Выполнила: Белоусова Е., ИП-911

Задача

Цель: познакомиться с принципами обмена данными между процессами на основе отображения файлов в память.
Задание:
• Протестировать программы Лекции12.
• Написать программу для просмотра PE-файлов на основе отображения файлов в память.

Описание работы программы

Протестируем программы из лекции.

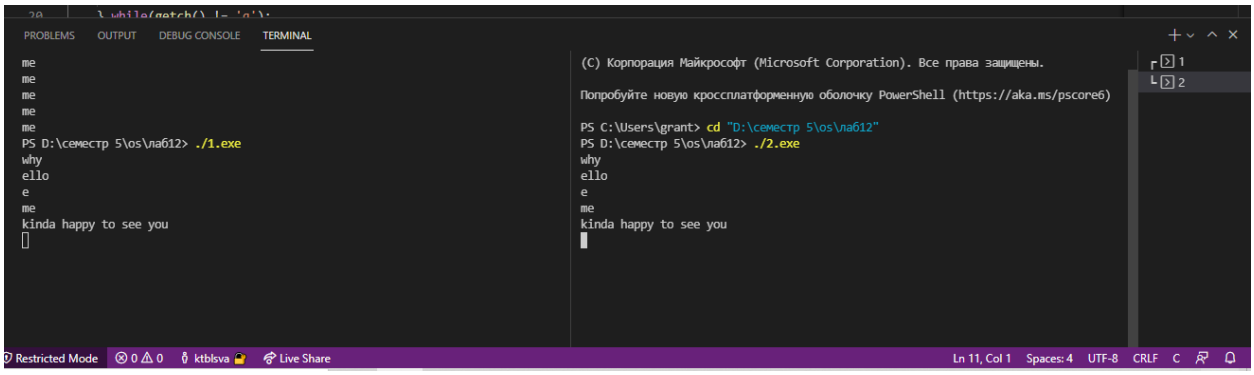


Рисунок 1 тест

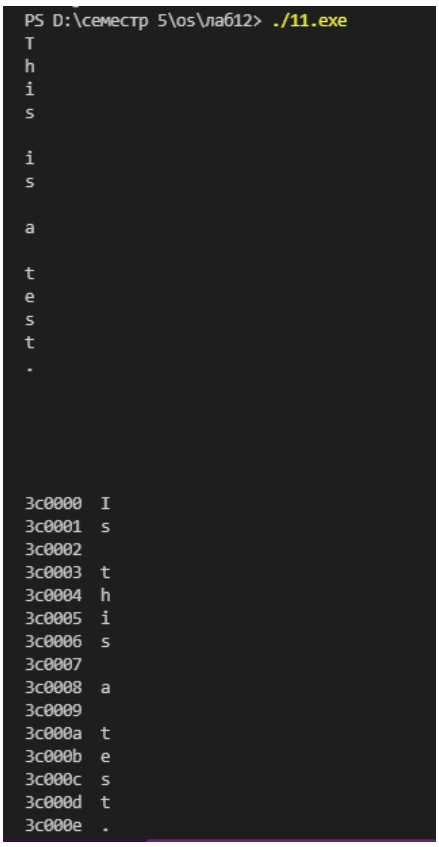
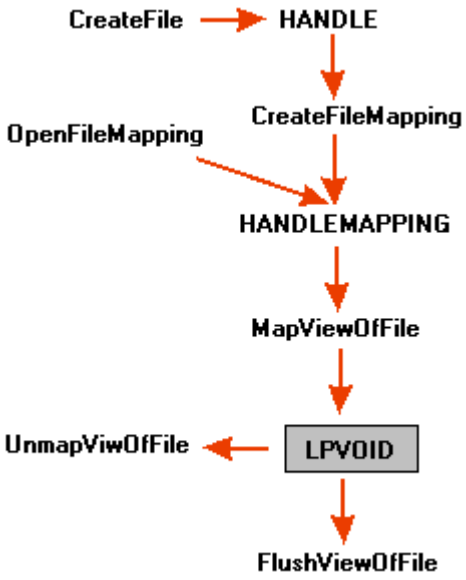
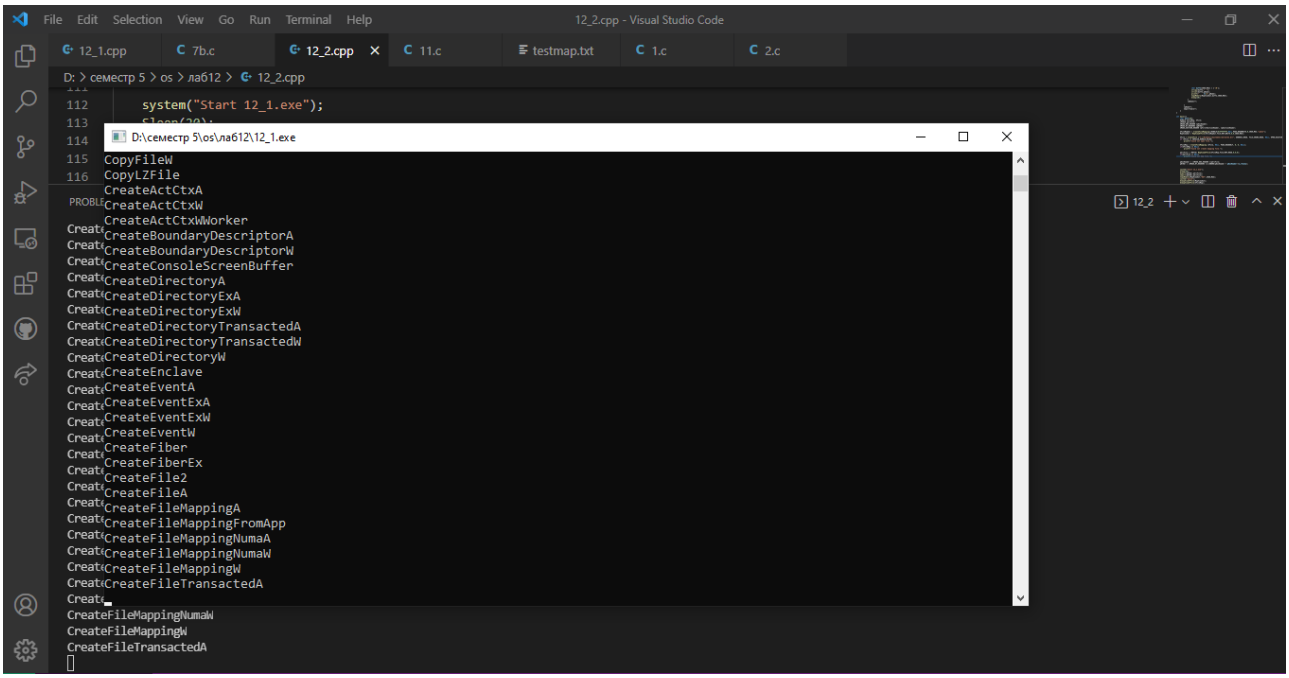


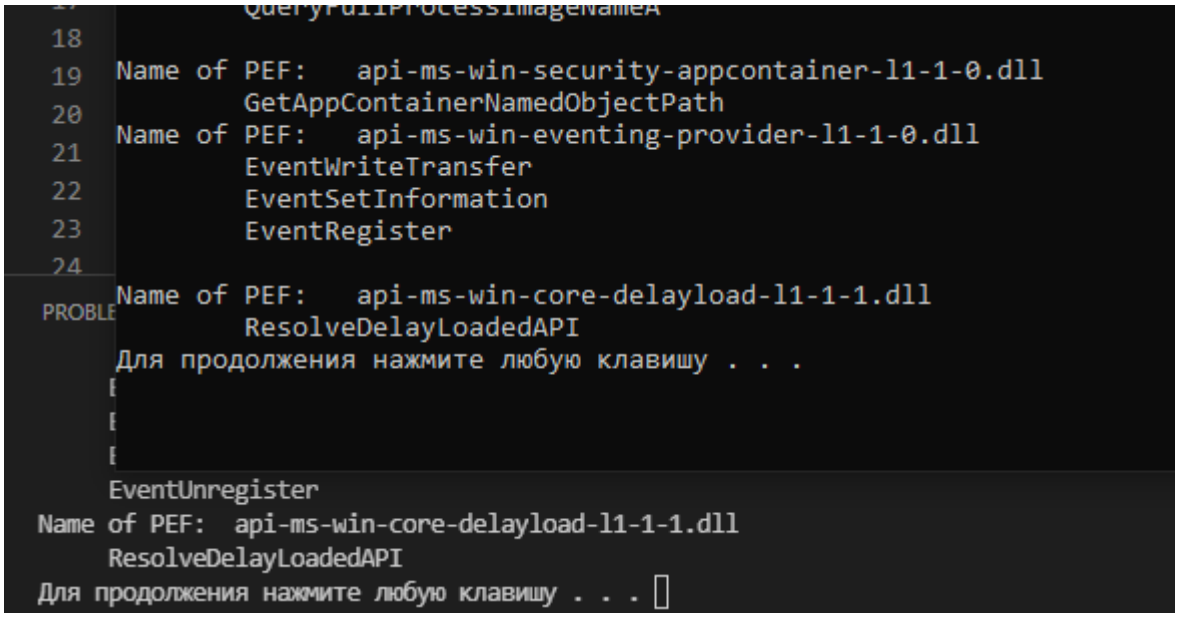
Рисунок 2 тест

Напишем программу для просмотра PE-файлов на основе отображения файлов в память. Отобразим kernell32.dll в память. Для этого создаем файл с флагами FILE_SHARE_READ и OPEN_EXISTING. Также будем использовать существующий swap-файл. Он отображается в память, и мы используем его для обмена данными.



Сначала файл открывается с помощью **CreateFile()**, и если открытие прошло успешно, то идентификатором файла можно воспользоваться для создания отображенного файла **CreateFileMapping()**, после которой мы получим идентификатор отраженного файла, на основе которого мы можем проецировать файл в память с помощью функции **MapViewOfFile()**. Эта функция даст нам указатель **LPVOID**, который используется для последующих операций, в том числе и по отмене проецированного файла **UnmapViewOfFile()** или принудительной записи на диск **FlushViewOfFile()**. В том случае, если с отображенным файлом будут работать несколько приложений одно приложение создает файл проецированный на память **CreateFileMaping()**, а остальные открывают эту проекцию **OpenFileMapping()**.





Листинг

```
//12_2.cpp

#include <windows.h>

#include <stdio.h>

#include <imagehlp.h>

#include <process.h>

HANDLE hFileMapOut;

LPVOID MapViewOut;

//cmd.exe /k "C:\Program Files (x86)\Microsoft Visual Studio\2019\Enterprise\VC\Auxiliary\Build\vcvars32.bat" `& powershell

void Export(IMAGE_NT_HEADERS *pNtHdr, LPVOID pSrcFile){

    char buff[CHAR_MAX] = {'\0'};

    IMAGE_EXPORT_DIRECTORY* ExpTable;

    char *pName, *sName, **pNames;

    DWORD nNames;

    DWORD RVAExpDir, VAExpAddress;

    int i;


    RVAExpDir = pNtHdr->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress;

    VAExpAddress = (DWORD)ImageRvaToVa(pNtHdr, pSrcFile, RVAExpDir, NULL);
```

```
    ExpTable=(IMAGE_EXPORT_DIRECTORY*)VAExpAddress;

    sName=(char*)ImageRvaToVa(pNtHdr,pSrcFile,ExpTable->Name,NULL);
    strcat(buff,"Name of PEF:  ");
    strcat(buff,sName);
    printf("Name of PEF:  %s\n",sName);
    CopyMemory(MapViewOut,buff,CHAR_MAX);
    Sleep(20);

    pNames=(char**)ImageRvaToVa(pNtHdr,pSrcFile,ExpTable->AddressOfNames,NULL);
    nNames=ExpTable->NumberOfNames;
    printf("Exported data: \n");
    CopyMemory(MapViewOut,"Exported data: ",CHAR_MAX);
    Sleep(20);
    for(i = 0; i < nNames; i++){
            pName = (char*)ImageRvaToVa(pNtHdr,pSrcFile,(DWORD)*pNames,NULL);
            printf("%s\n",pName);
            CopyMemory(MapViewOut,pName,CHAR_MAX);
            Sleep(20);
            *pNames++;
    }
}

void Import(IMAGE_NT_HEADERS *pNtHdr, LPVOID pSrcFile){
    char *pName, *sName, **pNames;
    DWORD nNames;
    DWORD RVAExpDir, VAExpAddress;
    IMAGE_IMPORT_DESCRIPTOR* ImportTable;

    RVAExpDir = pNtHdr-
>OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_IMPORT].VirtualAddress;
    VAExpAddress = (DWORD)ImageRvaToVa(pNtHdr,pSrcFile,RVAExpDir,NULL);
    ImportTable=(IMAGE_IMPORT_DESCRIPTOR*)VAExpAddress;
```

```
printf("Imported data: \n");
CopyMemory(MapViewOut,"Imported data: ",CHAR_MAX);
Sleep(20);

while(ImportTable->Name != NULL){
	pNames=(char**)ImageRvaToVa(pNtHdr,pSrcFile,ImportTable->FirstThunk,NULL);
	sName=(char*)ImageRvaToVa(pNtHdr,pSrcFile,ImportTable->Name,NULL);
	SecureZeroMemory(MapViewOut,CHAR_MAX);
	char buff[CHAR_MAX] = {'\0'};
	printf("Name of PEF:  %s\n",sName);
	strcpy(buff,"Name of PEF:   ");
	strcat(buff,sName);
	CopyMemory(MapViewOut,buff,CHAR_MAX);
	Sleep(20);

	while(pName != 0){
		pName = (char*)ImageRvaToVa(pNtHdr,pSrcFile,(DWORD)*pNames+2,NULL);
		if(pName != 0){
			char buff2[CHAR_MAX] = {'\0'};
			strcpy(buff2,"    ");
			strcat(buff2,pName);
			printf("    %s\n",pName);
			CopyMemory(MapViewOut,buff2,CHAR_MAX);
			Sleep(20);
		}
		*pNames++;

	}
	*pName++;
	ImportTable++;
}
```

```
}

int main(){
    char buff[10];
    HANDLE hFileMap, hFile;
    LPVOID pSrcFile;
    IMAGE_DOS_HEADER *pDosHeader;
    IMAGE_NT_HEADERS *pNtHdr;
    IMAGE_SECTION_HEADER *pFirstSectionHeader, *pSectionHeader;

    hFileMapOut =
CreateFileMapping((HANDLE)0xFFFFFFFF,NULL,PAGE_READWRITE,0,CHAR_MAX,"Lab12");
    MapViewOut = MapViewOfFile(hFileMapOut,FILE_MAP_WRITE,0,0,CHAR_MAX);

    hFile = CreateFile ("C:\\Windows\\System32\\kernel32.dll", GENERIC_READ, FILE_SHARE_READ, NULL,
OPEN_EXISTING, 0, NULL);
    if (hFile == INVALID_HANDLE_VALUE)
            printf("Could not open file.");

    hFileMap = CreateFileMapping (hFile, NULL, PAGE_READONLY, 0, 0, NULL);
    if(hFileMap == NULL)
            printf("Could not create mapping file.");

    pSrcFile = (PBYTE) MapViewOfFile(hFileMap,FILE_MAP_READ,0,0,0);
    if(pSrcFile == NULL)
            printf("Could not map file.");

    pDosHeader = (IMAGE_DOS_HEADER *)pSrcFile;
    pNtHdr = (IMAGE_NT_HEADERS *)((DWORD)pDosHeader + pDosHeader->e_lfanew);

   system("Start 12_1.exe");
```

```
    Sleep(20);
    Export(pNtHdr,pSrcFile);
    Import(pNtHdr,pSrcFile);
    CopyMemory(MapViewOut,"OUT",CHAR_MAX);
    system("pause");
    UnmapViewOfFile(MapViewOut);
    UnmapViewOfFile(hFileMap);
    CloseHandle(hFileMapOut);
    CloseHandle(hFileMap);
    return 0;
}

//12_1.cpp
#include <windows.h>
#include <stdio.h>

int main(){
    HANDLE hFileMap;
    LPVOID lpMapView;
    char buff[80];

    hFileMap = OpenFileMapping(FILE_MAP_READ,TRUE,"Lab12");
    lpMapView = MapViewOfFile(hFileMap,FILE_MAP_READ,0,0,CHAR_MAX);
    while(1){
            Sleep(20);
            CopyMemory(buff,lpMapView,80);
            if (strcmp(buff, "OUT") == 0)
                    break;
            if (buff != NULL) printf("%s\n",buff);
    }
    UnmapViewOfFile(lpMapView);
    CloseHandle(hFileMap);
```

```
    system("PAUSE");
    return 0;
}
```