

**Федеральное государственное автономное образовательное учреждение
высшего образования**

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Московский институт электроники и математики им. Тихонова

Департамент электронной инженерии

ОТЧЕТ

О ПРАКТИЧЕСКОЙ РАБОТЕ №3

по дисциплине «Системное программирование»

«Основы дизассемблирования»

Студент гр. БИБ201

Шадронов Алексей

Дата выполнения: 30 января 2023 г.

Преподаватель:

Смирнов Д. В.

«__» _____ 2023 г.

Москва, 2023

Содержание

1	Задание на практическую работу	3
2	Ход работы	3
2.1	Выделение памяти с помощью malloc в main	3
2.2	Обработка команды в fgets	5
2.3	Вызов API-функции CreateFile	5
2.4	Вызов API-функции WriteFile	7
2.5	Вызов API-функции ReadFile	8
2.6	Добавить аргументы к командам	9
2.7	Найти утечку ресурсов в программе	13
3	Выводы о проделанной работе	14
	Приложение А	15

1 Задание на практическую работу

Скомпилировать программу на языке C. Дизассемблировать полученный исполняемый файл с помощью x64dbg. Проследить за работой программы с помощью отладчика и Process Monitor и Process Hacker.

2 Ход работы

2.1 Выделение памяти с помощью malloc в main

Установим точку прерывания на месте последнего вызова функции перед завершением программы (cexit). Этот вызов соответствует функции main (Рисунок 1).

00D912C9	A1 14D0D900	mov eax,dword ptr ds:[D9D014]	
00D912CE	8B15 C8E1D900	mov edx,dword ptr ds:[<&__initenv>]	edx:EntryPoint
00D912D4	8902	mov dword ptr ds:[edx],eax	edx:EntryPoint
00D912D6	894424 08	mov dword ptr ss:[esp+8],eax	
00D912DA	A1 18D0D900	mov eax,dword ptr ds:[D9D018]	
00D912DF	894424 04	mov dword ptr ss:[esp+4],eax	
00D912E3	A1 1CD0D900	mov eax,dword ptr ds:[D9D01C]	
00D912E8	890424	mov dword ptr ss:[esp],eax	
00D912EB	E8 7A080000	call file.D91B6A	
00D912F0	8B0D 0CD0D900	mov ecx,dword ptr ds:[D9D00C]	ecx:EntryPoint
00D912F6	A3 10D0D900	mov dword ptr ds:[D9D010],eax	
00D912FB	85C9	test ecx,ecx	ecx:EntryPoint
00D912FD	0F84 CC000000	je file.D913CF	
00D91303	8B15 08D0D900	mov edx,dword ptr ds:[D9D008]	edx:EntryPoint
00D91309	85D2	test edx,edx	edx:EntryPoint
00D9130B	74 73	je file.D91380	
00D9130D	8D65 F0	lea esp,dword ptr ss:[ebp-10]	
00D91310	59	pop ecx	ecx:EntryPoint
00D91311	5B	pop ebx	
00D91312	5E	pop esi	esi:EntryPoint
00D91313	5F	pop edi	edi:EntryPoint
00D91314	5D	pop ebp	
00D91315	8D61 FC	lea esp,dword ptr ds:[ecx-4]	
00D91318	C3	ret	
00D91319	8DB426 00000000	lea esi,dword ptr ds:[esi]	esi:EntryPoint
00D91320	A1 44D0D900	mov eax,dword ptr ds:[D9D044]	
00D91325	BB 01000000	mov ebx,1	
00D9132A	83F8 01	cmp eax,1	
00D9132D	0F85 8FFEFFFF	jne file.D911C2	
00D91333	C70424 1F000000	mov dword ptr ss:[esp],1F	
00D9133A	E8 29750000	call <JMP.&_amsq_exit>	
00D9133F	A1 44D0D900	mov eax,dword ptr ds:[D9D044]	
00D91344	83F8 01	cmp eax,1	
00D91347	0F85 9AFEFFFF	jne file.D911E7	
00D9134D	C74424 04 08F0D900	mov dword ptr ss:[esp+4],file.D9F008	
00D91355	C70424 00F0D900	mov dword ptr ss:[esp],file.D9F000	
00D9135C	E8 1F750000	call <JMP.&_initterm>	
00D91361	C705 44D0D900 02000000	mov dword ptr ds:[D9D044],2	
00D9136B	85DB	test ebx,ebx	
00D9136D	0F85 7CFEFFFF	jne file.D911EF	
00D91373	871D 40D0D900	xchg dword ptr ds:[D9D040],ebx	
00D91379	E9 71FEFFFF	jmp file.D911EF	
00D9137E	66:90	nop	
00D91380	E8 EB740000	call <JMP.&_cexit>	
00D91385	A1 10D0D900	mov eax,dword ptr ds:[D9D010]	
00D9138A	8B65 F0	lea esp,dword ptr ss:[ebp-10]	

Рисунок 1 – Последний вызов перед выходом

После перехода по адресу функции мы видим два вызова (соответствуют функциям printf в исходном файле), инициализацию переменных и функции выделения памяти malloc (Рисунок 2). Выделяется память под переменную buf (куда попадают вводимые данные с консоли) и переменную command (первое слово из buf).

После вызова malloc в регистр EAX сохраняется адрес выделенной ячейки

00D91B6A	55	push ebp		
00D91B6B	89E5	mov ebp,esp		
00D91B6D	53	push ebx		
00D91B6E	83E4 F0	and esp,FFFFFFF0		
00D91B71	83EC 50	sub esp,50		
00D91B74	E8 47040000	call file.D91FC0		
00D91B79	C70424 79A1D900	mov dword ptr ss:[esp],file.D9A179	D9A179:"main\n"	
00D91B80	E8 98F9FFFF	call file.D91510		
00D91B85	C74424 40 00000000	mov dword ptr ss:[esp+40],0		
00D91B8D	C74424 4C 00000000	mov dword ptr ss:[esp+4C],0		
00D91B95	C74424 3C 00000000	mov dword ptr ss:[esp+3C],0		
00D91B9D	C74424 38 00000000	mov dword ptr ss:[esp+38],0	[esp+38]:EntryPoint	
00D91BA5	C74424 34 00000000	mov dword ptr ss:[esp+34],0	[esp+34]:EntryPoint	
00D91BAD	C74424 23 44616E69	mov dword ptr ss:[esp+23],696E6144		
00D91BB5	C74424 27 6C536D69	mov dword ptr ss:[esp+27],696D536C		
00D91BBD	C74424 2B 726E6F76	mov dword ptr ss:[esp+2B],766F6E72		
00D91BC5	C74424 2F 2E747874	mov dword ptr ss:[esp+2F],7478742E		
00D91BCD	C64424 33 00	mov byte ptr ss:[esp+33],0		
00D91BD2	C74424 16 48656C6C	mov dword ptr ss:[esp+16],6C6C6548		
00D91BDA	C74424 1A 6F20776F	mov dword ptr ss:[esp+1A],6F77206F		
00D91BE2	C74424 1E 726C640A	mov dword ptr ss:[esp+1E],A646C72		
00D91BEA	C64424 22 00	mov byte ptr ss:[esp+22],0		
00D91BEF	C74424 48 00000000	mov dword ptr ss:[esp+48],0		
00D91BF7	C74424 10 FFFFFFFF	mov dword ptr ss:[esp+10],FFFFFFF		
00D91BFF	C74424 44 00000000	mov dword ptr ss:[esp+44],0		
00D91C07	C70424 7FA1D900	mov dword ptr ss:[esp],file.D9A17F	D9A17F:"malloc\n"	
00D91C0E	E8 0AF9FFFF	call file.D91510		
00D91C13	C70424 81000000	mov dword ptr ss:[esp],81		
00D91C1A	E8 C96C0000	call <JMP.&malloc>		
00D91C1F	894424 3C	mov dword ptr ss:[esp+3C],eax		
00D91C23	C70424 81000000	mov dword ptr ss:[esp],81		
00D91C2A	E8 B96C0000	call <JMP.&malloc>		
00D91C2F	894424 38	mov dword ptr ss:[esp+38],eax	[esp+38]:EntryPoint	
00D91C33	C70424 87A1D900	mov dword ptr ss:[esp],file.D9A187	D9A187:"waiting for user input...\n"	
00D91C3A	E8 DEF8FFFF	call file.D91510		
00D91C3F	C70424 00000000	mov dword ptr ss:[esp],0		
00D91C46	A1 4890D900	mov eax,dword ptr ds:[D99048]		
00D91C48	FFD0	call eax		
00D91C4D	8B5424 3C	mov edx,dword ptr ss:[esp+3C]		
00D91C51	894424 08	mov dword ptr ss:[esp+8],eax		
00D91C55	C74424 04 7E000000	mov dword ptr ss:[esp+4],7E	7E:'~'	
00D91C5D	891424	mov dword ptr ss:[esp],edx		
00D91C60	E8 536C0000	call <JMP.&fgets>		
00D91C65	8B4424 3C	mov eax,dword ptr ss:[esp+3C]		
00D91C69	894424 04	mov dword ptr ss:[esp+4],eax		
00D91C6D	C70424 A2A1D900	mov dword ptr ss:[esp],file.D9A1A2	D9A1A2:"buf is %s\n"	
00D91C74	E8 A4F8FFFF	call file.7C1510		
00D91C79	C74424 48 00000000	mov dword ptr ss:[esp+48],0		
00D91C81	EB 05	jmp file.7C1C88		
00D91C83	834424 48 01	add dword ptr ss:[esp+48],1		
00D91C88	8B4424 3C	mov eax,dword ptr ss:[esp+3C]		
00D91C8C	890424	mov dword ptr ss:[esp],eax		
00D91C8F	E8 7C6C0000	call <JMP.&strlen>		
00D91C94	394424 48	cmp dword ptr ss:[esp+48],eax		
00D91C98	73 33	jae file.7C1CDD		
00D91C9A	8B5424 3C	mov edx,dword ptr ss:[esp+3C]		

Рисунок 2 – Выделение памяти с помощью malloc

памяти. На рисунке 3 видно, что этот участок памяти заполнен случайными значениями.

EIP

007C1C1A

007C1C1F

007C1C23

007C1C2A

007C1C2F

007C1C33

007C1C3A

007C1C3F

007C1C46

007C1C48

007C1C4D

007C1C51

007C1C55

007C1C5D

007C1C60

007C1C65

007C1C69

007C1C6D

007C1C74

007C1C79

007C1C81

007C1C83

007C1C88

007C1C8C

007C1C8F

007C1C94

007C1C98

007C1C9A

E8 C96C0000

894424 3C

C70424 81000000

E8 B96C0000

894424 38

C70424 87A17C00

E8 DEF8FFFF

C70424 00000000

A1 48907C00

FFD0

8B5424 3C

894424 08

C74424 04 7E000000

891424

E8 536C0000

8B4424 3C

894424 04

C70424 A2A17C00

E8 A4F8FFFF

C74424 48 00000000

EB 05

834424 48 01

8B4424 3C

890424

E8 7C6C0000

394424 48

73 33

8B5424 3C

call <JMP.&malloc>

mov dword ptr ss:[esp+3C],eax

mov dword ptr ss:[esp],81

call <JMP.&malloc>

mov dword ptr ss:[esp+38],eax

mov dword ptr ss:[esp],file.7CA187

call file.7C1510

mov dword ptr ss:[esp],0

mov eax,dword ptr ds:[7C9048]

call eax

mov edx,dword ptr ss:[esp+3C]

mov dword ptr ss:[esp+8],eax

mov dword ptr ss:[esp+4],7E

mov dword ptr ss:[esp],edx

call <JMP.&fgets>

mov eax,dword ptr ss:[esp+3C]

mov dword ptr ss:[esp+4],eax

mov dword ptr ss:[esp],file.7CA1A2

call file.7C1510

mov dword ptr ss:[esp+48],0

jmp file.7C1C88

add dword ptr ss:[esp+48],1

mov eax,dword ptr ss:[esp+3C]

mov dword ptr ss:[esp],eax

call <JMP.&strlen>

cmp dword ptr ss:[esp+48],eax

jae file.7C1CDD

mov edx,dword ptr ss:[esp+3C]

dword ptr ss:[esp+3C]=[00FFFB5C]=0

eax=01890E20

.text:007C1C1F file.exe:\$1C1F #101F

Dump 1

Dump 2

Dump 3

Dump 4

Dump 5

Watch 1

[x=]

Lo

Address	Hex	ASCII
01890E20	0D F0 AD BA	0D F0 AD BA
01890E30	0D F0 AD BA	0D F0 AD BA
01890E40	0D F0 AD BA	0D F0 AD BA
01890E50	0D F0 AD BA	0D F0 AD BA
01890E60	0D F0 AD BA	0D F0 AD BA
01890E70	0D F0 AD BA	0D F0 AD BA

EAX

EBX

ECX

EDX

EBP

ESP

ESI

EDI

EIP

EFLAGS

ZF 0 PF 0 AF 0

OF 0 SF 0 DF 0

CF 0 TF 1 IF 1

LastError 0000007

LastStatus C000013

CS 002B ES 0053

Default (stdcall)

1: [esp+4] 00000000

2: [esp+8] 00FFFC2

3: [esp+C] 753FCC0

4: [esp+10] FFFFFFF

5: [esp+14] 6548FF

00FFFB20

00000081

00FFFB24

00000002

00FFFB28

00FFFC24

00FFFB2C

753FCC00

00FFFB30

FFFFFFFF

00FFFB34

6548FFFE

00FFFB38

206F6C6C

00FFFB3C

6C726F77

00FFFB40

44000A64

00FFFB44

6C696E61

Рисунок 3 – Memory dump

2.2 Обработка команды в fgets

В консоль введём слово open для открытия файла. На рисунке 4 видно, что введённое значение было записано в ячейку памяти.

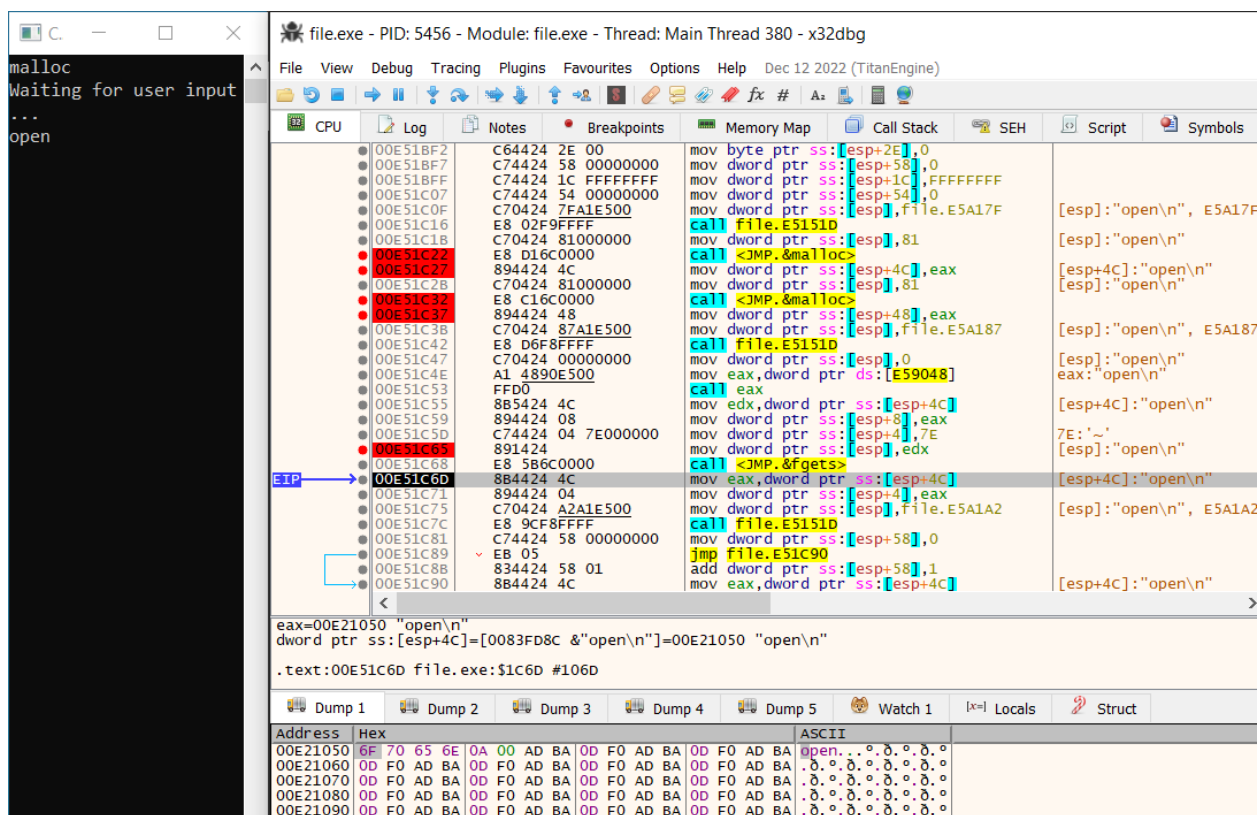


Рисунок 4 – Содержимое памяти

2.3 Вызов API-функции CreateFile

Далее вызывается функция CreateFileA. На рисунке 5 приведён нужный фрагмент библиотеки kernel32.dll.

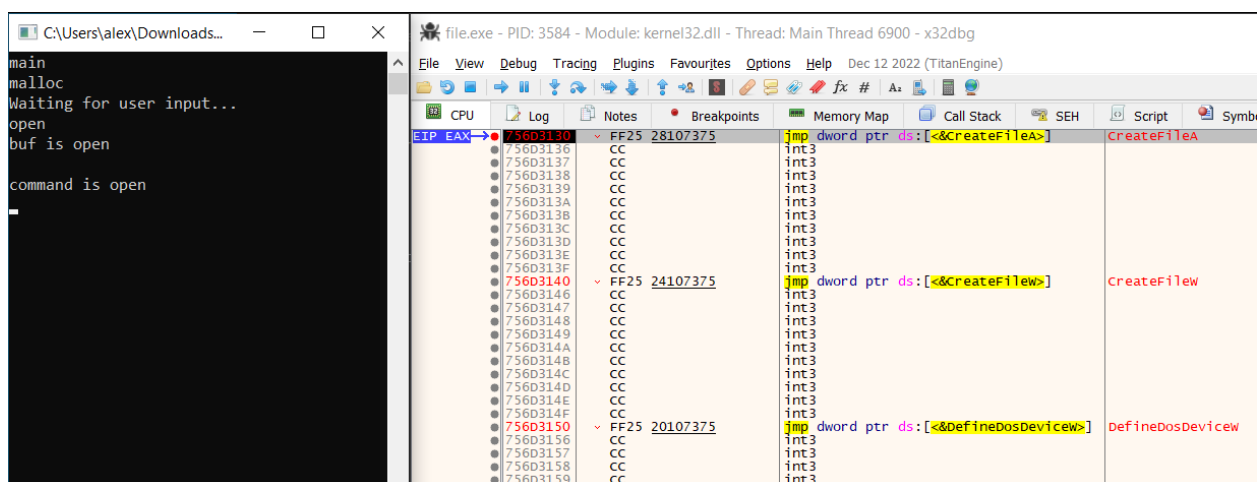


Рисунок 5 – Остановка на функции CreateFileA

На рисунке 6 видно, что в стеке сохранены аргументы функции, например,

название файла (AlekseyShadrinov.txt).

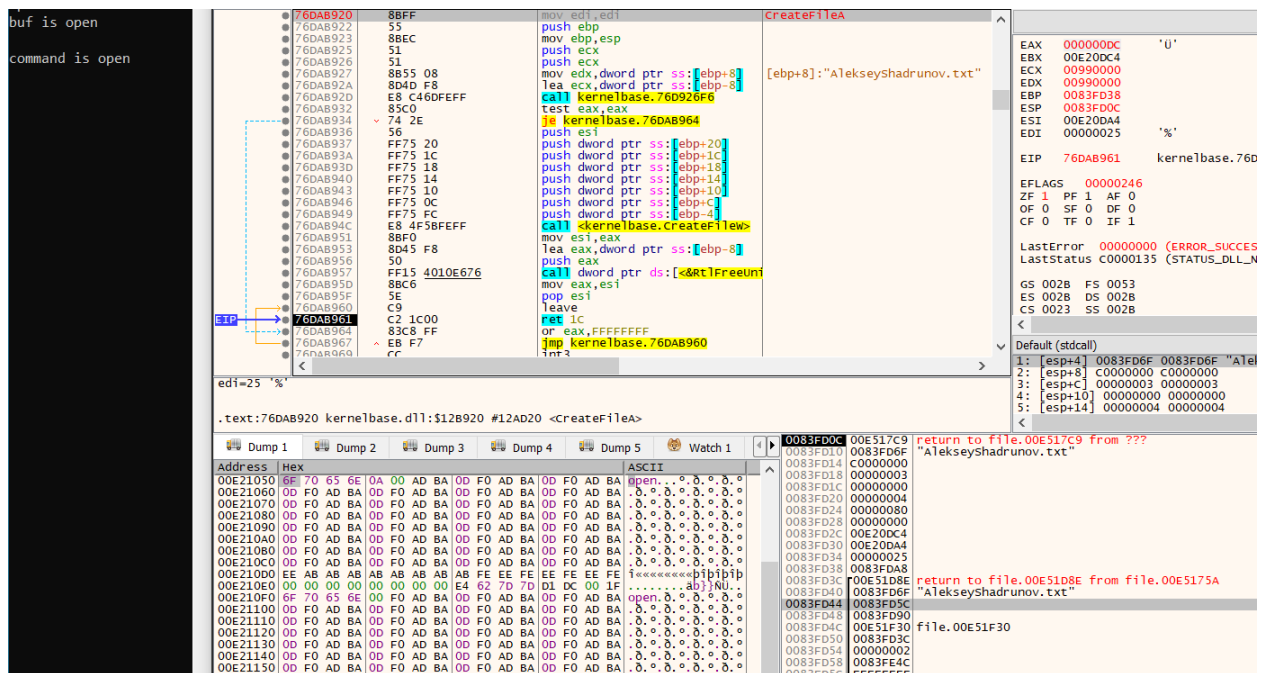


Рисунок 6 – Стек

После вызова API-функции CreateFileA можно увидеть, что в EAX записано значение 0xE0. Это файловый дескриптор или хэнгл открытого файла (Рисунок 7).

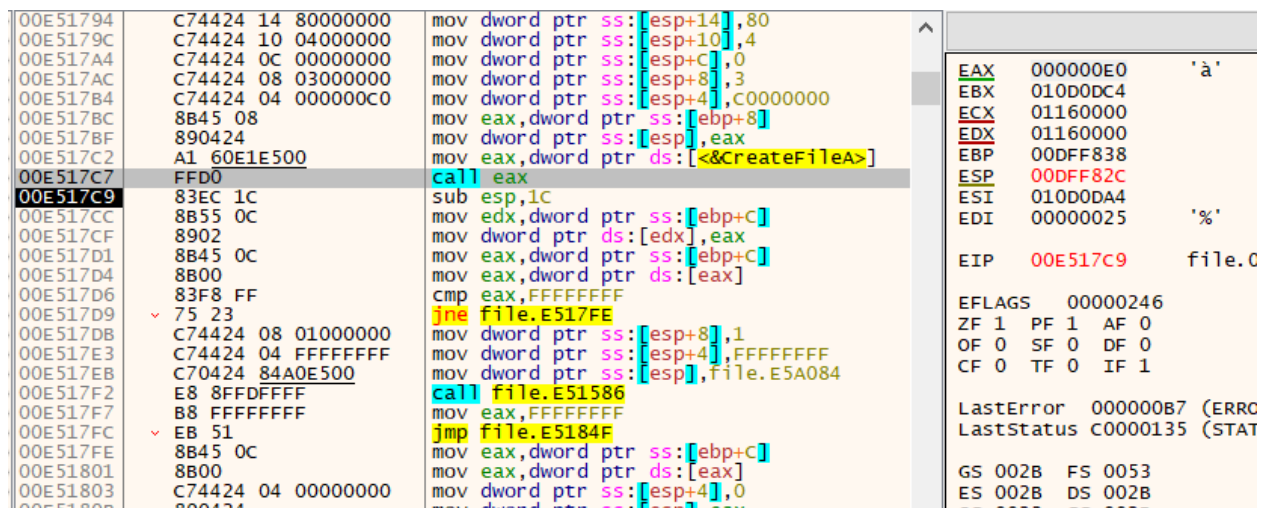


Рисунок 7 – Хэнгл открытого файла

Этот же хэнгл можно пронаблюдать в Process Hacker (Рисунок 8).

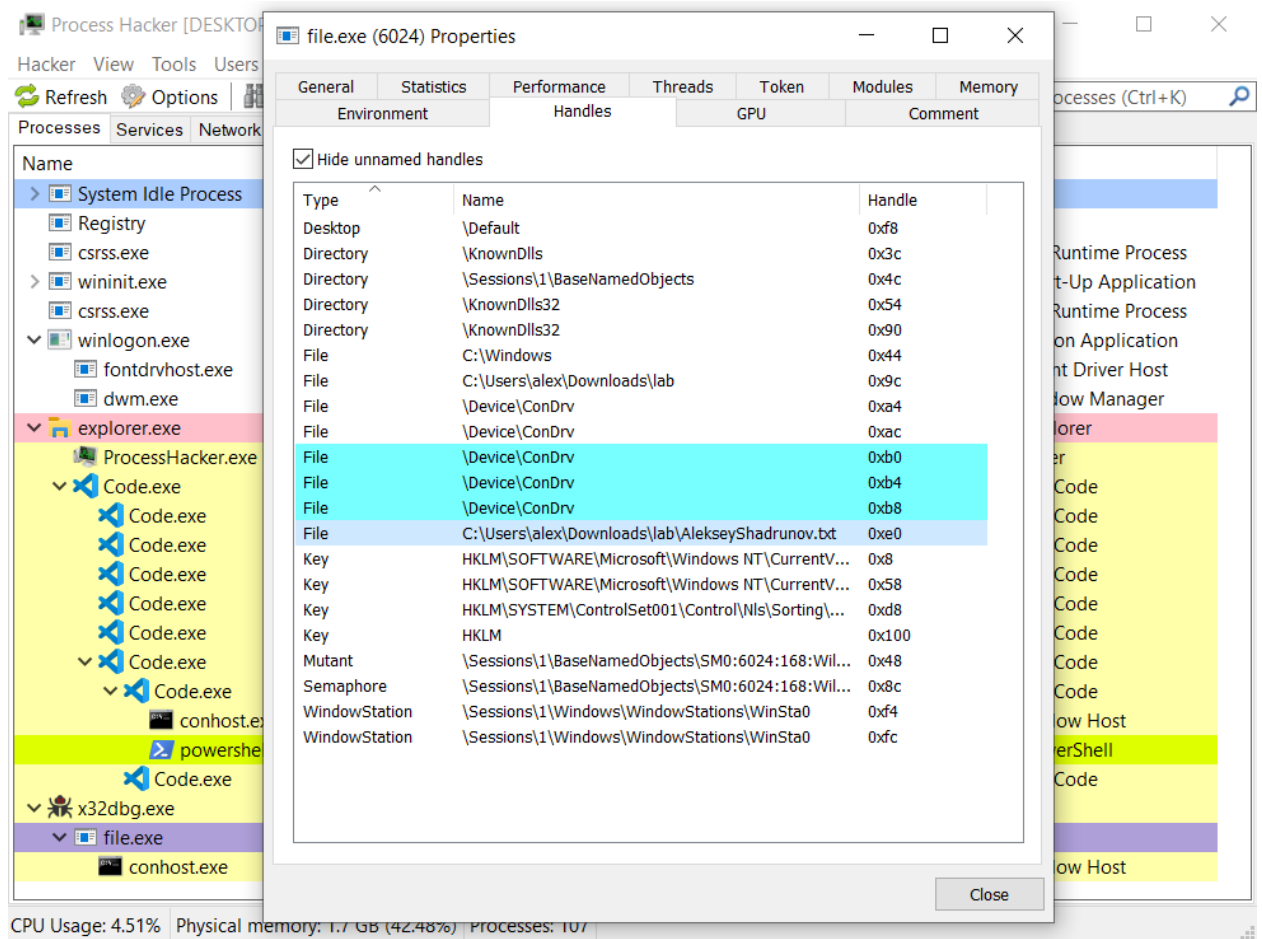


Рисунок 8 – Хэнды в Process Hacker

2.4 Вызов API-функции WriteFile

Вводим в консоль команду `write` для записи строки в файл. На рисунке 9 показано состояние системы перед вызовом функции `WriteFile`. Все аргументы записаны в стек: хэнды файла (104), буфер со строкой, количество байт для записи (0xC, то есть 13) и ещё два аргумента.

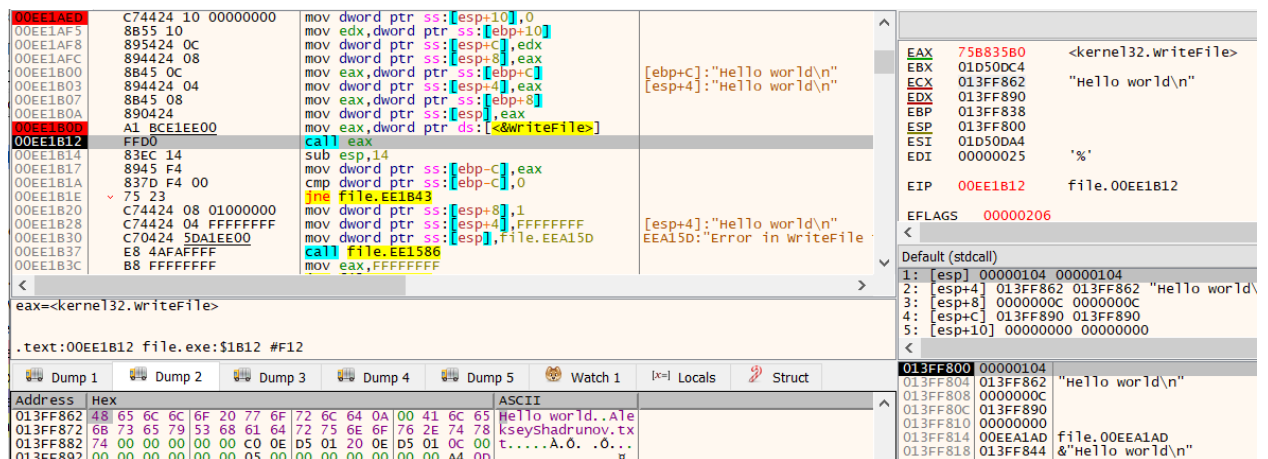


Рисунок 9 – Аргументы в стеке

На рисунке 10 видно тело функции WriteFile, а также строка для записи, которая находится в дампе памяти. На рисунке 11 — возвращаемое значение 1 (в регистре EAX), что соответствует успешному выполнению функции.

766507CF	CC	int3		
766507D0	6A 18	push 18	writeFile	
766507D2	68 20A67076	push kernelbase.7670A620		
766507D7	E8 E48F0200	call kernelbase.766797C0		
766507DC	33C9	xor ecx,ecx	ecx:"Hello world\n"	
766507DE	894D E0	mov dword ptr ss:[ebp-20],ecx	[ebp-20]:&"Hello world\n"	
766507E1	894D E4	mov dword ptr ss:[ebp-1C],ecx		
766507E4	8B75 14	mov esi,dword ptr ss:[ebp+14]		
766507E7	85F6	test esi,esi		
766507E9	74 02	je kernelbase.766507ED	ecx:"Hello world\n"	
766507EB	890E	mov dword ptr ds:[esi],ecx		
766507ED	8B7D 08	mov edi,dword ptr ss:[ebp+8]		
766507F0	83FF F4	cmp edi,FFFFFFFF		
766507F3	0F84 0F540300	je kernelbase.76685C08		
766507F9	83FF F5	cmp edi,FFFFFFF5		
766507FC	0F84 F5530300	je kernelbase.76685BF7		
76650802	83FF F6	cmp edi,FFFFFFF6		
76650805	0F84 0B530300	je kernelbase.76685BE6		
7665080B	8B5D 18	mov ebx,dword ptr ss:[ebp+18]		

EAX	75B835B0	<kernel32.writeFile>
EBX	00F40DC4	
ECX	013FF702	"Hello world\n"
EDX	013FF730	
EBP	013FF6D8	
ESP	013FF698	
ESI	00F40DA4	'%'
EDI	00000025	
EIP	766507D2	kernelbase.766507D2
EFLAGS	00000206	

1:	[esp+4]	00EE1B14	file.00EE1B14
2:	[esp+8]	00000108	00000108
3:	[esp+C]	013FF702	013FF702 "Hello world\n"
4:	[esp+10]	0000000C	0000000C
5:	[esp+14]	013FF730	013FF730

Address	Hex	ASCII
013FF702	48 65 6C 6C 6F 20 77 6F 72 6C 64 0A 00 41 6C 65	Hello world..Ale
013FF712	68 73 65 79 53 68 61 64 72 75 6E 6F 76 2E 74 78	kseyshadrnunov.tx
013FF722	74 00 00 00 00 00 C0 0E F4 00 20 0E F4 00 0C 00	t....A.o..o...
013FF732	00 00 00 00 00 00 05 00 00 00 00 00 00 00 A4 0DM.
013FF742	F4 00 C4 0D F4 00 88 F7 3F 01 F0 12 EE 00 01 00	o.A.o..p.d.i...

Рисунок 10 – Дамп памяти со строкой

00EE1AF5	8B55 10	mov edx,dword ptr ss:[ebp+10]		
00EE1AF8	895424 0C	mov dword ptr ss:[esp+C],edx		
00EE1AFC	894424 08	mov dword ptr ss:[esp+8],eax		
00EE1B00	8B45 0C	mov eax,dword ptr ss:[ebp+C]		
00EE1B03	894424 04	mov dword ptr ss:[esp+4],eax		
00EE1B07	8B45 08	mov eax,dword ptr ss:[ebp+8]		
00EE1B0A	890424	mov dword ptr ss:[esp],eax		
00EE1B0D	A1 BCE1EE00	mov eax,dword ptr ds:[<writeFile>]		
00EE1B12	FFD0	call eax		
00EE1B14	83EC 14	sub esp,14		
00EE1B17	8945 F4	mov dword ptr ss:[ebp-C],eax		
00EE1B1A	837D F4 00	cmp dword ptr ss:[ebp-C],0		
00EE1B1E	75 23	jne file.EE1B43		
00EE1B20	C74424 08 01000000	mov dword ptr ss:[esp+8],1		
00EE1B28	C74424 04 FFFFFFFF	mov dword ptr ss:[esp+4],FFFFFFFF		
00EE1B30	C70424 5DA1EE00	mov dword ptr ss:[esp],file.EEA15D		
00EE1B37	E8 4AFAFFFF	call file.EE1586		
00EE1B3C	B8 FFFFFFFF	mov eax,FFFFFFFF		
00EE1B41	EB 05	jmp file.EE1B48		

EAX	00000001
EBX	00BF0DC4
ECX	7ED967D5
EDX	009FF928
EBP	009FF988
ESP	009FF964
ESI	00BF0DA4
EDI	00000025
EIP	00EE1B14
EFLAGS	00000200

1:	[esp+4]	009FF964
----	---------	----------

Рисунок 11 – Возвращаемое значение 1

2.5 Вызов API-функции ReadFile

Далее считываем из файла с помощью функции ReadFile. Аргументы функции: хэндл (0x100), адрес выходного буфера (0xDC0F60), количество байт для чтения (0xC) и ещё два опциональных аргумента (Рисунок 12). Аргументы записываются в стек перед вызовом функции.

После вызова функции по адресу выходного буфера содержится строка из файла (Рисунок 13)

00EE1995	8B55 0C	mov	edx,dword ptr	ss:[ebp-c]	
00EE1998	8B0A	mov	ecx,dword ptr	ds:[edx]	
00EE199A	8B55 F4	mov	edx,dword ptr	ss:[ebp-c]	
00EE199D	01D1	add	ecx,edx		
00EE199F	C74424 10 00000000	mov	dword ptr	ss:[esp+10],0	
00EE19A7	8D55 E8	lea	edx,dword ptr	ss:[ebp-18]	
00EE19AA	895424 0C	mov	dword ptr	ss:[esp+c],edx	
00EE19AE	894424 08	mov	dword ptr	ss:[esp+8],eax	
00EE19B2	894C24 04	mov	dword ptr	ss:[esp+4],ecx	
00EE19B6	8B45 08	mov	eax,dword ptr	ss:[ebp+8]	
00EE19B9	890424	mov	dword ptr	ss:[esp],eax	
00EE19BC	A1 9CE1EE00	mov	eax,dword ptr	ds:[<ReadFile>]	
00EE19C1	FFD0	call	eax		
00EE19C3	83EC 14	sub	esp,14		
00EE19C6	8945 F0	mov	dword ptr	ss:[ebp-10],eax	
00EE19C9	837D F0 00	cmp	dword ptr	ss:[ebp-10],0	
00EE19CD	75 23	jne	file.EE19F2		
00EE19CF	C74424 08 01000000	mov	dword ptr	ss:[esp+8],1	
00EE19D7	C74424 04 FFFFFFFF	mov	dword ptr	ss:[esp+4],FFFFFFFF	

<

eax=<kernel32.ReadFile>

.text:00EE19C1 file.exe:\$19C1 #DC1

Dump 1	Dump 2	Dump 3	Dump 4	Dump 5	Watch 1	[x=] Locals	Struct
Address	Hex					ASCII	
77861000	16 00 18 00 88 83 86 77	14	00	16	00	00 7C 86 77w..... .w
77861010	00 00 02 00 FC 5D 86 77	0E	00	10	00	80 85 86 77u].w......w
77861020	00 00 00 00 00 00 00 00	0A	00	00	00	00 00 00 00p.w......w

EAX

75B834C0

<kernel32.ReadFile>

EBX

009C0DC4

ECX

009C0F60

EDX

00DFFC10

EBP

00DFFC28

ESP

00DFFBF0

ESI

009C0DA4

EDI

00000025

'%

EIP

00EE19C1

file.00EE19C1

EFLAGS

00000206

<

Default (stdcall)

1:

[esp]

00000100

00000100

2:

[esp+4]

009C0F60

009C0F60

3:

[esp+8]

0000000C

0000000C

4:

[esp+c]

00DFFC10

00DFFC10

5:

[esp+10]

00000000

00000000

<

00DFFBF0

00000100

00DFFBF4

009C0F60

00DFFBF8

0000000C

00DFFBFC

00DFFC10

00DFFC00

00000000

00DFFC04

00EEA1AD

file.00EEA1AD

Рисунок 12 – Аргументы в стеке

00EE1E0C	85C0	test eax,eax	eax:"Hello world\n"
00EE1E0E	75 45	jne file.EE1E55	
00EE1E10	8B4424 1C	mov eax,dword ptr ss:[esp+1C]	
00EE1E14	8D5424 50	lea edx,dword ptr ss:[esp+50]	
00EE1E18	895424 08	mov dword ptr ss:[esp+8],edx	
00EE1E1C	8D5424 44	lea edx,dword ptr ss:[esp+44]	[esp+44]:"Hello world\n"
00EE1E20	895424 04	mov dword ptr ss:[esp+4],edx	[esp+4]:&"Hello world\n"
00EE1E24	890424	mov dword ptr ss:[esp],eax	
00EE1E27	E8 25FAFFFF	call file.EE1E81	
00EE1E2C	894424 5C	mov dword ptr ss:[esp+5C],eax	
00EE1E30	8B4424 44	mov eax,dword ptr ss:[esp+44]	[esp+44]:"Hello world\n"
00EE1E34	894424 04	mov dword ptr ss:[esp+4],eax	[esp+4]:&"Hello world\n"
00EE1E38	C70424 E1A1EE00	mov dword ptr ss:[esp],file.EEA1E1	EEA1E1:content of the file
00EE1E3F	E8 D9F6FFFF	call file.EE1E1D	
00EE1E44	8D4424 44	lea eax,dword ptr ss:[esp+44]	[esp+44]:"Hello world\n"
00EE1E48	890424	mov dword ptr ss:[esp],eax	
00EE1E4B	E8 E2F8FFFF	call file.EE1E73	
00EE1E50	E9 E6FDFFFF	jmp file.EE1E38	
00EE1E55	8B4424 48	mov eax,dword ptr ss:[esp+48]	[esp+48]:"read"

dword ptr ss:[esp+4]=[00DFFC34]=00DFFC74 &"Hello world\n"

eax=009C0F60 "Hello world\n"

.text:00EE1E34 file.exe:\$1E34 #1234

Dump 1	Dump 2	Dump 3	Dump 4	Dump 5	Watch 1	Locals	Struct
Address	Hex					ASCII	
009C0F60	48 65 6C 6C 6F 20 77 6F 72 6C 64 0A 00 AB AB AB					Hello world...<<<	
009C0F70	AB AB AB AB AB FE EE FE 00 00 00 00 00 00 00					<<<<<bip.....	
009C0F80	81 0B 53 44 5D 89 00 1C 00 00 00 00 00 00 00					..SD]'.<<<<<<<	
009C0F90	00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00				<<<<<<<	

EAX	009C0F60	"Hello world\n"
EBX	009C0DC4	
ECX	F98B8DB2	
EDX	009C0F60	"Hello world\n"
EBP	00DFFC98	
ESP	00DFFC30	
ESI	009C0DA4	'%'
EDI	00000025	
EIP	00EE1E34	file.00EE1E34
EFLAGS	00000206	

Default (stdcall)

1: [esp+4] 00DFFC74 00DFFC74 &"Hello world\n"

2: [esp+8] 00DFFC80 00DFFC80

3: [esp+C] 00EE1F30 file.00EE1F30

4: [esp+10] 00DFFC2C 00DFFC2C

5: [esp+14] 00000002 00000002

00DFFC30	00000100	
00DFFC34	00DFFC74	&"Hello world\n"
00DFFC38	00DFFC80	
00DFFC3C	00EE1F30	file.00EE1F30
00DFFC40	00DFFC2C	
00DFFC44	00000002	
00DFFC48	00DFFD3C	

Рисунок 13 – Строка в памяти

2.6 Добавить аргументы к командам

С помощью ввода в консоль реализовано использование аргументов для задания параметров программы. Для открытия файлов можно задать имя файла, для записи можно задать хэндл и записываемое значение, для чтения можно задать хэндл, для закрытия — тоже хэндл. Для удобства работы пользователя хэндл выводится и вводится в виде десятичного числа. Примеры работы программы представлены на рисунках 14-18. Листинг кода содержится в приложении А.

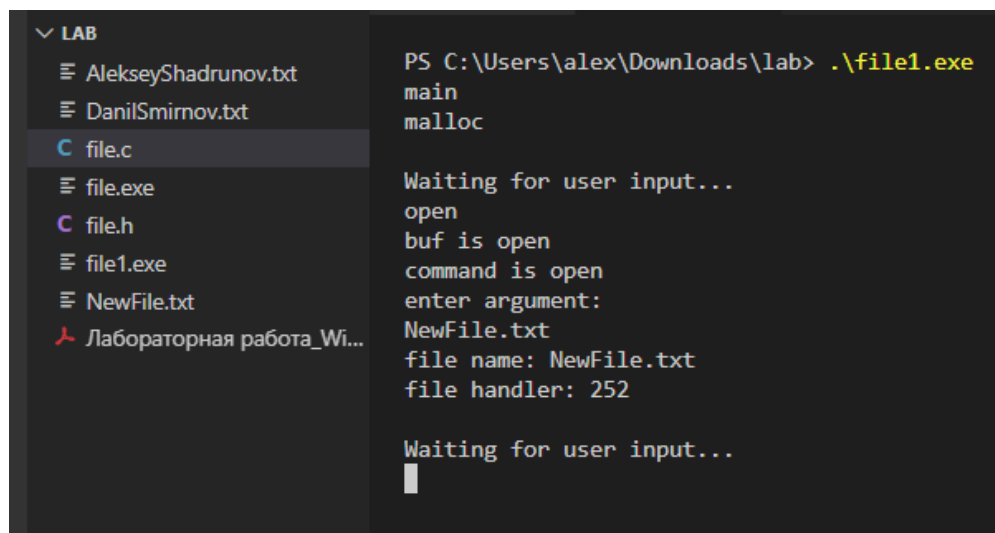


Рисунок 14 – Открытие файла: вводим название файла, получаем хэндл в виде числа (252)

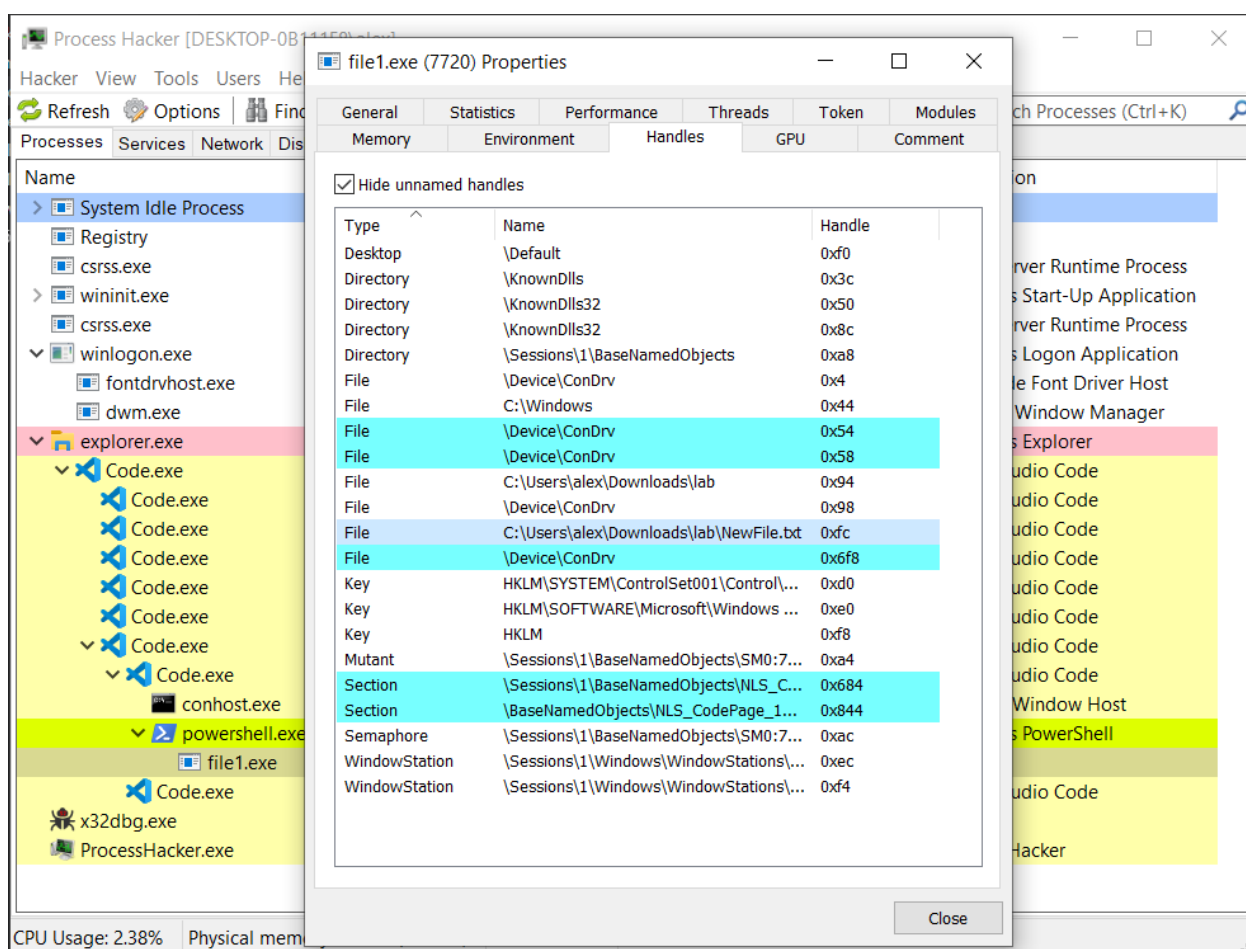


Рисунок 15 – Этот хэндл в Process Hacker (0xFC = 252)

```
Waiting for user input...
write
buf is write
command is write
enter handle argument:
252
pointer: 252
enter file value:
This string is a perfect example for my file!
file value: This string is a perfect example for my file!

Waiting for user input...
█
```

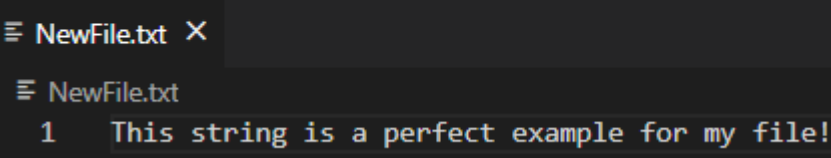


Рисунок 16 – Команда запись: вводим хэндл файла, значение для записи

```
Waiting for user input...
read
buf is read
command is read
enter handle argument:
252
pointer: 252
Content of the file is:
This string is a perfect example for my file!

Waiting for user input...
█
```

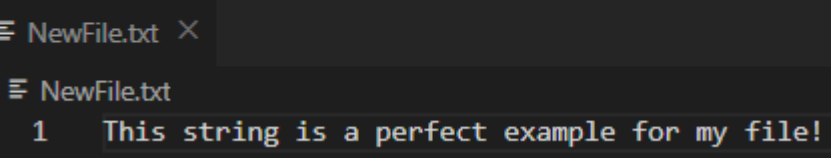


Рисунок 17 – Команда чтение: вводим хэндл файла, программа выводит содержимое

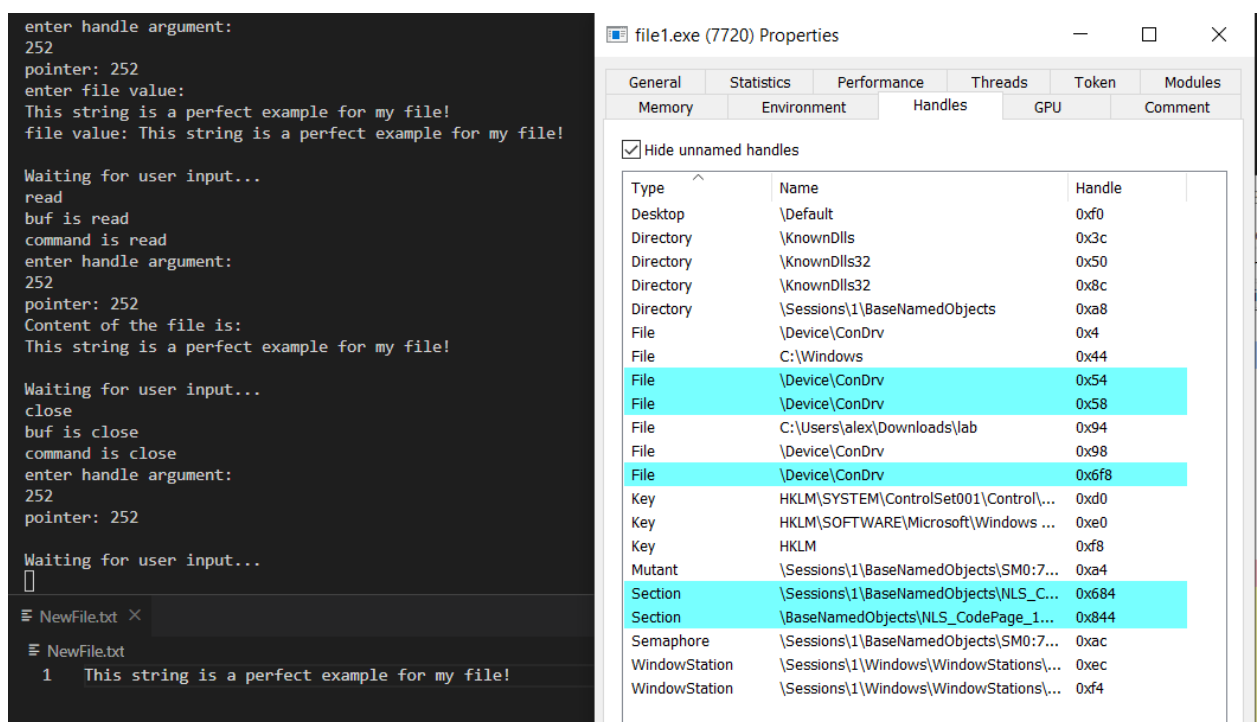


Рисунок 18 – Команда закрыть: видно, что хэндл исчез из Process Hacker

2.7 Найти утечку ресурсов в программе

Утечка памяти в исходном файле связана с тем, как обрабатываются хэндлеры открытого файла. Видно, что для хранения хэндла используется переменная `HANDLE hFile`, которая создаётся при запуске программы. Далее при вводе команды открыть в эту переменную присваивается хэндл на открытый файл. Если ввести команду открыть ещё раз, то предыдущее значение хэндла будет перезаписано. Это значит, что закрыть предыдущий хэндл и освободить память невозможно.

Есть несколько путей исправления этой утечки: можно создать массив хэндлов и записывать каждый следующий файл в массив, можно закрывать старый файл перед открытием нового файла. Наконец, можно запрещать открытие файла, если хэндл уже определён (и файл уже открыт). Реализация этого способа приведена на листинге 1. Результат работы программы — на рисунке 19.

```
else if ( strncasecmp(command, "open", strlen(command)) == 0 )
{
    if ( hFile == INVALID_HANDLE_VALUE )
    {
        ret = LabOpenFile(chFileName, &hFile, &len);
    }
    else
    {
        printf("handle already defined. can't open file again! \n");
    }
}
```

Листинг 1 – Фрагмент функции `main`

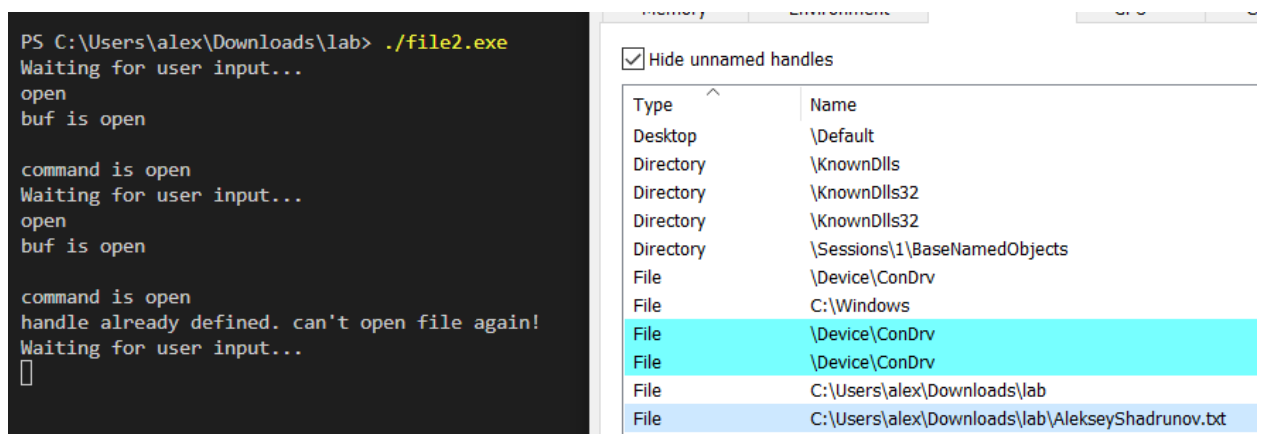


Рисунок 19 – Повторная команда открыть не работает

3 Выводы о проделанной работе

В рамках данной работы получены знания: о принципах работы ЭВМ, об особенностях архитектуры компьютера фон Неймана и x86/x64 и о работе с объектами в ОС Windows. В рамках данной работы получены навыки: анализ программ с использованием отладчика уровня ассемблера и отладка и поиск проблем с использованием средств Sysinternals.

Приложение А

```
1 // Подключить нашу библиотеку
2 #include "file.h"
3
4 void ErrorOutput(const char *chFuncFailMessage, int iErr, int iFlag)
5 {
6     DWORD n = 0; // Код ошибки
7     const DWORD size = LAB_MAX_BUF_SIZE; // Максимальный размер сообщения об
        ошибке
8     char buf[LAB_MAX_BUF_SIZE]; // Буфер сообщения
9
10    switch ( iFlag ) // Обработка флага
11    {
12        case LAB_FLAG_GET:
13            n = GetLastError();
14            FormatMessageA(FORMAT_MESSAGE_FROM_SYSTEM,
15                        NULL,
16                        n,
17                        MAKELANGID(LANG_ENGLISH, SUBLANG_DEFAULT),
18                        buf,
19                        size - 1,
20                        NULL);
21            break;
22
23        case LAB_FLAG_ERRNO:
24            n = errno;
25            FormatMessageA(FORMAT_MESSAGE_FROM_SYSTEM,
26                        NULL,
27                        n,
28                        MAKELANGID(LANG_ENGLISH, SUBLANG_DEFAULT),
29                        buf,
30                        size - 1,
31                        NULL);
32            break;
33
34        case LAB_FLAG_RESULT:
35            FormatMessageA(FORMAT_MESSAGE_FROM_SYSTEM,
36                        NULL,
37                        iErr,
38                        MAKELANGID(LANG_ENGLISH, SUBLANG_DEFAULT),
```

```

39             buf,
40             size - 1,
41             NULL);
42         break;
43
44     default :
45         sprintf(buf, "Error in programmer's function code is %d", iErr);
46         break;
47     }
48
49     MessageBoxA(0, buf, chFuncFailMessage, MB_ICONERROR | MB_OK); // Вывод
сообщения об ошибке
50 }
51
52 //-----
53
54 int ReleaseHandle(HANDLE *h)
55 {
56     int bRet = 0;
57
58     // Если хендл валидный
59     if ( *h != INVALID_HANDLE_VALUE )
60     {
61         bRet = CloseHandle(*h); // Закрыть
62         *h = INVALID_HANDLE_VALUE; // Сделать невалидным
63     }
64
65     return bRet;
66 }
67
68 //-----
69
70 void ReleaseVoidMemory(void **vMemory)
71 {
72     // Если указатель на область памяти валидный
73     if ( *vMemory != NULL )
74     {
75         free(*vMemory); // Освободить
76         *vMemory = NULL; // Сделать невалидным
77     }

```

```

78 }
79
80 //-----
81
82 int LabOpenFile(const char *chFileName, HANDLE *hFile, DWORD *dwFileSize)
83 {
84     if ( chFileName == NULL ) // Если плохой буфер
85     {
86         ErrorOutput("Error in input buffer", LAB_BAD_BUF, LAB_FLAG_MY); //
Сообщение
87         return LAB_BAD_BUF; // Вернуть ошибку
88     }
89
90
91     // Открыть файл на чтение и запись
92     *hFile = CreateFileA(chFileName, GENERIC_READ | GENERIC_WRITE,
FILE_SHARE_READ | FILE_SHARE_WRITE, NULL, OPEN_ALWAYS,
FILE_ATTRIBUTE_NORMAL, NULL);
93
94     if ( *hFile == INVALID_HANDLE_VALUE ) // Если ошибка
95     {
96         ErrorOutput("Error in CreateFileA function", LAB_ERR, LAB_FLAG_GET);
// Вывести сообщение
97         return LAB_ERR; // Вернуть -1
98     }
99
100     // Получение размера файла
101     *dwFileSize = GetFileSize(*hFile, NULL);
102
103     if ( *dwFileSize == INVALID_FILE_SIZE ) // Если ошибка
104     {
105         ErrorOutput("Error in GetFileSize function", LAB_ERR, LAB_FLAG_GET);
// Вывести сообщение
106         return LAB_ERR; // Вернуть -1
107     }
108
109     return LAB_OK;
110 }
111
112 //-----

```

```

113
114 int LabReadFile(HANDLE hFile, char **chFileContent, DWORD *dwFileSize)
115 {
116     BOOL bResult = FALSE; // Возвращаемое функцией значение
117     DWORD dwNumberOfBytesRead = 0; // Количество считанных байт
118     DWORD dwSum = 0; // Суммарное количество считанных байт
119     DWORD dwFilePointer = INVALID_SET_FILE_POINTER; // Смещение в файле
120
121     if ( hFile == INVALID_HANDLE_VALUE ) // Если плохой хендл
122     {
123         ErrorOutput("Error in file", LAB_BAD_FILE, LAB_FLAG_MY); // Сообщение
124         return LAB_BAD_FILE; // Вернуть ошибку
125     }
126
127     dwFilePointer = SetFilePointer(hFile, 0, NULL, FILE_BEGIN);
128
129     if ( dwFilePointer == INVALID_SET_FILE_POINTER ) // Если ошибка
130     {
131         ErrorOutput("Error in SetFilePointer function", LAB_ERR,
132 LAB_FLAG_GET); // Вывести сообщение
133         return LAB_ERR; // Вернуть -1
134     }
135
136     // Получение размера файла
137     *dwFileSize = GetFileSize(hFile, NULL);
138
139     if ( *dwFileSize == INVALID_FILE_SIZE ) // Если ошибка
140     {
141         ErrorOutput("Error in GetFileSize function", LAB_ERR, LAB_FLAG_GET);
142         // Вывести сообщение
143         return LAB_ERR; // Вернуть -1
144     }
145
146     // Выделение памяти под содержимое файла
147     *chFileContent = (char *)malloc(*dwFileSize + 1);
148
149     if ( *chFileContent == NULL ) // Если память не выделилась
150     {
151         ErrorOutput("Error in malloc function", LAB_ERR, LAB_FLAG_ERRNO); //
Вывести сообщение

```



```

150         return LAB_ERR; // Вернуть -1
151     }
152
153     // Чтение файла
154     while ( dwSum < *dwFileSize )
155     {
156         bResult = ReadFile(hFile, (LPVOID)(*chFileContent + dwSum),
157 *dwFileSize - dwSum, &dwNumberOfBytesRead, NULL);
158
159         if ( bResult == FALSE ) // Если ошибка
160         {
161             ErrorOutput("Error in ReadFile function", LAB_ERR, LAB_FLAG_GET);
162             // Вывести сообщение
163             return LAB_ERR; // Вернуть -1
164         }
165         else if ( dwNumberOfBytesRead == 0 )
166         {
167             ErrorOutput("Error in ReadFile function: dwNumberOfBytesRead is
168 0", LAB_BAD_FILE, LAB_FLAG_MY); // Вывести сообщение
169             return LAB_BAD_FILE; // Вернуть -1
170         }
171
172         dwSum += dwNumberOfBytesRead;
173     }
174
175     *dwFileSize = dwSum;
176
177     (*chFileContent)[*dwFileSize] = '\0'; // Нультерм-
178
179     return LAB_OK;
180 }
181
182 //-----
183
184 int LabWriteFile(HANDLE hFile, const char *chFileContent, DWORD *dwFileSize)
185 {
186     BOOL bResult = FALSE; // Возвращаемое функцией значение
187     DWORD dwFilePointer = INVALID_SET_FILE_POINTER; // Смещение в файле
188
189     if ( hFile == INVALID_HANDLE_VALUE ) // Если плохой хендл

```

```

187     {
188         ErrorOutput("Error in file", LAB_BAD_FILE, LAB_FLAG_MY); // Сообщение
189         return LAB_BAD_FILE; // Вернуть ошибку
190     }
191
192     dwFilePointer = SetFilePointer(hFile, 0, NULL, FILE_END);
193
194     if ( dwFilePointer == INVALID_SET_FILE_POINTER ) // Если ошибка
195     {
196         ErrorOutput("Error in SetFilePointer function", LAB_ERR,
LAB_FLAG_GET); // Вывести сообщение
197         return LAB_ERR; // Вернуть -1
198     }
199
200     // Запись в файл
201     bResult = WriteFile(hFile, (LPCVOID)(chFileContent),
strlen(chFileContent), dwFileSize, NULL);
202
203     if ( bResult == FALSE ) // Если ошибка
204     {
205         ErrorOutput("Error in WriteFile function", LAB_ERR, LAB_FLAG_GET); //
Вывести сообщение
206         return LAB_ERR; // Вернуть -1
207     }
208
209     return LAB_OK;
210 }
211
212 //-----
213
214 int LabCloseFile(HANDLE *hFile)
215 {
216     int bRet = 0;
217
218     bRet = ReleaseHandle(hFile);
219
220     return bRet;
221 }
222
223 //-----

```

```

224
225 int main()
226 {
227     printf("main\n");
228
229     DWORD len = 0;
230     int ret = LAB_OK;
231     char *buf = NULL;
232     char *command = NULL;
233     char *chFileRead = NULL;
234     // char chFileName[] = {"AlekseyShadrinov.txt"};
235     // char chFileToWrite[] = {"Hello world\n"};
236     unsigned long i = 0;
237     HANDLE hFile = INVALID_HANDLE_VALUE;
238     BOOL bRet = FALSE;
239
240     printf("malloc\n");
241
242
243     // Выделение памяти под сырой"" буфер с командой
244     buf = (char *)malloc(LAB_MAX_BUF_SIZE + 1);
245
246     // Выделение памяти под буфер с командой
247     command = (char *)malloc(LAB_MAX_BUF_SIZE + 1);
248
249     while ( 1 )
250     {
251         printf("\nWaiting for user input...\n");
252
253         // Считать команду без двух последних символов
254         fgets(buf, LAB_MAX_BUF_SIZE - 2, stdin);
255
256         printf("buf is %s", buf);
257
258         // Вычисление позиции первого символа разделителя-
259         for ( i = 0; (i < strlen(buf)) && (buf[i] != ' ') && (buf[i] != '\t')
&& (buf[i] != '\n') ; i++ ) ;
260
261         strncpy(command, buf, i);
262         command[i] = '\0';

```

```

263
264     printf("command is %s\n", command);
265
266     if ( strlen(command) == 0 )
267     {
268         fprintf(stderr, "command %s is empty\n", command);
269     }
270     else if ( strncasecmp(command, "open", strlen(command)) == 0 )
271     {
272         // argument
273         printf("enter argument: \n");
274         char file_name[127];
275         fgets(file_name, sizeof(file_name), stdin);
276         file_name[strcspn(file_name, "\n")] = 0;
277         printf("file name: %s\n", file_name);
278
279
280         // ret = LabOpenFile(chFileName, &hFile, &len);
281         ret = LabOpenFile(file_name, &hFile, &len);
282
283         // print pointer
284         printf("file handler: %d \n", hFile);
285     }
286
287     else if ( strncasecmp(command, "write", strlen(command)) == 0 )
288     {
289         // argument
290         printf("enter handle argument: \n");
291         HANDLE hFile;
292         char hFile_str[10];
293         fgets(hFile_str, 10, stdin);
294         hFile = (void *)strtoul(hFile_str, NULL, 10);
295         printf("pointer: %d\n", hFile);
296
297         // argument
298         printf("enter file value: \n");
299         char file_value[256];
300         fgets(file_value, sizeof(file_value), stdin);
301         file_value[strcspn(file_value, "\n")] = 0;
302         printf("file value: %s\n", file_value);

```

```

303
304         ret = LabWriteFile(hFile, file_value, &len);
305     }
306     else if ( strncasecmp(command, "read", strlen(command)) == 0 )
307     {
308         // argument
309         printf("enter handle argument: \n");
310         HANDLE hFile;
311         char hFile_str[10];
312         fgets(hFile_str, 10, stdin);
313         hFile = (void *)strtoul(hFile_str, NULL, 10);
314         printf("pointer: %d\n", hFile);
315
316         ret = LabReadFile(hFile, &chFileRead, &len);
317
318         // Вывести на экран содержимое файла
319         printf("Content of the file is:\n%s\n", chFileRead);
320         ReleaseVoidMemory((void **)&chFileRead);
321     }
322     else if ( strncasecmp(command, "close", strlen(command)) == 0 )
323     {
324         // argument
325         printf("enter handle argument: \n");
326         HANDLE hFile;
327         char hFile_str[10];
328         fgets(hFile_str, 10, stdin);
329         hFile = (void *)strtoul(hFile_str, NULL, 10);
330         printf("pointer: %d\n", hFile);
331
332         ret = LabCloseFile(&hFile);
333     }
334     else if ( strncasecmp(command, "end", strlen(command)) == 0 )
335     {
336         break;
337     }
338     else
339     {
340         fprintf(stderr, "command %s error\n", command);
341     }
342 }

```



```
343
344     // Освободить память, если не освобождена
345
346     ReleaseVoidMemory((void **)&buf);
347     ReleaseVoidMemory((void **)&command);
348     ReleaseVoidMemory((void **)&chFileRead);
349
350     // Закрыть файл, если не закрыт
351
352     bRet = ReleaseHandle(&hFile);
353
354     return ret;
355 }
```