRECTORY\_ENTRY\_EXPORT].VirtualAddress; //get relativity virtual adress for export table

printf("0x%lx -RVA\n", RVAExpDir);

VAExpAddress = (DWORD)ImageRvaToVa(LoadedImage.FileHeader, BaseAddress, RVAExpDir, NULL); //get virtual adress for array string by him rva

printf("0x%lx -VA\n",VAExpAddress);

ExpTable=(IMAGE\_EXPORT\_DIRECTORY\*)VAExpAddress;

sName = (char\*)ImageRvaToVa(LoadedImage.FileHeader, BaseAddress, ExpTable->Name, NULL); //get virtual adress string name pe-file by him rva

printf("Name of PEF: %s\n", sName);

pNames = (char\*\*)ImageRvaToVa(LoadedImage.FileHeader, BaseAddress, ExpTable->AddressOfNames, NULL); //get virtual adress array string by him rva

nNames = ExpTable->NumberOfNames; //get quantity export name from table export

printf("Export:\n");

for(unsigned int i=0; i<nNames; i++)

{

pName = (char\*)ImageRvaToVa(LoadedImage.FileHeader, BaseAddress, (DWORD)\*pNames, NULL);//get virtual adress name with index i by him rva

printf("%s\n", pName);

\*pNames++;//go next string

}

}

else

{//if size entry directory equally zero

printf("No export directory\n");

}

UnMapAndLoad(&LoadedImage);//unloaded pe-file

return 0;

}

int Check\_Second\_Argument(char\* library\_name)

{

if(library\_name != NULL)

{

return 0;

}

return ERROR\_SECOND\_ARGUMENT;

}

int Get\_Import\_Function(PCSTR library\_name)

{

#ifdef TRIGGER\_USE\_MONITORING

char modName[MAX\_PATH];

int error\_flag = 0b0;

int temp\_err = 0;//

char\* process\_name = NULL;

int size\_process\_name = 0;//TEST\_SIZE\_PROCESS\_NAME;

DWORD pID = -1; //id process

HANDLE pHndl; //handle for process

HMODULE\* modHndls = NULL; //handle for modules

DWORD b\_alloc = 8, b\_needed = 0; //size for allocation memory for array handle for moduls(in quantity bytes)

if(Copy\_String\_In\_Char\_Array(&process\_name, (char\*)library\_name, &size\_process\_name) != 0) //copying name process in char arrray

{

return ERROR\_ALLOCATION\_MEMORY;

}

error\_flag = (error\_flag | Get\_Pid\_Need\_Procces\_By\_Name(&pID, process\_name, size\_process\_name));//get processID by name this process

if(error\_flag != 0)

{

return error\_flag;

}

if(pID == -1)

{

printf("this process dont founded\n");

return ERROR\_FOUND\_PROCESS;

}

printf("PID: %d\n",pID);//

pHndl = OpenProcess(PROCESS\_ALL\_ACCESS, FALSE, pID); //get handle for process by processID

while(1)

{

printf("iterarion in while\n");

modHndls=(HMODULE\*)malloc(b\_alloc); //allocation memory in quantity b\_alloc(size in byte)

if(EnumProcessModules(pHndl, modHndls, b\_alloc, &b\_needed) == 0) //get handle for every modules in specified dll(pHndl)

{

printf("%d\n",GetLastError());//

printf("Error: EnumProcessModules fail\n");

return ERROR\_ENUM\_PROCESS\_MODULS;

}

else//

{

printf("srabotal enum\n");//

}

// printf("%u %u\n", pID, pHndl);

// printf("%u %u\n",b\_alloc, b\_needed);

if(b\_needed <= b\_alloc)

{

printf("norm\n");//

break;

}

else

{

printf("malo\n");//

free(modHndls); //free past allocation memory

b\_alloc = b\_needed; //assign allocation size - needed quantity byte

}

}

// printf("a==%d == n==%d",b\_alloc,b\_needed);

for(int i = 0; i < ((b\_needed/sizeof(DWORD))); i++)

{

GetModuleBaseName(pHndl, modHndls[i], (LPSTR)modName, sizeof(modName)); //get name dll

printf("%u\t%s", modHndls[i], modName);

GetModuleFileName(modHndls[i], (LPSTR)modName, sizeof(modName)); //get full path for dll

// temp\_err = Get\_Export\_Function(modName); //get export function in this dll

printf("\t%s\n",modName);

}

#else

LOADED\_IMAGE LoadedImage;

PIMAGE\_IMPORT\_DESCRIPTOR ImportDescriptor; //pointer type for descriptor import

PIMAGE\_THUNK\_DATA OriginalFirstThunk;//rv for table name import

PIMAGE\_THUNK\_DATA FirstThunk; //rva for table adress import

PIMAGE\_THUNK\_DATA ThunkData; // tempurature for relativiry adress for table import

PIMAGE\_IMPORT\_BY\_NAME ImportFunctionName;// strings included name of dll

DWORD uSize;

if(!MapAndLoad(library\_name, NULL, &LoadedImage, TRUE, TRUE)) //load pe-file

{

printf("Need file dont find\n");

printf("Maybe incorrect second argument, please check this\n");

printf("Example correct second argument notepad.exe");

return ERROR\_UNKNOWN\_MAP\_AND\_LOAD;

}

ImportDescriptor = (PIMAGE\_IMPORT\_DESCRIPTOR)ImageDirectoryEntryToData(LoadedImage.MappedAddress, FALSE, IMAGE\_DIRECTORY\_ENTRY\_IMPORT, &uSize); //get value for descriptor for import table

while(ImportDescriptor->Name || ImportDescriptor->TimeDateStamp)

{

OriginalFirstThunk = (PIMAGE\_THUNK\_DATA)ImageRvaToVa(LoadedImage.FileHeader, LoadedImage.MappedAddress, ImportDescriptor->OriginalFirstThunk, NULL); //get virtual adress for import table

printf("\n\tOriginal First Thunk = %lx\n", (ULONG)((PUCHAR)OriginalFirstThunk - LoadedImage.MappedAddress));

FirstThunk = (PIMAGE\_THUNK\_DATA)ImageRvaToVa(LoadedImage.FileHeader, LoadedImage.MappedAddress, ImportDescriptor->FirstThunk, NULL); //get relativity virtual adress for import table

printf("\tFirst Thunk = %lx\n\n", (ULONG)((PUCHAR)FirstThunk - LoadedImage.MappedAddress));

printf("Name of function: \t\t\t\t");

printf("Library:\n");

ThunkData = OriginalFirstThunk; //save in tempurature data of virtual adress table import

if(ThunkData == NULL) //if virtual adress table import equally null

{

ThunkData = FirstThunk; //save in tempurature data of relativity virtual adress table import

}

while(ThunkData->u1.AddressOfData) //while adress of data exists

{

if(!IMAGE\_SNAP\_BY\_ORDINAL(ThunkData->u1.Ordinal)) //if ordinal number end

{

ImportFunctionName = (PIMAGE\_IMPORT\_BY\_NAME)ImageRvaToVa(LoadedImage.FileHeader, LoadedImage.MappedAddress, ThunkData->u1.AddressOfData, NULL);//get function name of need dll

printf("%-40s\t", ImportFunctionName->Name);

printf("%s\n",(PUCHAR)ImageRvaToVa(LoadedImage.FileHeader, LoadedImage.MappedAddress, ImportDescriptor->Name, NULL));

}

ThunkData++;

OriginalFirstThunk++;

FirstThunk++;

}

ImportDescriptor++;

}

UnMapAndLoad(&LoadedImage);//unloaded portable executable file

#endif

return 0;

}

int main(int argc, char\* argv[])

{

int error\_flag = 0b0;

if(Check\_Argc(argc) == ERROR\_QUANTITY\_ARGUMENT)

{

error\_flag = (error\_flag | ERROR\_QUANTITY\_ARGUMENT);

return error\_flag;

}

if(Check\_Second\_Argument(argv[1]) == ERROR\_SECOND\_ARGUMENT)

{

error\_flag = (error\_flag | ERROR\_SECOND\_ARGUMENT);

return error\_flag;

}

if((argv[2][0] == GET\_EXPORT\_FUNCTION) && (argv[2][1] == '\0'))

{

error\_flag = (error\_flag | Get\_Export\_Function(argv[1]));

printf("write enter for exit programm\n");

getch();

return error\_flag;

}

else

{

if((argv[2][0] == GET\_IMPORT\_FUNCTION) && (argv[2][1] == '\0'))

{

error\_flag = (error\_flag | Get\_Import\_Function(argv[1]));

printf("write enter for exit programm\n");

getch();

return error\_flag;

}

else

{

printf("Error: incorrect third argument\n");

printf("example correct argument - E");

error\_flag = (error\_flag | ERROR\_THIRD\_ARGUMENT);

}

}

printf("End programm\n");

getch();

return error\_flag;

}

***lec7.c***

#include <windows.h>

#include <imagehlp.h>

#include <stdio.h>

// #define

int main(int argc, char\* argv[])

{

    //my check

    if(argv[0] == NULL)

    {

        printf("argv err\n");

        return 0;

    }

    if(argc == 1)

    {

        printf("1)argc err\n");

        return 0;

    }

    if(argc == 2)

    {

        printf("2)argc tru\n");

        printf("%s\n",argv[0]);

        printf("%s\n",argv[1]);

        // return 0;

    }

    //end check need delete in release version

    LOADED\_IMAGE LoadedImage;

    PUCHAR BaseAddress;

    DWORD RVAExpDir, VAExpAddress;

    IMAGE\_EXPORT\_DIRECTORY\* ExpTable;

    char\* sName;

    DWORD nNames;

    char\* pName;

    char\*\* pNames;

    DWORD i;

    //Загружаем PE-файл

    if(!MapAndLoad("imagehlp", NULL, &LoadedImage, TRUE, TRUE))

    {

        printf("Something's wrong!\n");

        system("PAUSE");

        exit(1);

    }

    //Считываем базовый адрес загрузочного модуля

    BaseAddress=LoadedImage.MappedAddress;

    printf("0x%lx - Base Address\n",(unsigned long)BaseAddress);

    //Определяем относительный виртуальный адрес - RVA, таблицы экспорта

    RVAExpDir= LoadedImage.FileHeader->OptionalHeader.DataDirectory[IMAGE\_DIRECTORY\_ENTRY\_EXPORT].VirtualAddress;

    printf("0x%lx -RVA\n", RVAExpDir);

    //Определяем виртуальный адрес массива строк по его RVA

    VAExpAddress=(DWORD)ImageRvaToVa(LoadedImage.FileHeader,

    BaseAddress, RVAExpDir,NULL);

    printf("0x%lx -VA\n",VAExpAddress);

    ExpTable = (IMAGE\_EXPORT\_DIRECTORY\*)VAExpAddress;

    //Определяем виртуальный адрес строки - имени PE-файла,

    //по его RVA

    sName = (char\*)ImageRvaToVa(LoadedImage.FileHeader, BaseAddress, ExpTable->Name,NULL);

    printf("Name of PEF: %s\n",sName);

    //Определяем виртуальный адрес массива строк по его RVA

    pNames=(char\*\*)ImageRvaToVa(LoadedImage.FileHeader, BaseAddress, ExpTable->AddressOfNames,NULL);

    //Считываем количество экспортируемых имен из таблицы

    //экспорта

    nNames=ExpTable->NumberOfNames;

    // printf("Exported data: %s\n",pName);

    for(i = 0; i < nNames; i++)

    {

        //Определяем виртуальный адрес i-ого имени по его RVA

        pName = (char\*)ImageRvaToVa(LoadedImage.FileHeader, BaseAddress, (DWORD)\*pNames,NULL);

        printf("%s\n",pName);

        \*pNames++; //переходим к следующей строке

    }

    UnMapAndLoad(&LoadedImage);

    getch();

    return 0;

}

**Lab7a:**

***library\_1.c***

#include <windows.h>

extern \_\_declspec(dllexport)

int value\_one = 44;

\_\_declspec(dllexport)

int function\_one(int b)

{

return (b\*b);

}

\_\_declspec(dllexport)

int function\_two(int b)

{

return b\*b\*b;

}

***monitoring\_1.c***

#include <windows.h>

#include <psapi.h>

#include <stdlib.h>

#include <stdio.h>

#define TRIGGER\_TEST

#define ERROR\_ALLOCATION\_MEMORY 0b00000001

#define ERROR\_OPEN\_FILE\_TEMP\_TXT 0b00000010

#define ERROR\_ENUM\_PROCESS\_MODULS 0b00000100

#ifdef TRIGGER\_TEST

#define TEST\_PROCESS\_NAME "programm\_1.exe"

#define TEST\_SIZE\_PROCESS\_NAME 20

#endif

#define SIZE\_STRING\_TO\_SYSTEM\_COMMAND 31

#define CAPACITY\_DIGIT\_TO\_DECIMAL 10

#define QUANTITY\_NEED\_ITERATION\_PROGRAMM 5

#define TIME\_DELAY\_PROGRAMM\_IN\_THOUSANDTH\_OF\_SECOND 7000

#define FILE\_NAME\_FOR\_TEMP\_TXT "temp.txt"

#define READ\_ONLY "r"

#define FIRST\_CODE\_NUMBER\_TO\_INTEGER\_SYMBOL 0x30

#define LAST\_CODE\_NUMBER\_TO\_INTEGER\_SYMBOL 0x39

#define CONVERT\_CODE\_NUMBER\_FROM\_SYMBOL 0x0F

int Copy\_String\_In\_Char\_Array(char\* process\_name[], char copy\_string[], int size\_process\_name)

{

\*process\_name = (char\*)calloc(size\_process\_name,sizeof(char)); //allocation need memory

if(\*process\_name == NULL) //if allocation fail

{

return ERROR\_ALLOCATION\_MEMORY;

}

for(short int i = 0; i < size\_process\_name; i++)

{

(\*process\_name)[i] = copy\_string[i]; //copying in char array to char array

}

return 0;

}

int Get\_Pid\_Need\_Procces\_By\_Name(int\* pID, char\* process\_name, int size\_process\_name)

{

FILE\* file = NULL; //pointer for file

char\* system\_command = NULL; // char array for system command string

char symbol = '0';

system\_command = (char\*)calloc((SIZE\_STRING\_TO\_SYSTEM\_COMMAND+size\_process\_name),sizeof(char));//allocation memmory

if(system\_command == NULL)

{

return ERROR\_ALLOCATION\_MEMORY;

}

sprintf(system\_command,"tasklist | findstr %s > temp.txt", process\_name); //fill system command by need command string

system(system\_command); //execute system command

free(system\_command); //free memory for char array system command

system\_command = NULL;

file = fopen(FILE\_NAME\_FOR\_TEMP\_TXT,READ\_ONLY);//open temp txt file in ode read

if(file == NULL)

{

return ERROR\_OPEN\_FILE\_TEMP\_TXT;

}

do

{

symbol = getc(file); //read one symbol from temp file

} while((symbol != ' ') && (!feof(file))); //while dont run first field in result tasklist command

do

{

symbol = getc(file); //read one symbol from temp file while not digital

} while((symbol < FIRST\_CODE\_NUMBER\_TO\_INTEGER\_SYMBOL) || (symbol > LAST\_CODE\_NUMBER\_TO\_INTEGER\_SYMBOL) || feof(file));

do

{

(\*pID) = ((\*pID) + (symbol & CONVERT\_CODE\_NUMBER\_FROM\_SYMBOL)); // convert digital char in digital int

(\*pID) = ((\*pID) \* CAPACITY\_DIGIT\_TO\_DECIMAL); //shift right number of digit in decimal system

symbol = getc(file); //read one symbol from temp file while read digit char

}while((symbol >= FIRST\_CODE\_NUMBER\_TO\_INTEGER\_SYMBOL) && (symbol <= LAST\_CODE\_NUMBER\_TO\_INTEGER\_SYMBOL) && !feof(file));

(\*pID) = ((\*pID) / CAPACITY\_DIGIT\_TO\_DECIMAL); // shift left number of digit in decimal system

fclose(file); // close file

return 0;

}

int main()

{

char modName[MAX\_PATH]; //char array for path modules name

char\* process\_name = NULL;

int error\_flag = 0b0;

int size\_process\_name = 0;

int iteration = 0;

DWORD pID = 0; //id process

HANDLE pHndl; //handle for process

HMODULE\* modHndls; //handle for modules

DWORD b\_alloc = 8, b\_needed = 0; //size for allocation memory for array handle for moduls(in quantity bytes)

#ifdef TRIGGER\_TEST

size\_process\_name = TEST\_SIZE\_PROCESS\_NAME; // test quantity symbol in process name

#endif

if(Copy\_String\_In\_Char\_Array(&process\_name, TEST\_PROCESS\_NAME, size\_process\_name) != 0) //copying name process in char arrray

{

return ERROR\_ALLOCATION\_MEMORY;

}

error\_flag = (error\_flag | Get\_Pid\_Need\_Procces\_By\_Name(&pID, process\_name, size\_process\_name)); //get processID by name this process

if(error\_flag != 0)

{

return error\_flag;

}

pHndl = OpenProcess(PROCESS\_ALL\_ACCESS, FALSE, pID); //get handle for process by processID

do

{

while(1)

{

modHndls=(HMODULE\*)malloc(b\_alloc); //allocation memory in quantity b\_alloc(size in byte)

if(EnumProcessModules(pHndl, modHndls, b\_alloc, &b\_needed) == 0) //get handle for every modules in specified dll(pHndl)

{

return ERROR\_ENUM\_PROCESS\_MODULS;

}

// printf("%u %u\n", pID, pHndl);

// printf("%u %u\n",b\_alloc, b\_needed);

if(b\_needed <= b\_alloc) // if allocation memory less need

{

break;

}

else

{

free(modHndls); //free past allocation memory

b\_alloc = b\_needed; //assign allocation size - needed quantity byte

}

}

for(int i = 0; i < ((b\_needed/sizeof(DWORD))); i++)

{

GetModuleBaseName(pHndl, modHndls[i], (LPSTR)modName, sizeof(modName)); //get name dll

printf("%u\t%s", modHndls[i], modName);

GetModuleFileName(modHndls[i], (LPSTR)modName, sizeof(modName)); //get full path for dll

printf("\t:%s\n",modName);

}

Sleep(TIME\_DELAY\_PROGRAMM\_IN\_THOUSANDTH\_OF\_SECOND); //program execution delay to allow time for the programm loaded/unloaded the modules(library.dll)

iteration++; //increment iteration

}while(iteration < QUANTITY\_NEED\_ITERATION\_PROGRAMM);

printf("End monitoring\n");

getch();

return error\_flag;

}

***programm\_1.c***

#include <windows.h>

#include <stdio.h>

#define LECTION\_EXAMPLE 1

#define QUANTITY\_NEED\_ITERATION\_PROGRAMM 5

#define TIME\_DELAY\_PROGRAMM\_IN\_THOUSANDTH\_OF\_SECOND 5000

#if LECTION\_EXAMPLE

typedef int (\*function)(int); //new type that is pointer function for return integer

#endif

int main()

{

int iteration = 0; //counter for exiting the program

HINSTANCE hInst; //handle to the instance (void pointer for differently objects)

do

{

hInst = LoadLibrary("library\_1.dll"); //loading the specified module(library.dll) in the program

printf("loaded\n"); //message for clarity about what library loaded

Sleep(TIME\_DELAY\_PROGRAMM\_IN\_THOUSANDTH\_OF\_SECOND); //program execution delay to allow time for the monitor to read the modules(with library.dll)

#if LECTION\_EXAMPLE

function pf; //variable for first funtion

function pg; //variable for second funtion

int\* pa; //variable for first value

pa=(int\*)GetProcAddress(hInst, "value\_one"); //extract adress variable "value\_one" from dll which it points to hInst

pf=(function)GetProcAddress(hInst, "function\_one"); //same as above but for the "function\_one"

pg=(function)GetProcAddress(hInst, "function\_two"); //same as above but for the "function\_two"

printf("%i %i %i\n",\*pa,pf(4),pg(5)); //print all gets

#else

FARPROC return\_value\_one = NULL; //pointer type reterned import function

FARPROC return\_function\_one = NULL;

FARPROC return\_function\_two = NULL;

hInst = LoadLibrary("library\_1.dll");

return\_value\_one = GetProcAddress(hInst, "value\_one");

return\_function\_one = GetProcAddress(hInst, "function\_one");

return\_function\_two = GetProcAddress(hInst, "function\_two");

printf("1)%i\n", \*(int\*)return\_value\_one); //dereferencing pointer for value\_one

printf("2)%i\n", return\_function\_one(4));

printf("3)%i\n", return\_function\_two(5));

Sleep(5000);

#endif

FreeLibrary(hInst); //unloaded module which points to hInst

printf("unloaded\n"); //message for clarity about what library loaded

Sleep(TIME\_DELAY\_PROGRAMM\_IN\_THOUSANDTH\_OF\_SECOND); //program execution delay to allow time for the monitor to read the modules(without library.dll)

iteration++; //increment iteration

} while(iteration < QUANTITY\_NEED\_ITERATION\_PROGRAMM);

printf("End programm\n");

getch();

return 0;

}

***Выполнение программы:***

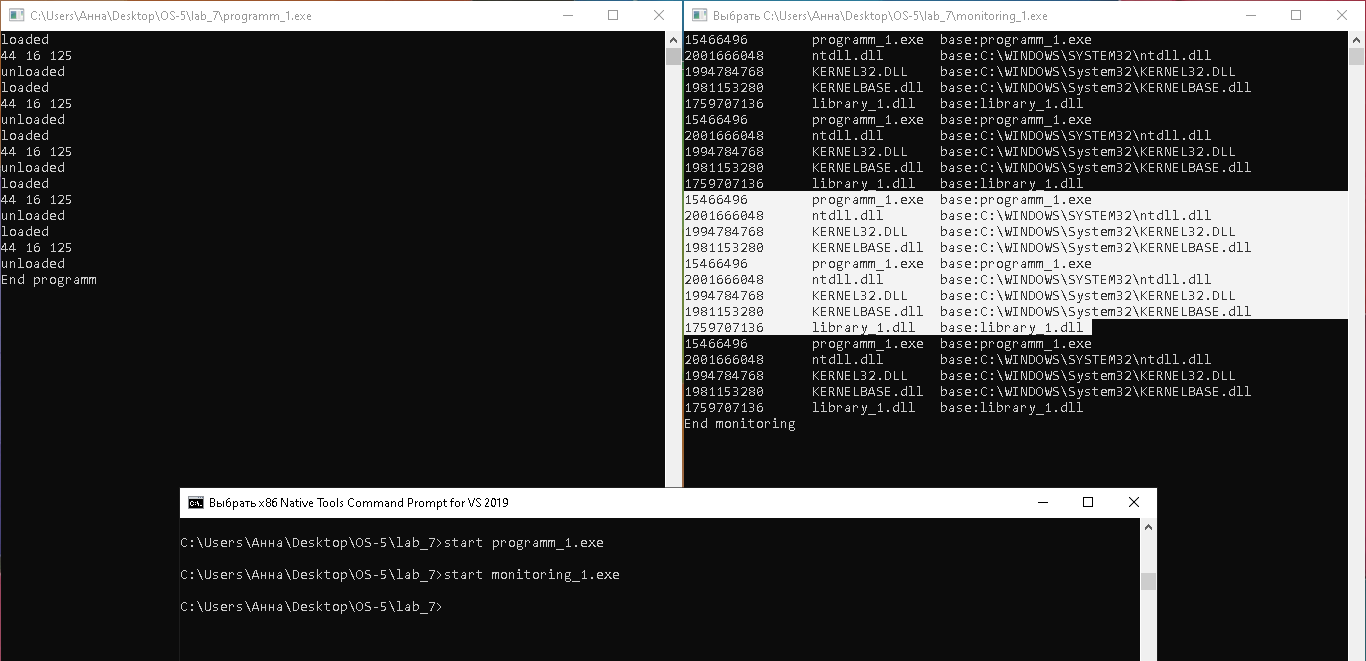


Рис.№1 «Результат одновременного выполнения пяти итераций ***programm\_1.c*** и ***monitoring\_1.c***»

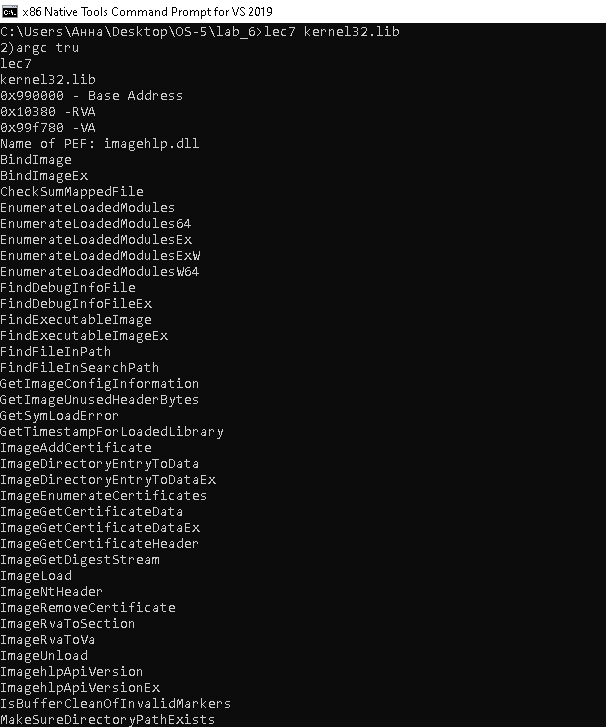


Рис.№2 «Результат выполнения программы ***lec7.c***, являющейся тестом лекции»

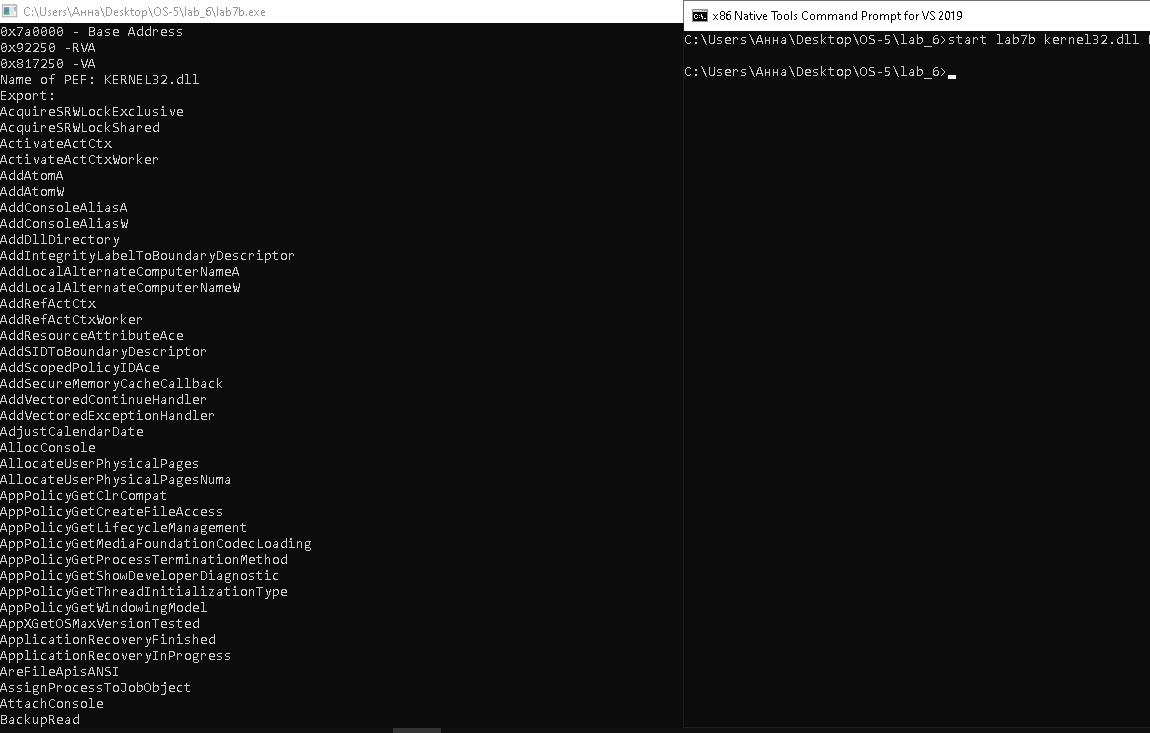


Рис.№3 «Результат выполнения программы ***lab7b.c*** ,в качестве входных параметров экспорт kernel32.dll»

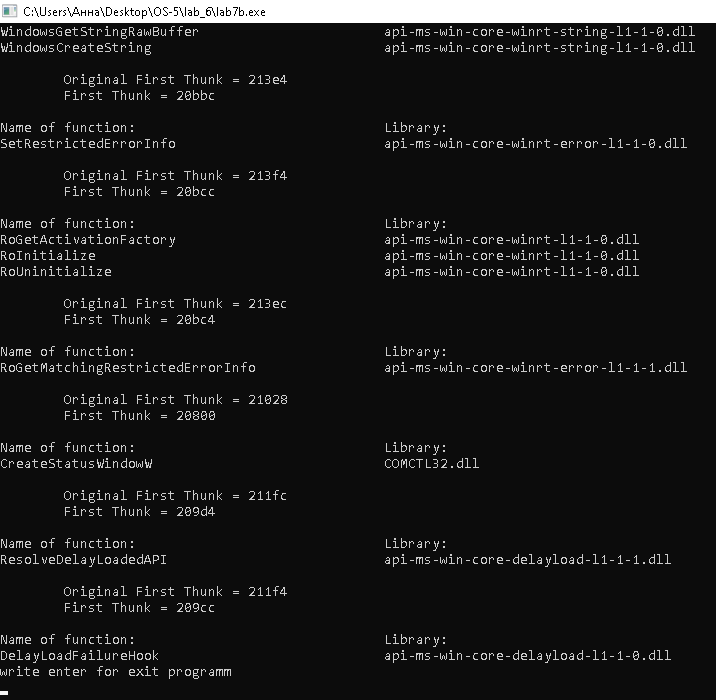


Рис.№4 «Результат выполнения программы ***lab7b.c*** ,в качестве входных параметров импорт notepade.exe»

***Заключение***

Лабораторная работа 7а: приобрел навыки разработки приложений с использованием dll и освоил интерфейс PSAPI.

Лабораторная работа 7b: закрепил сведения о структуре исполнямых файлов, получил навыки работы с интерфейсом IMAGEHLP.