Правительство Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»
(НИУ ВШЭ)

Московский институт электроники и математики им. А.Н. Тихонова

ОТЧЕТ
О ПРАКТИЧЕСКОЙ РАБОТЕ № 4.1
по дисциплине «Системное программирование» Основы откладки на ассемблере. Объекты ОС.

Студент гр. БИБ214
Е.Р.Портнягин
«29» апреля 2024 г.

Руководитель
Преподаватель
_____ Д.В, Смирнов
«___» _____ 2024 г.

Москва 2024

СОДЕРЖАНИЕ

1 Задание на практическую работу

Данное практическое задание направлено на изучение принципов работы компьютеров, основ архитектуры, таких как модель фон Неймана и архитектура x86/x64, а также на изучение взаимодействия с объектами в операционной системе Windows. Помимо этого будет проведён анализ программ с применением отладчика на уровне ассемблера и обнаружение проблем с помощью инструментов Sysinternals.

2 Ход работы

Загружаю файлы. (Рисунок 1)

ProcessMonitor	01.05.2024 20:46	Папка с файлами	
snapshot_2024-04-11_18-47	01.05.2024 20:42	Папка с файлами	
array_heap.c	09.02.2021 16:08	Исходный файл C	1 КБ
array_stack.c	09.02.2021 16:08	Исходный файл C	1 КБ
file.c	01.05.2024 20:50	Исходный файл C	17 КБ
file.exe	01.05.2024 20:54	Приложение	55 КБ
file.h	01.05.2024 20:42	Исходный файл C...	4 КБ
processhacker-2.39-setup.exe	01.05.2024 20:40	Приложение	2 215 КБ

Рисунок 1 – скачанные файлы

Открываю file.c и меняю значение в chFileName на "EgorPortnyagin.txt". (Рисунок 2)

```c
int main()
{
    unsigned long len = 0;
    int ret = LAB_OK;
    char *buf = NULL;
    char *command = NULL;
    char *chFileRead = NULL;
    char chFileName[] = {"EgorPortnyagin.txt"};
    char chFileToWrite[] = {"Hello world\n"};
    unsigned long i = 0;
    MyDescriptor hFile = INVALID_MYDESCRIPTOR_VALUE;

    // ?????????? ?????? ??? "?????" ????? ? ????
    buf = (char *)malloc(LAB_MAX_BUF_SIZE + 1);

    // ?????????? ?????? ??? ????? ? ?????????
```

Рисунок 2 – меняю chFileName

Компилирию file.c. (Рисунок 3)

```
.02.2021  16:08               490 array_heap.c
.02.2021  16:08               358 array_stack.c
.05.2024  20:50            16 584 file.c
.05.2024  20:42             4 023 file.h
.05.2024  20:40         2 267 848 processhacker-2.39-setup.exe
.05.2024  20:46    <DIR>          ProcessMonitor
.05.2024  20:42    <DIR>          snapshot_2024-04-11_18-47
               6 файлов      8 811 806 байт
               4 папок  12 651 028 480 байт свободно

\Users\user\Desktop\hw8>gcc -m32 file.c -o file.exe

\Users\user\Desktop\hw8>
```

Рисунок 3 – компиляция

Загружаю скомпилированный код в x64dbg, убрав заранее опцию “Загрузка системной DLL”. (Рисунок 4)

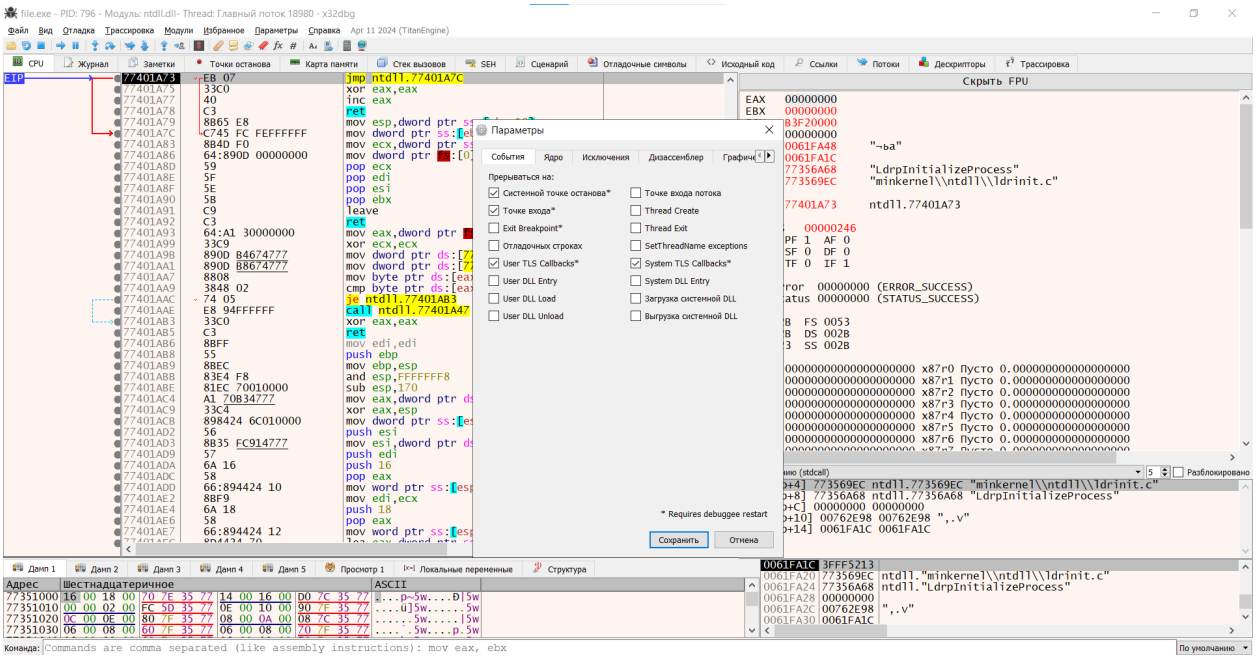

Рисунок 4 – заупск дебаггера

Выполняю до EntryPoint. Он находится по адресу 004012E0 (Рисунок 5)

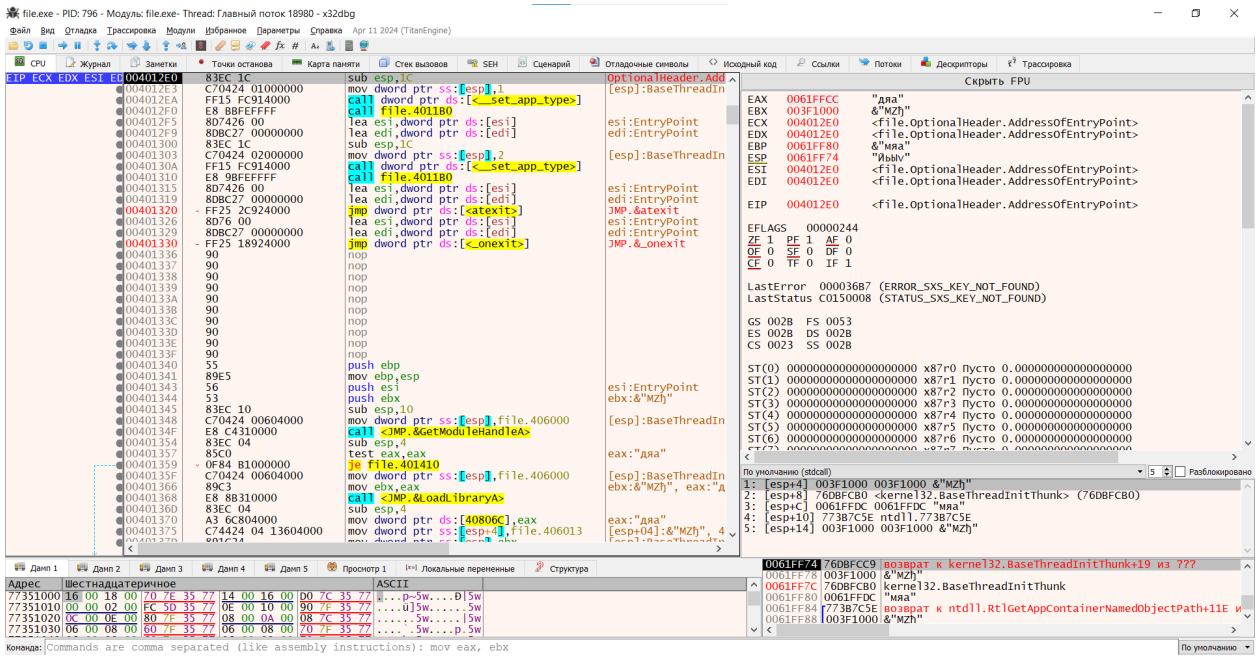

Рисунок 5 – EntryPoint

Ставлю breakpoint на main. (Рисунок 6)

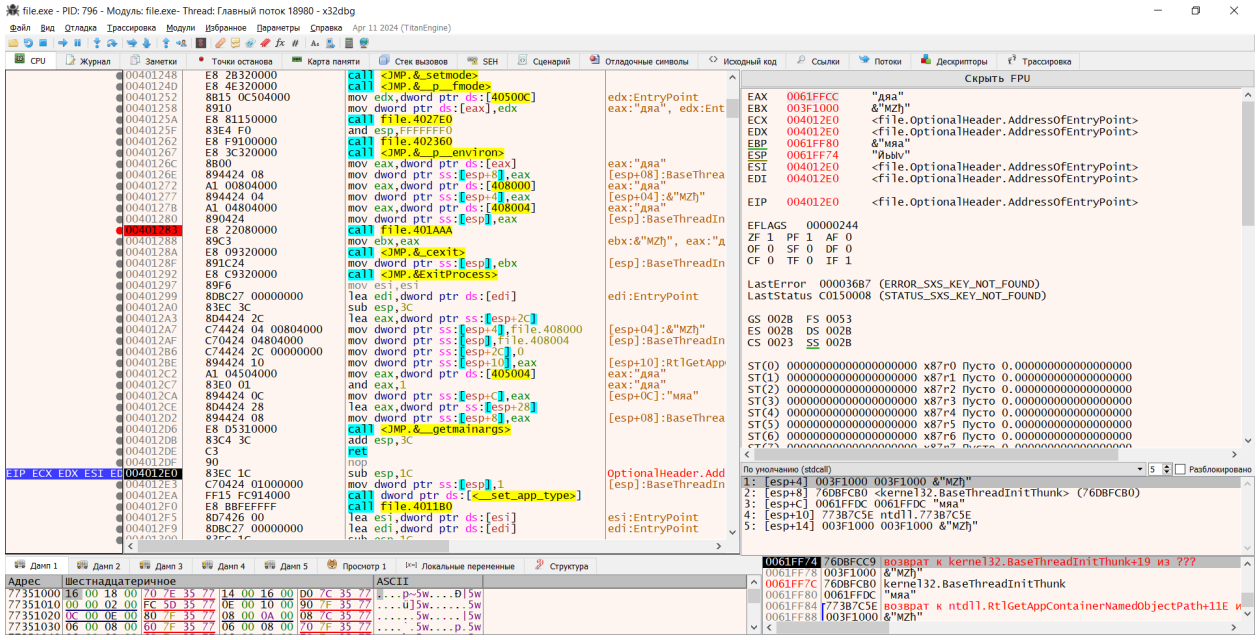

Рисунок 6 - Функция main

Выполняю до malloc в main. Сразу за malloc ставлю брейкпоинты, в EAX можно увидеть адреса выделенной памяти. Первый адрес и второй адрес соответственно: 010D0E68, 010D0F08. Один раз malloc вызывается для выделения памяти под buf, второй раз под command (Рисунок 7, Рисунок 8)

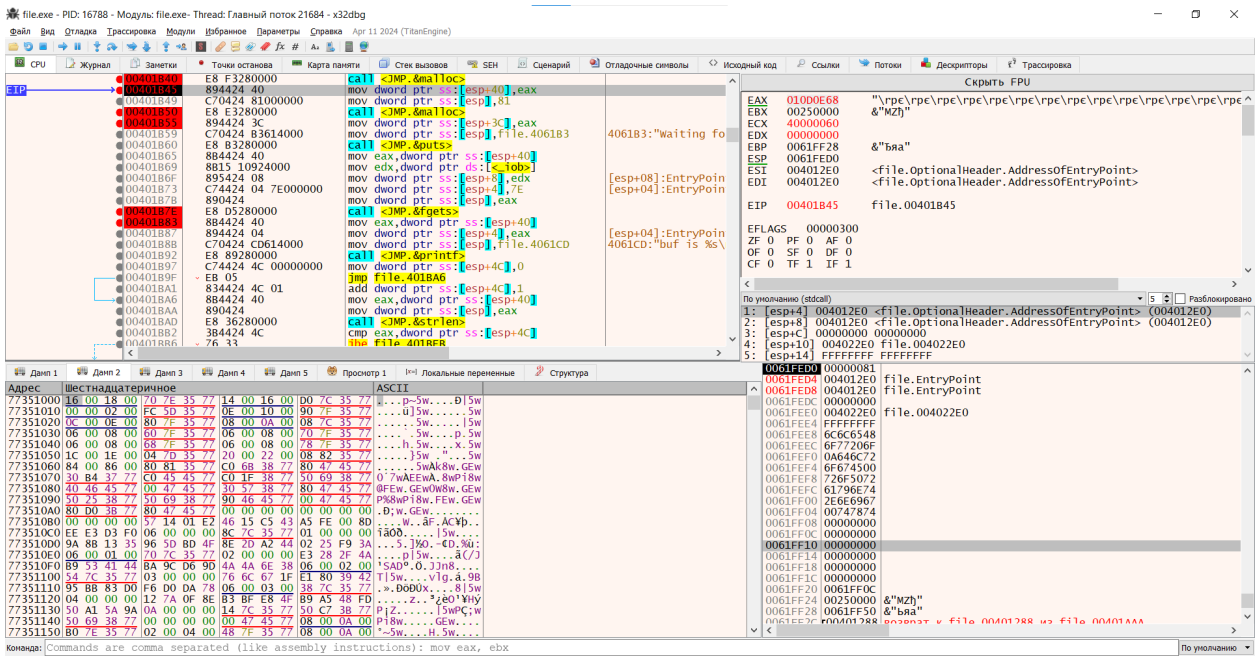

Рисунок 7 - первый breakpoint

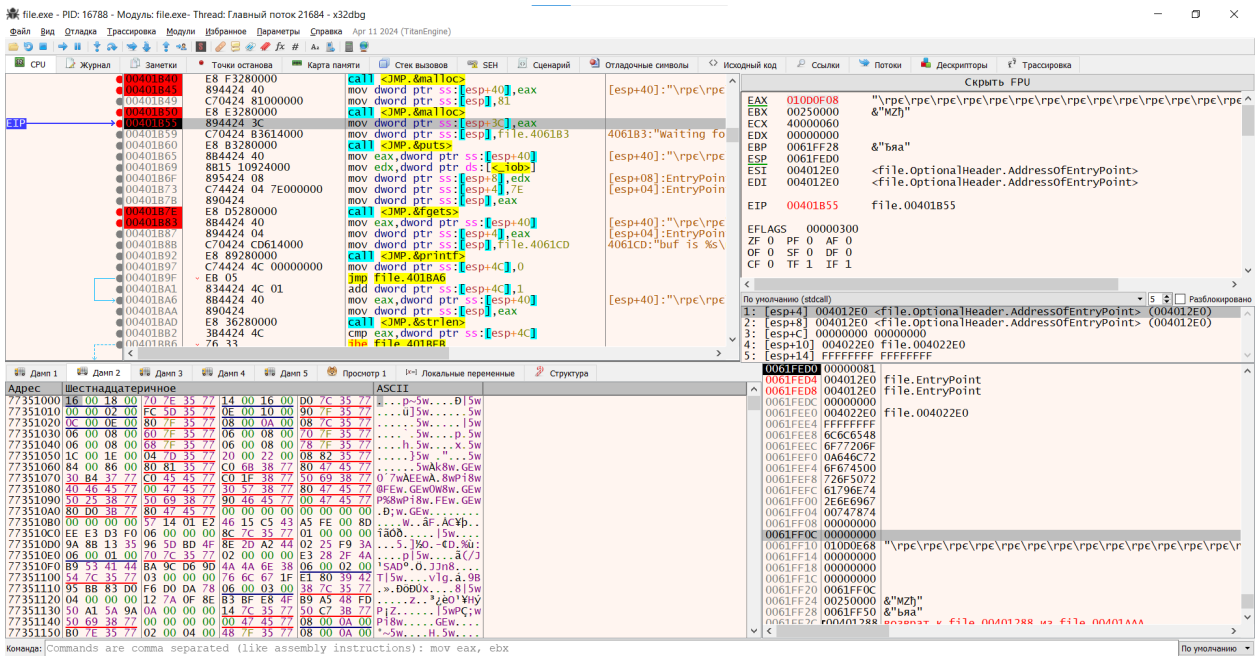

Рисунок 8 – второй breakpoint

fgets добавит “\0” символ в строку автоматически после спецсимвола «\n». (Рисунок 9)

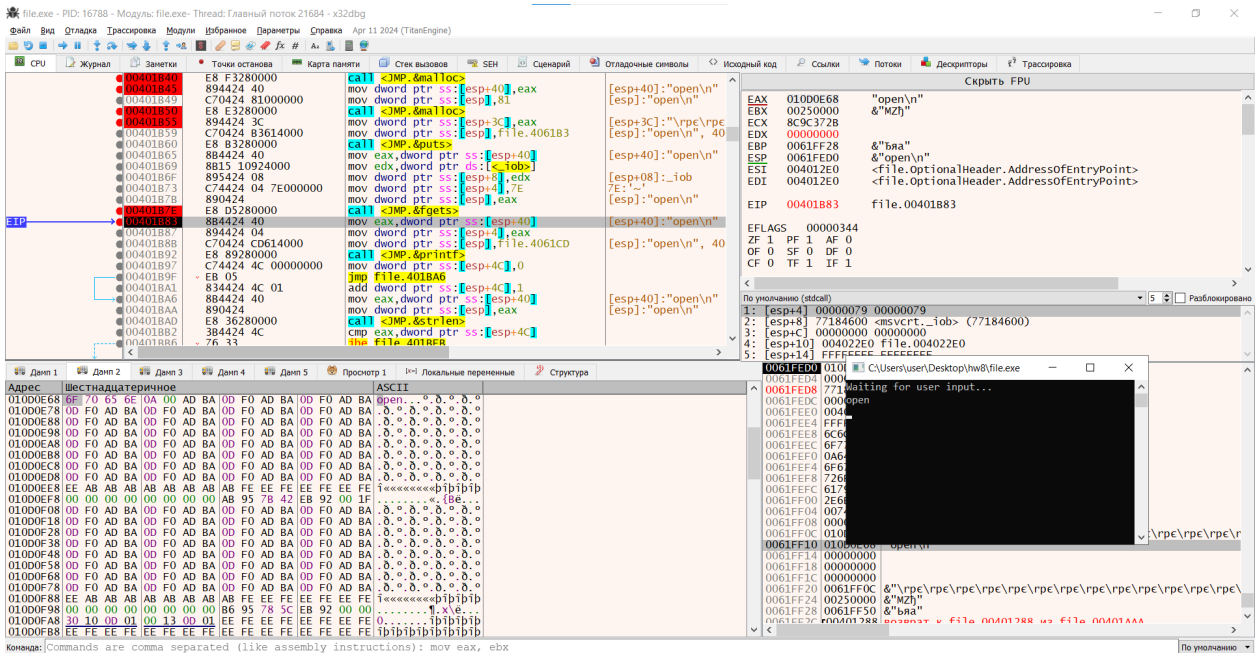

Рисунок 9 - Содержимое памяти по указателю buf после fgets

Если в input программы передать “open”, то в дальнейшем вызовется функция CreateFileA. (Рисунок 10)

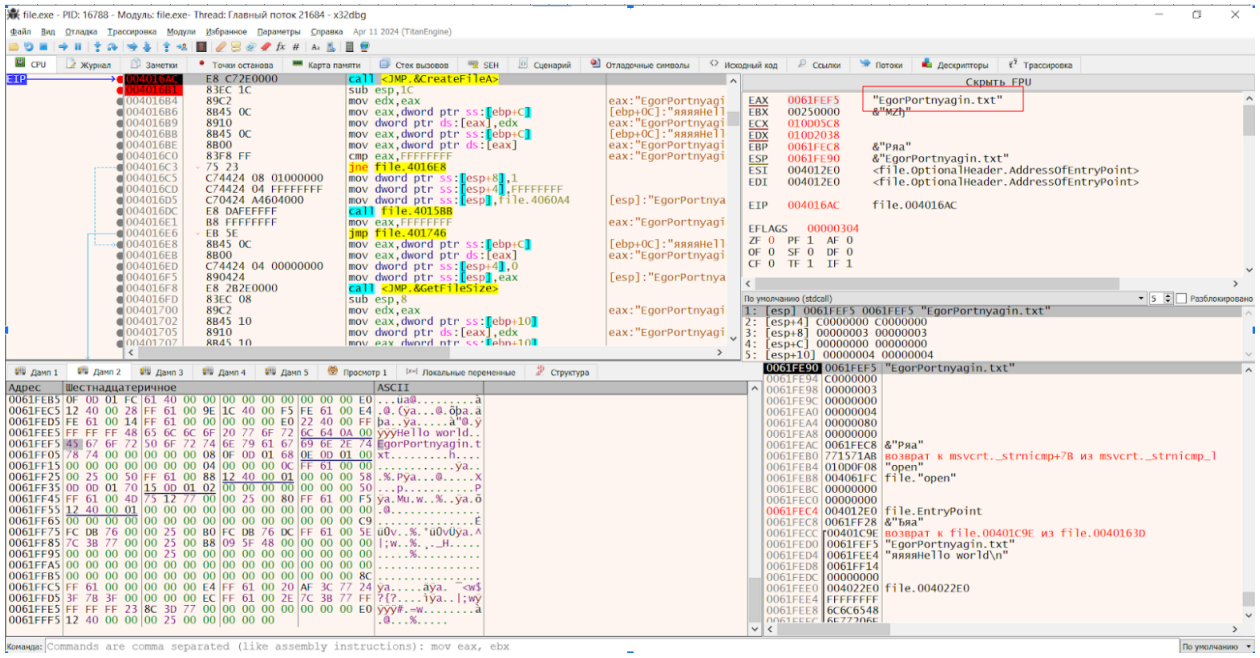

Рисунок 10 - Стек и регистр EAX перед CreateFileA

После CreateFileA, в EAX будет записан хэндл открытого файла, в данном случае 00000124. (Рисунок 11)

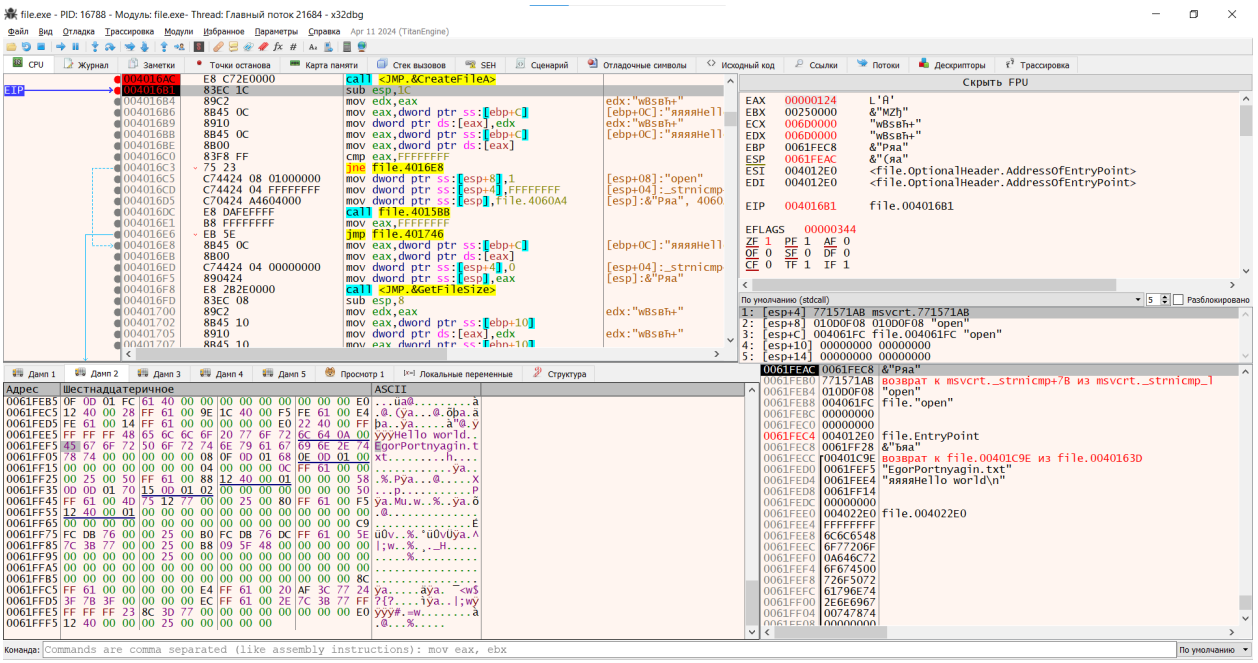

Рисунок 11 - Регистр EAX, хранящий хэндл файла

Точно такой же хэндл будет показан в Process Hacker. (Рисунок 12)

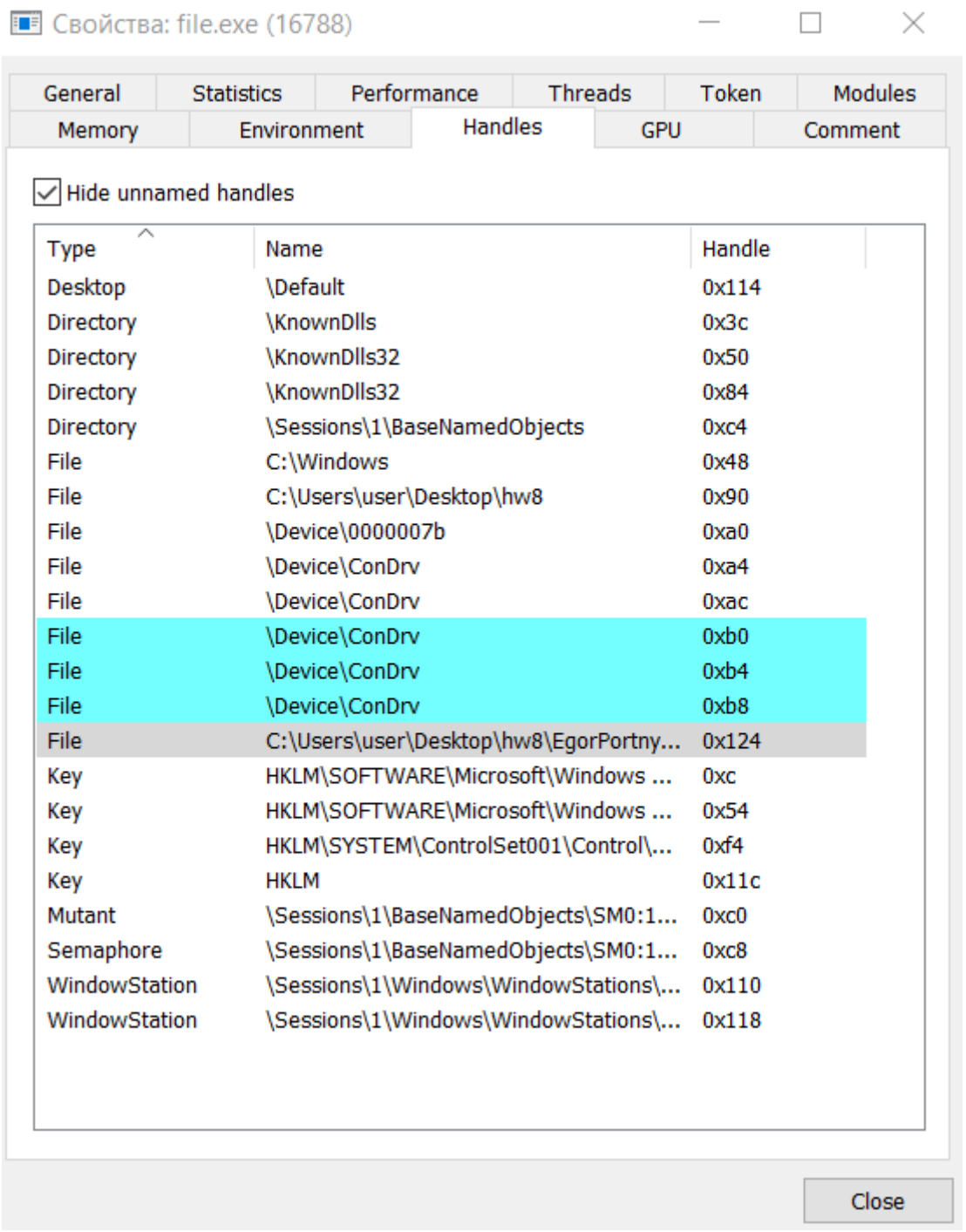

Рисунок 12 - Хэндлы программы в Process Hacker

В программе Process Monitor можно увидеть файлы, которые открыла наша программа. (Рисунок 13)

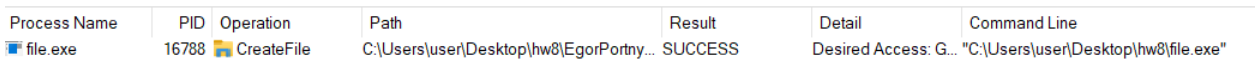

Рисунок 13 – открытие файла в Process Monitor

Повторяю то же самое для WriteFile. (Рисунок 14)

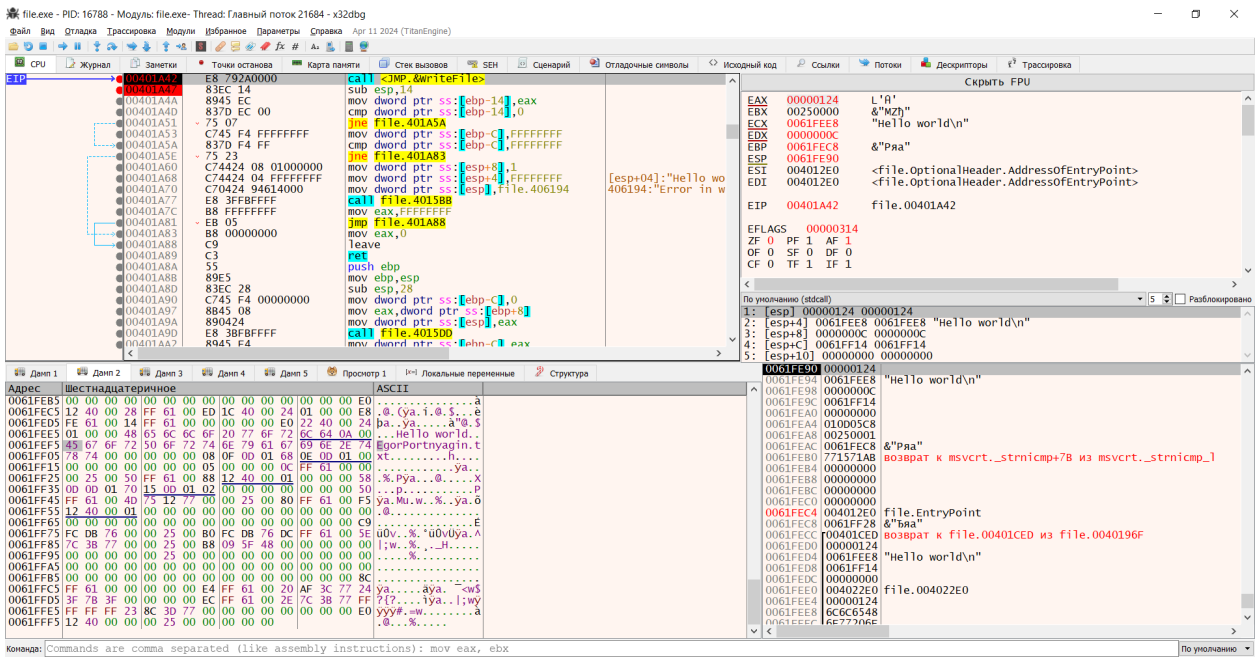

Рисунок 14 – до WriteFile

После WriteFile регистре EAX видим 00000001, значит данные успешно записались. (Рисунок 15)

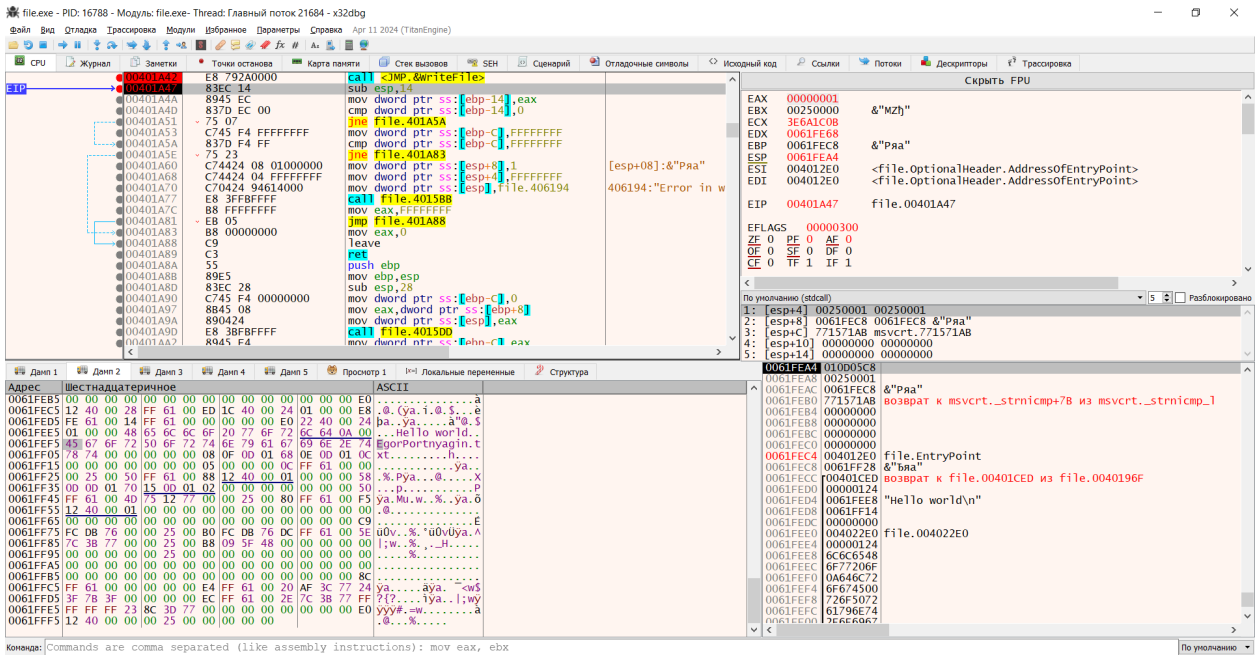

Рисунок 15 – после WriteFile

Операция записи в Process Monitor. (Рисунок 16)

Process Name	PID	Operation	Path	Result	Detail	Command Line
file.exe	16788	WriteFile	C:\Users\user\Desktop\hw8\EgorPortny...	SUCCESS	Offset: 0, Length: 12,...	"C:\Users\user\Desktop\hw8\file.exe"

Рисунок 16 - Операция записи в Process Monitor

Повторяю то же самое для ReadFile. В регистре EAX хранится хэндл файла, в данном случае это 00000124. Посмотрим на данные в буфере перед записью. (Рисунок 17)

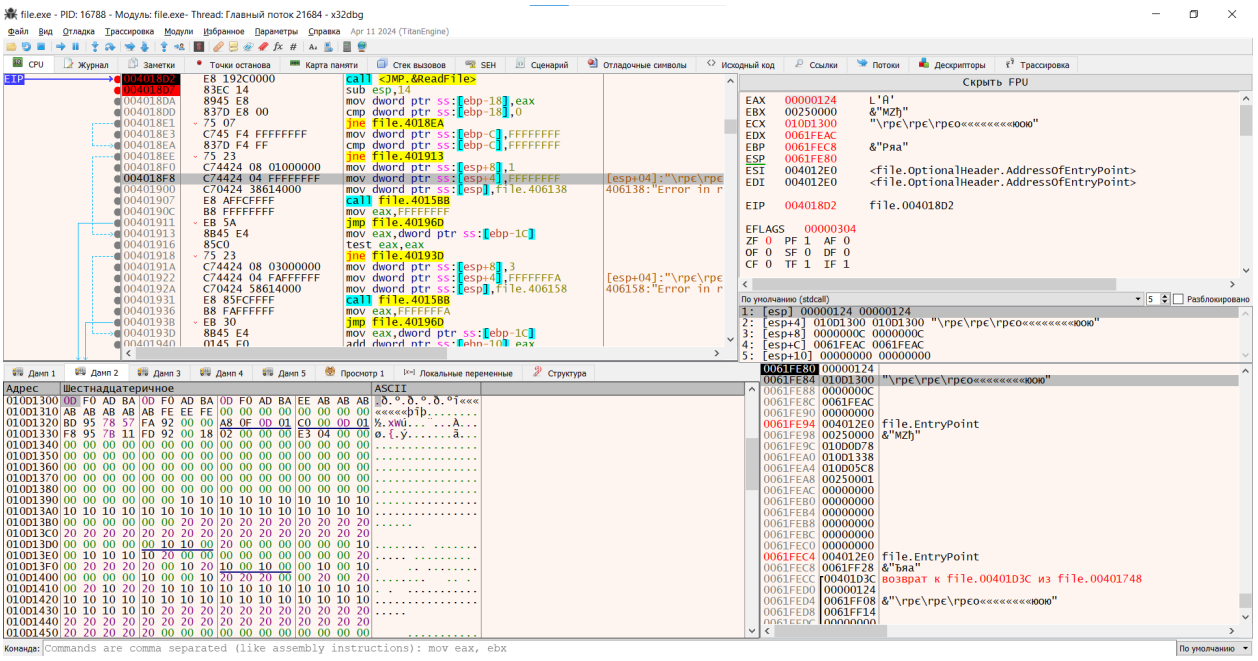

Рисунок 17 – память до ReadFile

После считывания данных, смотрю по адресу их появление. В EAX 00000001, значит данные успешно считались. (Рисунок 18)

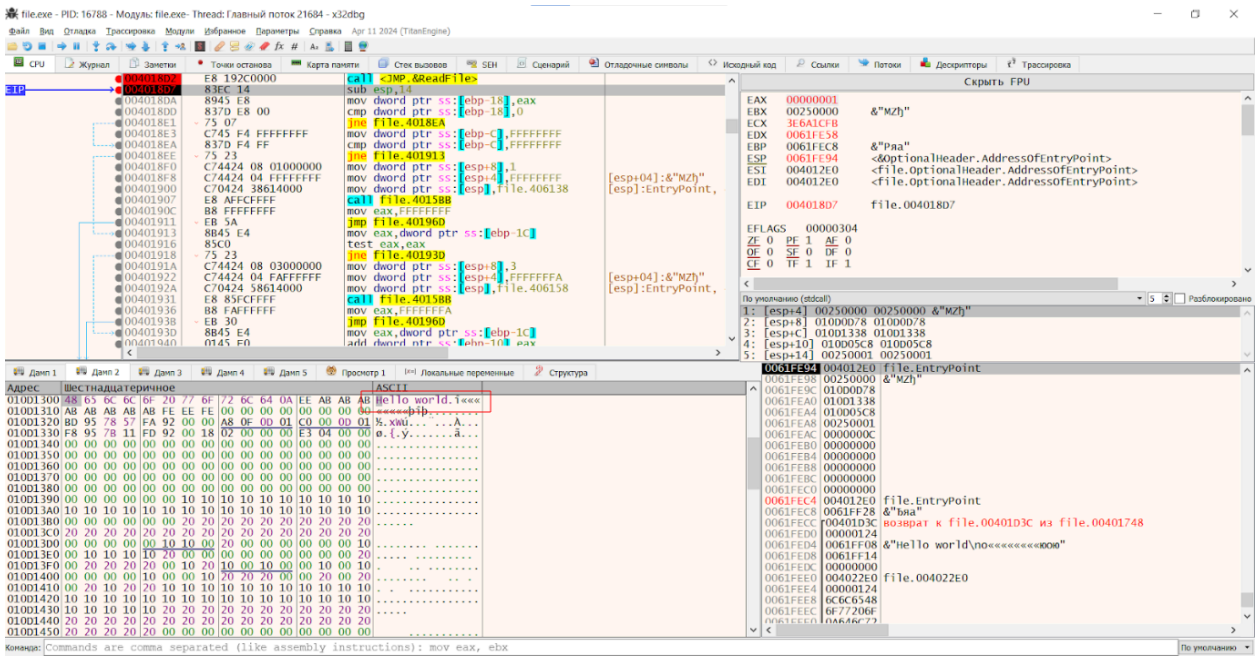

Рисунок 18 – память после считывания данных из файла

Операция чтения в Process Monitor. (Рисунок 19)

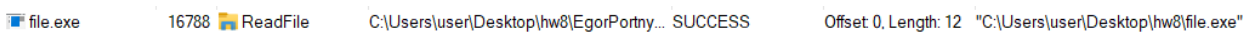

Рисунок 19 – операция чтения в Process Monitor

3 Выводы о проделанной работе

В процессе выполнения задания я получил знания о работе компьютерных систем и взаимодействии с ресурсами операционной системы Windows. Я изучил основные

способы работы с файлами, процессами и областями разделяемой памяти, используя язык программирования C и вызовы API ОС.

Особое внимание было уделено анализу и отладке программы с использованием отладчика на уровне ассемблера и инструментов Sysinternals.