

[Team Mission 6]

Making Own your Debugger

Team GAZUAAA - Final Draft

sLBsJi
B iWsj ikuJi vLiIY rlv
B B uV Ji iB uK BJ Bi
B iB sK iB su ur Bi
B iBirii sUiBiB Ju Vr Bi
B iQ B B jk Sv Bi
B WvQv VuisB VX EJ Br

iPuYBv	jB	uLMPq	iB	B	kB	Sq	BV
Xu rB	Bji		Vr iB	B	Bji	BU	UQ
Vv	ivij	iK	B	B	viis	viiv	Yi iv
Vv	U B	Bi	iB	B	I B	B B	Q V
kL rsB	B Ji	vK	B	B	ji	B ui	iU Q
Sv B	iQriUs	Bi	iB	B	iQiiBv	rQiiBv	vBiiBr
Su B	uv i iB	vs	iB	B	uv iB	Ui iiV	ki il
iKuvBr	B Bi	BUyji	ulvXL	B	Bi B	BiiB	Qi

Team GAZUAAA

팀원 :

14012848 최재현 (조장)

15012977 고창훈

15012978 서성관

15012981 이소영

16011783 박근진

Contents

1	Version #1, Version#2 실습 (디버깅)	2
1.1	Version#1	2
1.2	Version#2	5
1.3	General-Purpose Register	11
2	Version#2 소스코드 분석	13
2.1	Source Code	14
2.2	Functions	17
2.3	Flag Register 란?	19
3	Version #2의 디버거에 여러분 만의 기능을 추가하시오	20
3.1	모든 EVENT CODE, STATUS 출력	20
3.2	항목별 색 구분 및 로고 삽입	22
3.3	증복없는 로그파일 저장	23
4	Version#2를 GUI로 구현하시오	25
4.1	개발 환경	25
4.2	GUI 설명	25
4.3	Source Code	27
5	References	37

Assignment contents

1. Lecture Note 12의 “Making own your debugger” Version #1, Version #2를 컴파일하여 특정 프로세스를 대상으로 디버깅을 해보시오.

Version1의 소스코드는 간단한 Debug 코드이다. Version1의 소스코드에서 디버그 이벤트가 수신되면 printf 함수를 사용하여 이벤트의 내용을 표시하게 된다. 그 다음 union defined structure에 접근하여 Debug object의 정보를 얻게된다.

`ContinueDebugEvent` 함수를 호출하면 디버거가 작업을 다시 시작하게 되는데,

```
61     ContinueDebugEvent(
```

이 시점에서 디버거는 `WaitForDebugEvent` 함수로 돌아가서 다음 디버그 이벤트를 기다리게 된다.

```
24     |         if (!WaitForDebugEvent(&dbg_event, INFINITE))  
25     |             break;
```

디버거는 프로세스 생성, 스레드 생성, DLL로드 및 언로드와 같은 이벤트를 잡아낸다.

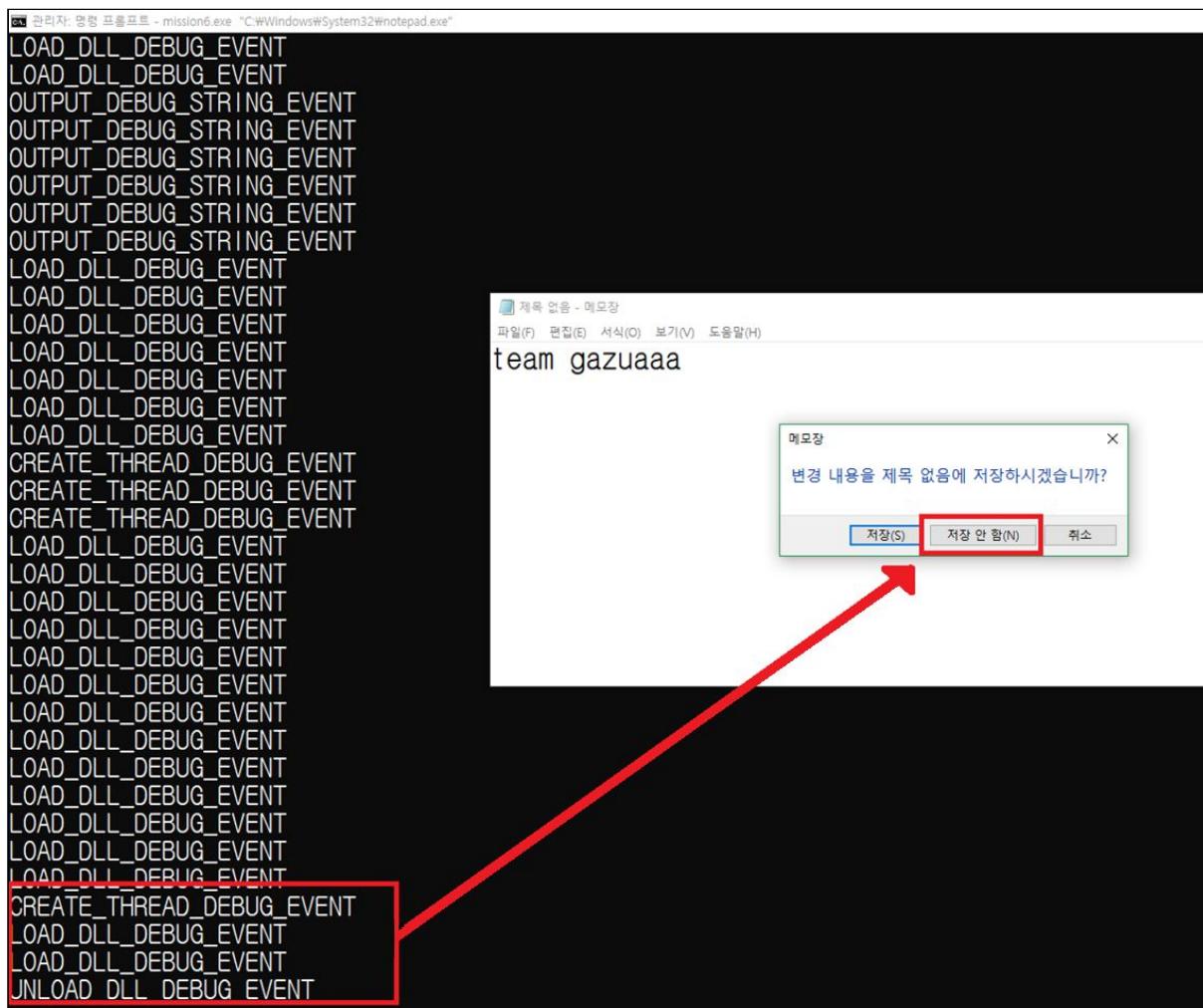
1.1 실습과정 - Version 1

Version1 을 컴파일&빌드 한 결과물인 mission6.exe 파일로 밑의 사진과 같이 "C:\Windows\System32\notepad.exe" (메모장 프로그램)를 디버깅 하였다.

C:\Users\최재현\Desktop\mission6\Debug>mission6.exe "C:\Windows\System32\notepad.exe"

디버깅을 시작하자 다음과 같이 우선 Process Debug EVENT를 생성하고 DLL을 불러오면서 THREAD DEBUG EVENT를 생성하는 것을 확인 할 수 있었다.

위의 EVENT들이 끝나고 나선 위의 사진과 같이 notepad.exe 가 실행되는 것을 확인 할 수 있다.



notepad.exe 를 실행 후 저장 여부를 묻는 팝업 창에서 위와 같이 새로운 DEBUG EVENT 들이 생성되는 것을 확인 할 수 있었으며.

```

LOAD_DLL_DEBUG_EVENT
UNLOAD_DLL_DEBUG_EVENT
EXIT_THREAD_DEBUG_EVENT
EXIT_THREAD_DEBUG_EVENT
EXIT_THREAD_DEBUG_EVENT
EXIT_THREAD_DEBUG_EVENT
EXIT_THREAD_DEBUG_EVENT
EXIT_THREAD_DEBUG_EVENT
EXIT_THREAD_DEBUG_EVENT
EXIT_THREAD_DEBUG_EVENT
EXIT_PROCESS_DEBUG_EVENT

```

C:\Users\최재현\source\repos\mission6\Debug>

notepad.exe를 save 과정 없이 종료 시 위와 같이 EXIT THREAD DEBUG EVENT 가 발생하면서 PROCESS인 notepad.exe가 종료되게 된다.

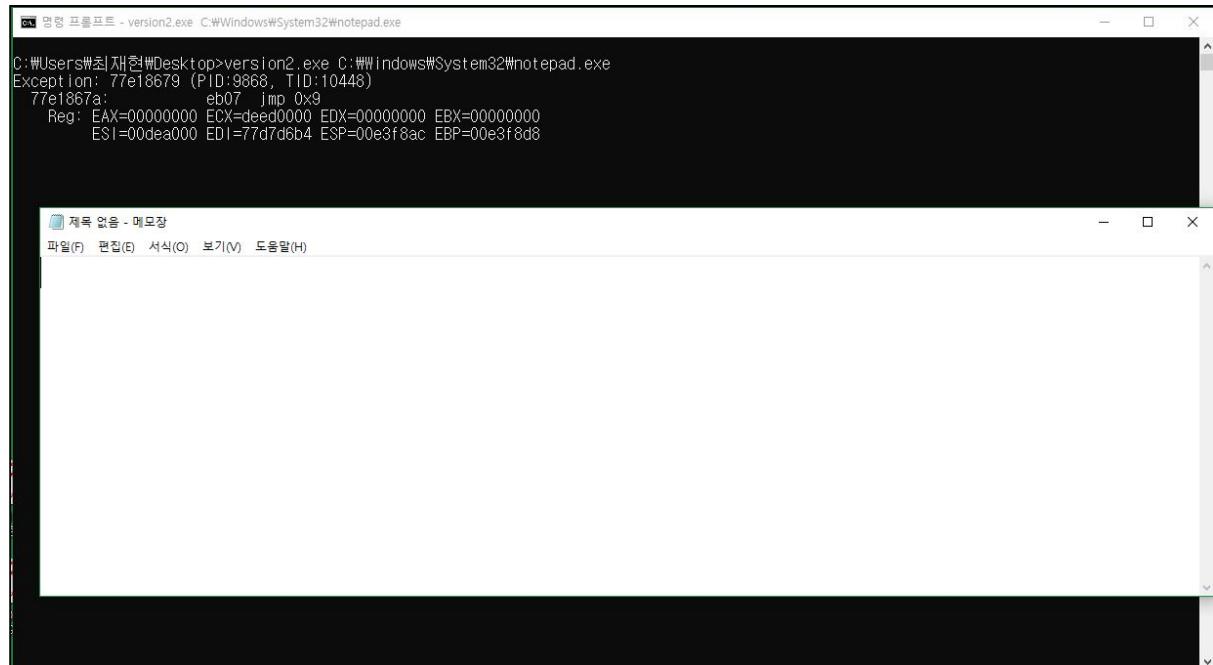
LOAD_DLL_DEBUG_EVENT	UNLOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
CREATE_THREAD_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
CREATE_THREAD_DEBUG_EVENT	UNLOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
LOAD_DLL_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
LOAD_DLL_DEBUG_EVENT	CREATE_THREAD_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
UNLOAD_DLL_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
EXIT_THREAD_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
LOAD_DLL_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
CREATE_THREAD_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
LOAD_DLL_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
LOAD_DLL_DEBUG_EVENT	CREATE_THREAD_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
LOAD_DLL_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
LOAD_DLL_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
LOAD_DLL_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
CREATE_THREAD_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
CREATE_THREAD_DEBUG_EVENT	UNLOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
LOAD_DLL_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
LOAD_DLL_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
LOAD_DLL_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
UNLOAD_DLL_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
LOAD_DLL_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT	OUTPUT_DEBUG_STRING_EVENT
LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT	OUTPUT_DEBUG_STRING_EVENT
LOAD_DLL_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
CREATE_THREAD_DEBUG_EVENT	UNLOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
EXIT_THREAD_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
CREATE_THREAD_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT	OUTPUT_DEBUG_STRING_EVENT
LOAD_DLL_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
LOAD_DLL_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
CREATE_THREAD_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
LOAD_DLL_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
CREATE_THREAD_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
LOAD_DLL_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
CREATE_THREAD_DEBUG_EVENT	UNLOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
CREATE_THREAD_DEBUG_EVENT	UNLOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
CREATE_THREAD_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
LOAD_DLL_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
LOAD_DLL_DEBUG_EVENT	UNLOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
LOAD_DLL_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
LOAD_DLL_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	UNLOAD_DLL_DEBUG_EVENT
LOAD_DLL_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	EXIT_THREAD_DEBUG_EVENT

LOAD_DLL_DEBUG_EVENT	CREATE_THREAD_DEBUG_EVENT	UNLOAD_DLL_DEBUG_EVENT
LOAD_DLL_DEBUG_EVENT	EXIT_THREAD_DEBUG_EVENT	UNLOAD_DLL_DEBUG_EVENT
UNLOAD_DLL_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	UNLOAD_DLL_DEBUG_EVENT
LOAD_DLL_DEBUG_EVENT	UNLOAD_DLL_DEBUG_EVENT	UNLOAD_DLL_DEBUG_EVENT
LOAD_DLL_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	UNLOAD_DLL_DEBUG_EVENT
LOAD_DLL_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT
LOAD_DLL_DEBUG_EVENT	LOAD_DLL_DEBUG_EVENT	EXIT_THREAD_DEBUG_EVENT
LOAD_DLL_DEBUG_EVENT	CREATE_THREAD_DEBUG_EVENT	EXIT_THREAD_DEBUG_EVENT
LOAD_DLL_DEBUG_EVENT	EXIT_THREAD_DEBUG_EVENT	EXIT_THREAD_DEBUG_EVENT
LOAD_DLL_DEBUG_EVENT	CREATE_THREAD_DEBUG_EVENT	EXIT_THREAD_DEBUG_EVENT
LOAD_DLL_DEBUG_EVENT	EXIT_THREAD_DEBUG_EVENT	EXIT_THREAD_DEBUG_EVENT
UNLOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT	EXIT_THREAD_DEBUG_EVENT
LOAD_DLL_DEBUG_EVENT	UNLOAD_DLL_DEBUG_EVENT	EXIT_THREAD_DEBUG_EVENT
LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT	EXIT_THREAD_DEBUG_EVENT
LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT	EXIT_THREAD_DEBUG_EVENT
UNLOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT	EXIT_THREAD_DEBUG_EVENT
LOAD_DLL_DEBUG_EVENT	OUTPUT_DEBUG_STRING_EVENT	EXIT_THREAD_DEBUG_EVENT
		EXIT_PROCESS_DEBUG_EVENT

저장을 눌러 string value 인 team gazuaaa 를 바탕화면에 1.txt로 저장한 경우 위와 같은 DEBUG EVENT 들이 발생하는 것을 확인 할 수 있었으며, 이때 SAVE 과정을 위해 OUTPUT_DEBUG_EVENT 가 발생하는 것을 확인 할 수 있었다.

1.2 실습과정 - Version 2

< notepad.exe >

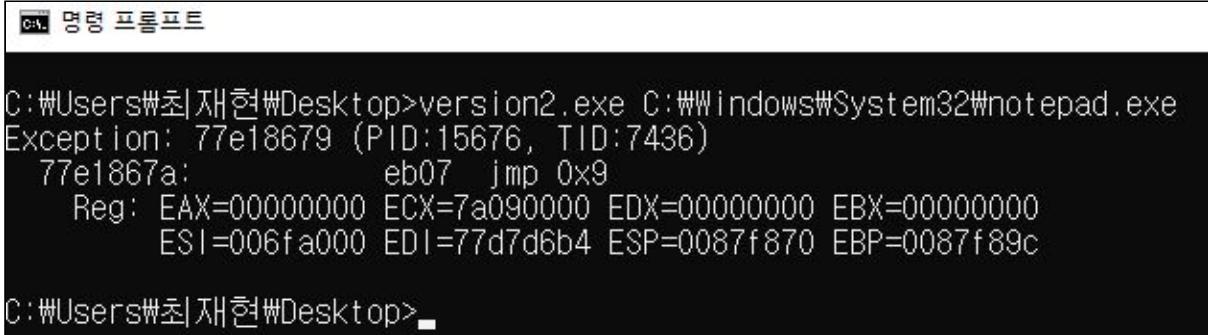


Version2 를 컴파일&빌드 한 결과물인 version2.exe 파일로 위의 사진과 같이 "C:\Windows\System32\notepad.exe" (메모장 프로그램)를 디버깅 하였다.

```
Exception: 77e18679 (PID:9868, TID:10448)
77e1867a: eb07 jmp 0x9
Reg: EAX=00000000 ECX=deed0000 EDX=00000000 EBX=00000000
ESI=00dea000 EDI=77d7d6b4 ESP=00e3f8ac EBP=00e3f8d8
```

처음 디버깅을 시작하자 Version1 처럼 target(notepad.exe)가 실행되면서 위 사진과 같이 Exception과 PID & TID, 그리고 실행이 시작되는 메모리 주소인 77e1867a (eb07 jmp 0x9)

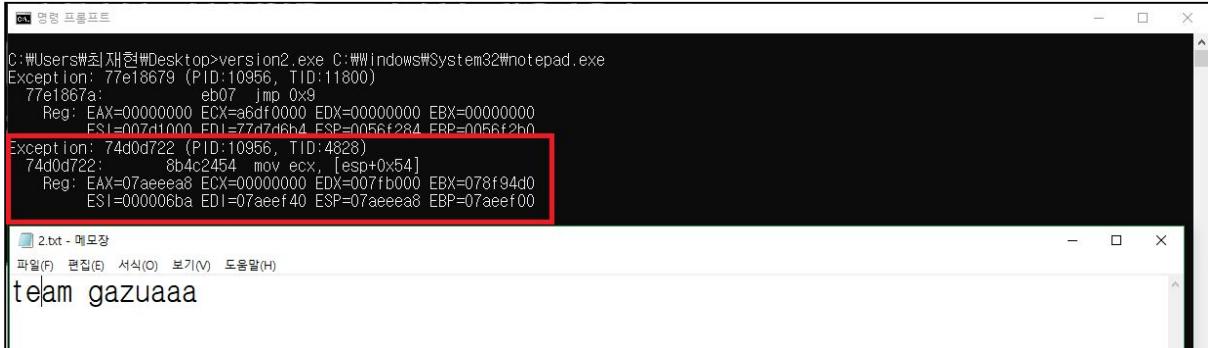
가 출력된다. 또한 target의 EAX,ECX,EDX,EBX, ESI, EDI,ESP,EBP 등 레지스터의 정보 또한 출력 되는 것을 확인 할 수 있다.



```
C:\#Users#\최재현\Desktop>version2.exe C:\Windows\System32\notepad.exe
Exception: 77e18679 (PID:15676, TID:7436)
77e1867a: eb07 jmp 0x9
Reg: EAX=00000000 ECX=7a090000 EDX=00000000 EBX=00000000
ESI=006fa000 EDI=77d7d6b4 ESP=0087f870 EBP=0087f89c

C:\#Users#\최재현\Desktop>
```

실행중이던 notepad.exe를 저장하지 않고 끝내자 위 사진처럼 디버거인 version2.exe 도 같이 종료 되는 것을 확인 할 수 있다.



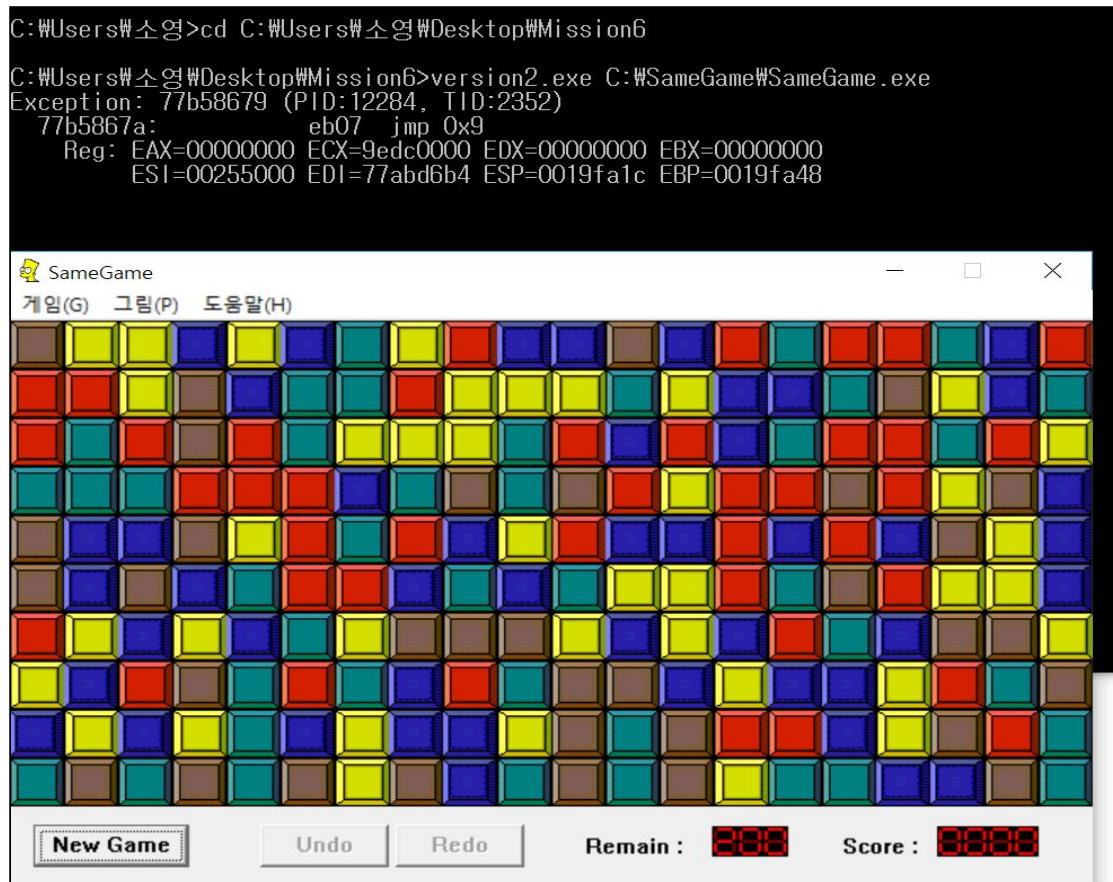
```
C:\#Users#\최재현\Desktop>version2.exe C:\Windows\System32\notepad.exe
Exception: 77e18679 (PID:10956, TID:11800)
77e1867a: eb07 jmp 0x9
Reg: EAX=00000000 ECX=a6df0000 EDX=00000000 EBX=00000000
ESI=007f1000 EDI=77d7d6b4 ESP=0056f284 EBP=0056f2b0
Exception: 74d0d722 (PID:10956, TID:4828)
74d0d722: 8b4c2454 mov ecx, [esp+0x54]
Reg: EAX=07aeeeea8 ECX=00000000 EDX=007fb000 EBX=078f94d0
ESI=000006ba EDI=07aeeef40 ESP=07aeeeea8 EBP=07aeeef00

2.txt - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
team gazuaaa
```

```
Exception: 74d0d722 (PID:10956, TID:4828)
74d0d722: 8b4c2454 mov ecx, [esp+0x54]
Reg: EAX=07aeeeea8 ECX=00000000 EDX=007fb000 EBX=078f94d0
ESI=000006ba EDI=07aeeef40 ESP=07aeeeea8 EBP=07aeeef00
```

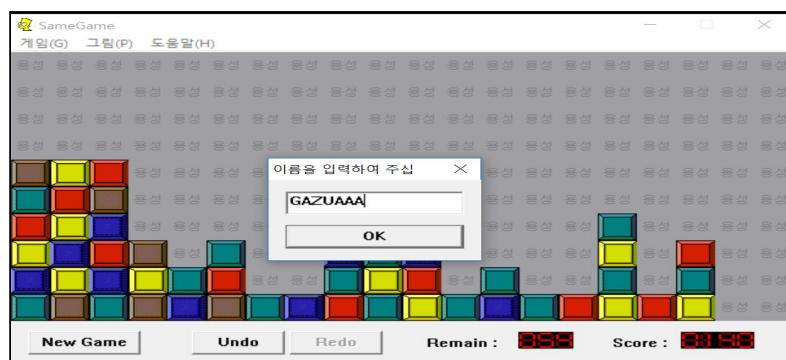
version1 때와 마찬가지로 메모장에 ‘team gazuaaa’ 를 입력하고 2.txt로 저장을 수행하자 위 사진과 같이 새로운 Exception 이 발생하는 것을 확인 할 수 있으며, 참조하는 메모리의 주소값과 Disassemble 된 내용, 그리고 레지스터의 정보들이 출력된다.

< 다른 프로그램으로 실행한 결과 - SameGame >



같은 색깔의 인접한 사각형들을 없애는 게임인 SameGame 파일을 다운받았다.

version2.exe 파일로 위의 사진과 같이 "C:\SameGame"에 있는 SameGame.exe를 디버깅하였다.

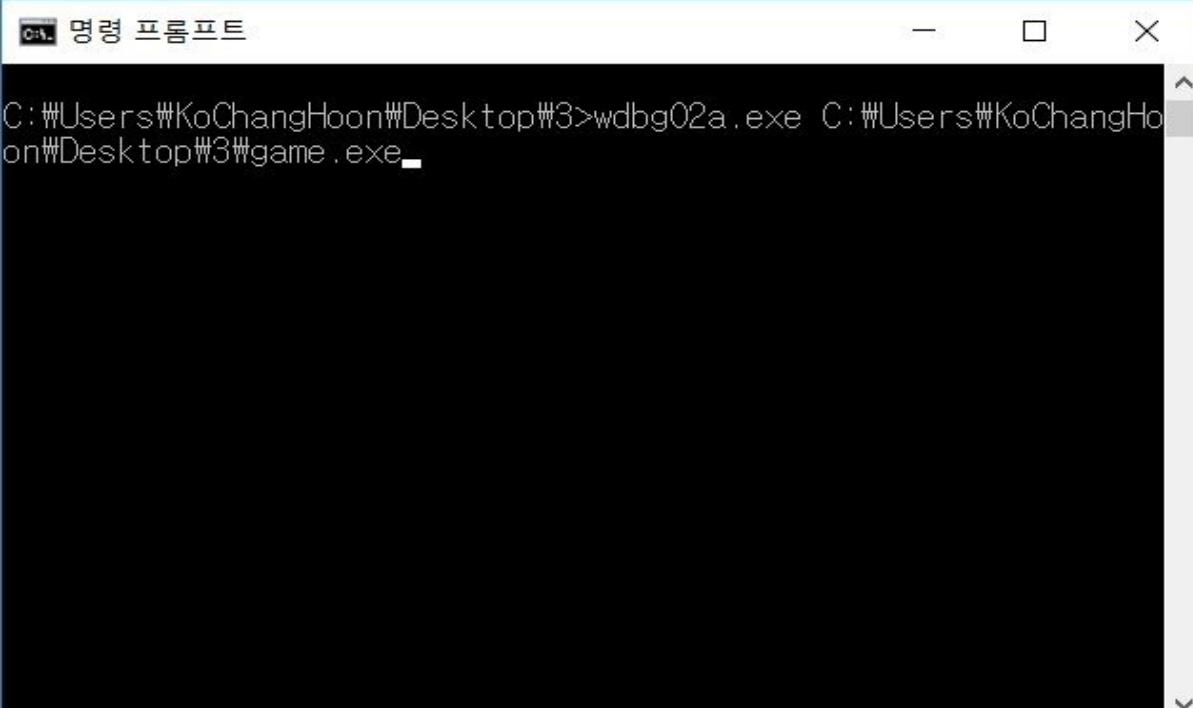


```
C:\Users\소영\Desktop\Mission6>version2.exe C:\SameGame\SameGame.exe
Exception: 77b58679 (PID:12284, TID:2352)
77b5867a: eb07 jmp 0x9
Reg: EAX=00000000 ECX=9edc0000 EDX=00000000 EBX=00000000
ESI=00255000 EDI=77abd6b4 ESP=0019fa1c EBP=0019fa48
```

```
C:\Users\소영\Desktop\Mission6>
```

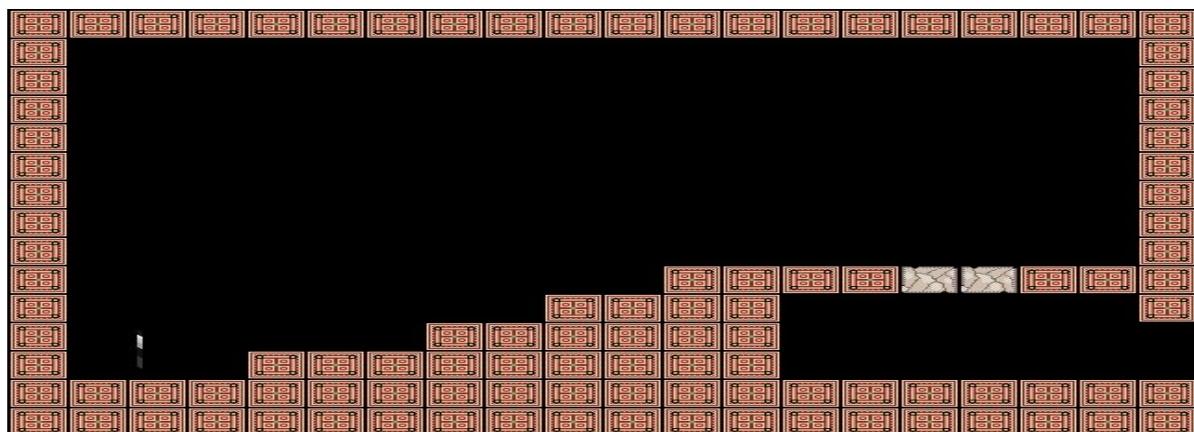
위의 사진과 같이 게임이 종료되면 디버거 version2.exe도 같이 종료된다.

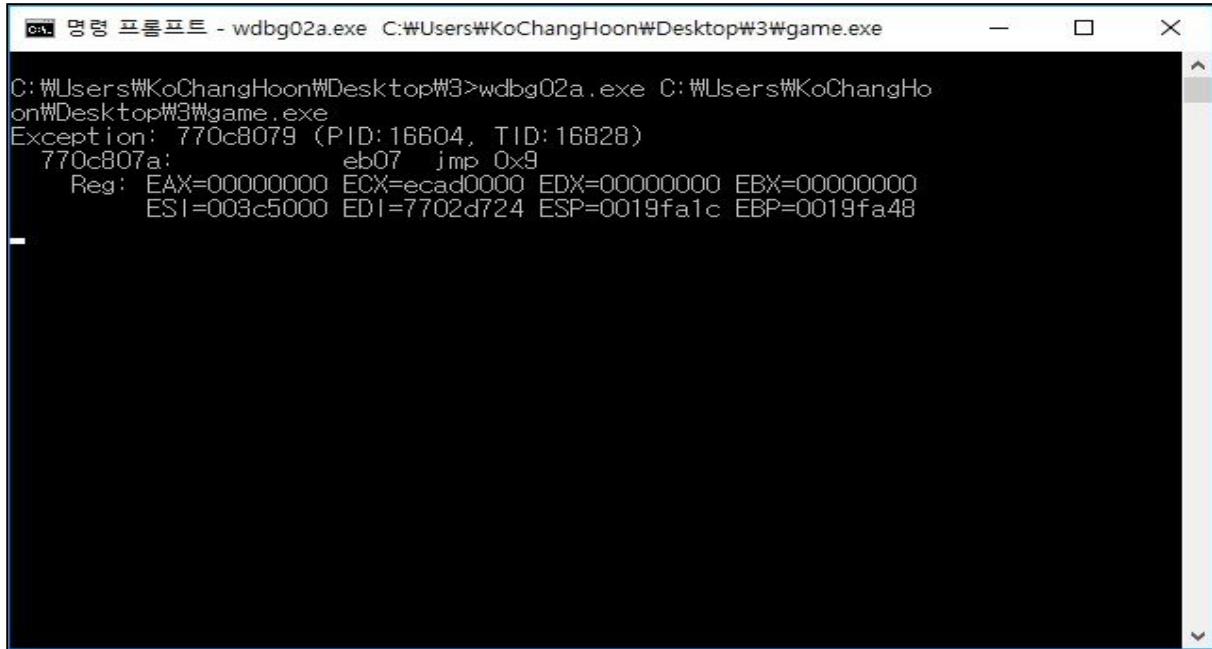
< 다른 프로그램으로 실행한 결과 - 공튀기기 게임 >



```
C:\Users\KoChangHoon\Desktop\3>wdbg02a.exe C:\Users\KoChangHoon\Desktop\3\game.exe
```

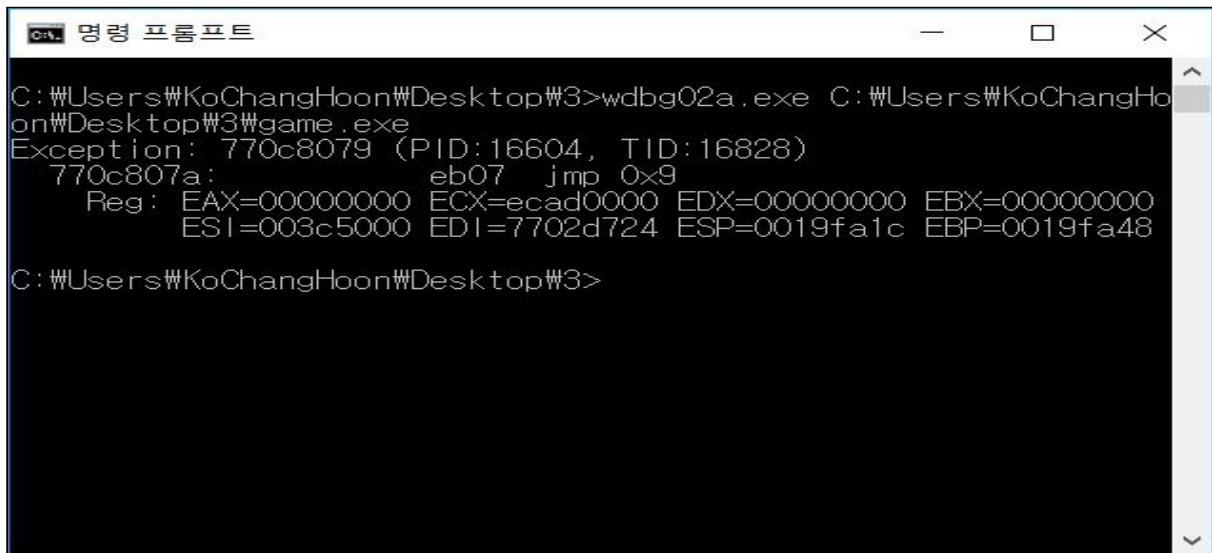
cmd창을 열고, version2 파일(wdbg02.exe) 파일을 실시키고, 같은 폴더내에 있는 game.exe파일을 디버깅 해보았다.





```
C:\Users\KoChangHoon\Desktop\3>wdbg02a.exe C:\Users\KoChangHoon\Desktop\3\game.exe
Exception: 770c8079 (PID:16604, TID:16828)
 770c807a: eb07 jmp 0x9
  Reg: EAX=00000000 ECX=ecad0000 EDX=00000000 EBX=00000000
        ESI=003c5000 EDI=7702d724 ESP=0019fa1c EBP=0019fa48
```

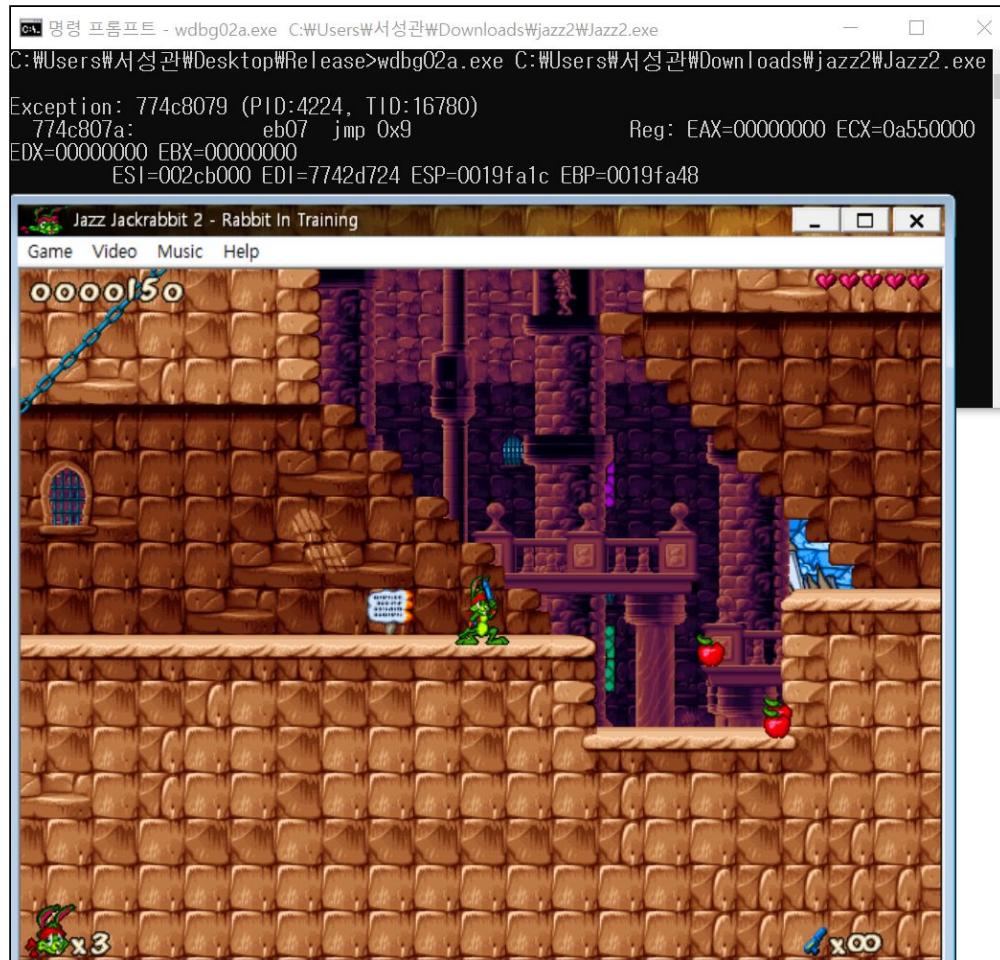
game.exe파일을 실행화면이고 그와 동시에 디버깅이 실행되었다.



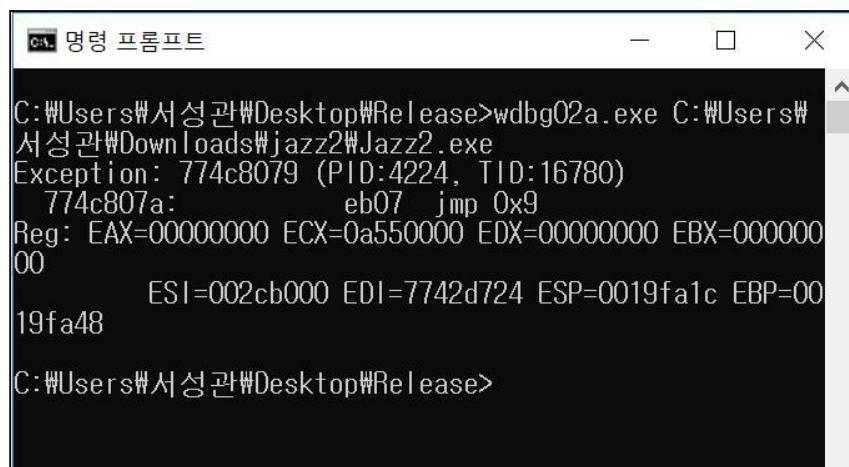
```
C:\Users\KoChangHoon\Desktop\3>wdbg02a.exe C:\Users\KoChangHoon\Desktop\3\game.exe
Exception: 770c8079 (PID:16604, TID:16828)
 770c807a: eb07 jmp 0x9
  Reg: EAX=00000000 ECX=ecad0000 EDX=00000000 EBX=00000000
        ESI=003c5000 EDI=7702d724 ESP=0019fa1c EBP=0019fa48
C:\Users\KoChangHoon\Desktop\3>
```

프로그램을 종료하자, 디버거도 작동이 중지되었다.

< 다른 프로그램으로 실행한 결과 - Jazz Jackrabbit 2 >



cmd창으로 version2 파일(wdbg02.exe 파일)을 실행 시켜서 C:\Users\서성관\Downloads\jazz2에 있는 Jazz2.exe를 디버깅 했다.



프로그램이 종료되면 디버거도 종지된다.

1.3 General-Purpose Register

32bit General-Purpose Registers	
EAX	EBP
EBX	ESP
ECX	ESI
EDX	EDI

EFLAGS	AF	CF	OF	PF	SF	ZF
--------	----	----	----	----	----	----

EIP	다음에 실행할 명령어의 주소를 가리킨다.
-----	------------------------

범용 레지스터에는 일반 레지스터 4개(EAX, ECX, EDX, EBX)와 일반주소 레지스터 4개(ESP, EBP, ESI, EDI)를 가지고 있으며, 일반레지스터에 속하지 않는 EIP 레지스터도 있다.

※ EIP는 명령 포인터 레지스터이다.

레지스터의 종류로는 범용, 명령포인터, 세그먼트 레지스터 등이 있다.

※ 범용레지스터의 각각의 레지스터는 4byte의 크기를 가진다.

범용 레지스터는 연산 결과를 받을 수도 있으며, 연산에 사용되어질 수도 있는, 필요에 따라서 자유롭게 사용이 가능한 레지스터이다.

(필요에 따라 자유롭게 사용하도록 CPU가 제공하는 레지스터)

General-Purpose Register

EAX (Extended Accumulator Register)

산술연산 명령에서 상수/변수 값을 저장하거나 함수의 리턴 값이 저장되는 용도로도 사용된다.

EBX (Extended Base Register)

EBX는 DS 세그먼트에 대한 포인터를 주로 저장하고 ESI나 EDI와 결합하여 인덱스에 사용된다. EBX는 메모리 주소 지정을 확장하기 위해 인덱스로 사용될 수 있는 유일한 범용 레지스터이다.

ECX (Extended Counter Register)

반복 명령어 사용 시 반복 카운터로 사용된다. ECX 레지스터에 반복할 횟수를 지정해 놓고 반복 작업을 수행하게 된다.

EDX (Extended Data Register)

데이터 레지스터이다. 입, 출력 포인터 값을 저장할 때 사용한다.

ESI (Extended Source Index)

데이터 복사나 조작 시 Source Data의 주소가 저장된다. ESI 레지스터가 가리키는 주소의 데이터를 EDI 레지스터가 가리키는 주소로 복사 하는 용도로 많이 사용된다.

EDI (Extended Destination Index)

복사 작업 시 Destination 의 주소가 저장된다. 주로 ESI 레지스터가 가리키는 주소의 데이터가 복사된다.

ESP (Extended Stack Pointer)

하나의 스택 프레임의 끝 지점 주소가 저장된다. PUSH, POP 명령어에 따라서 ESP의 값이 4Byte씩 변한다.

EBP (Extended Base Pointer)

하나의 스택 프레임의 시작 지점 주소가 저장된다. 현재 사용되는 스택 프레임이 소멸되지 않는 동안 EBP 의 값은 변하지 않는다. 현재의 스택 프레임이 소멸되면 이전에 사용되던 스택 프레임을 가리키게 된다.

EIP (Extended Instruction Pointer)

다음에 실행해야 할 명령어가 존재하는 메모리 주소가 저장된다. 현재 명령어를 실행 완료한 후에 EIP 레지스터에 저장되어 있는 주소에 위치한 명령어를 실행하게 된다. 실행 전 EIP 레지스터에는 다음 실행해야 할 명령어가 존재하는 주소의 값이 저장된다.

2. Version #2의 소스코드를 분석하시오.

Version2 의 소스코드는 disassembly function을 Implement 하는 코드이다.

DEBUG과정 중 Exception 이 발생하게 되면 예외가 발생한 주소와 현재 레지스터 값을 표시할 수 있으며, 예외가 발생할 때 실행 된 명령도 표시 할 수 있게 되고, 이를 통해 disassemble 할 부분을 찾을 수 있게 된다.

Version2에서 사용하는 header 파일인 udis86.h 은 오픈소스 기반의 디스어셈블러이다.

- <https://github.com/vmt/udis86>
- <http://udis86.sourceforge.net>

```
#include <stdio.h>
#include <udis86.h>

int main()
{
    ud_t ud_obj;

    ud_init(&ud_obj);
    ud_set_input_file(&ud_obj, stdin);
    ud_set_mode(&ud_obj, 64);
    ud_set_syntax(&ud_obj, UD_SYN_INTEL);

    while (ud_disassemble(&ud_obj)) {
        printf("\t%s\n", ud_insn_asm(&ud_obj));
    }

    return 0;
}
```

Usage Example of Udis86

2.1 Source Code

```
#include "stdafx.h"

#include <Windows.h>
#include "udis86.h" //디스어셈블링을 위한 udis86.h

#pragma comment(lib, "libudis86.lib") //udis86의 라이브러리 파일

int sj_disassembler(unsigned char *buff, char *out, int size)
{ //기계어를 분해하는 역할을 담당한다. 여기서 udis86의 기능이 사용되게 된다.

    ud_t ud_obj;
    ud_init(&ud_obj);
    ud_set_input_buffer(&ud_obj, buff, 32);
    ud_set_mode(&ud_obj, 32);
    ud_set_syntax(&ud_obj, UD_SYN_INTEL);

    if(ud_disassemble(&ud_obj)){
        sprintf_s(out, size, "%14s %s", ud_insn_hex(&ud_obj), ud_insn_asm(&ud_obj));
    }else{
        return -1;
    }
    return (int)ud_insn_len(&ud_obj);
}

int exception_debug_event(DEBUG_EVENT *pde)
{ //exception이 발생하면 실행되며, 밑의 함수들을 호출
/* Open Process,ReadProcessMemory, OpenThread, GetThreadContext, SetThreadContext */
    // 이 함수와 WriteProcessMemory number 는 다른 프로세스에 access 하는데 필요

    DWORD dwReadBytes;

    HANDLE process_hander= OpenProcess( // 프로세스 열기 , 헨들러에 할당
        PROCESS_VM_WRITE | PROCESS_VM_READ | PROCESS_VM_OPERATION,
        FALSE, pde->dwProcessId);

    if(!process_hander) return -1;

    HANDLE thread_handler= OpenThread(THREAD_GET_CONTEXT | THREAD_SET_CONTEXT, FALSE,
        pde->dwThreadId);
    //Open Thread : Register 정보 읽기위해 필요한 function

    if(!thread_handler) return -1;

    CONTEXT ctx;
    ctx.ContextFlags= CONTEXT_ALL; // related CONTEXT Thread
    GetThreadContext(thread_handler, &ctx);
    //GetThreadContext 통해 레지스터를 Read 할 수 있다.
```

```

char asm_string[256];
unsigned char asm_code[32];

ReadProcessMemory(process_hander, (VOID *)ctx.Eip, asm_code, 32, &dwReadBytes);
//exception_debug_event 함수에서 exception이 발생할때 실행 된 명령을 얻기 위해 필요

if(sj_disassembler(asm_code, asm_string, sizeof(asm_string)) == -1)
asm_string[0] = '\0';

printf("Exception: %08x (PID:%d, TID:%d)\n",
pde->u.Exception.ExceptionRecord.ExceptionAddress,
pde->dwProcessId, pde->dwThreadId);
printf(" %08x: %s\n", ctx.Eip, asm_string);
printf(" Reg: EAX=%08x ECX=%08x EDX=%08x EBX=%08x\n",
ctx.Eax, ctx.Ecx, ctx.Edx, ctx.Ebx);
printf(" ESI=%08x EDI=%08x ESP=%08x EBP=%08x\n",
ctx.Esi, ctx.Edi, ctx.Esp, ctx.Ebp);

SetThreadContext(thread_handler, &ctx);
//SetThreadContext를 통해 레지스터를 Write 할 수 있다.
CloseHandle(thread_handler);
CloseHandle(process_hander);
return 0;
}// End of exception_debug_event

int _tmain(int argc, _TCHAR* argv[])
{
    STARTUPINFO startup_info;
    PROCESS_INFORMATION process_info;

    if(argc < 2){
        fprintf(stderr, "C:\\>%s <sample.exe>\n", argv[0]); //Usage example output
        return 1;
    }
    // 생성될 프로세스의 정보가 저장될 구조체를 선언
    memset(&process_info, 0, sizeof(process_info));
    // startup_info 구조체의 정보를 명시
    memset(&startup_info, 0, sizeof(startup_info));
    startup_info.cb= sizeof(STARTUPINFO);

    BOOL r = CreateProcess( //Create the Process
NULL, argv[1], NULL, NULL, FALSE,
NORMAL_PRIORITY_CLASS | CREATE_SUSPENDED | DEBUG_PROCESS,
NULL, NULL, &startup_info, &process_info);
    if(!r)
        return -1;
/*=====BOOL return 함수의 경우 return 0; 와 겹쳐 ERROR 가 나므로 이름을 r 로 변경=====*/
    ResumeThread(process_info.hThread);

    int process_counter = 0; //프로세스 카운터 선언
}

```

```

do{
    DEBUG_EVENT dbg_event;
    if(!WaitForDebugEvent(&dbg_event, INFINITE)) break;

    DWORD dwContinueStatus = DBG_CONTINUE;
    //EXCEPTION_DEBUG_EVENT 발생 시 디버거에서 직접 예외를 처리

    switch(dbg_event.dwDebugEventCode) //디버그 이벤트 유형으로 Switch Case 를 나눔
    {
        case CREATE_PROCESS_DEBUG_EVENT:
            process_counter++;
            break;
        case EXIT_PROCESS_DEBUG_EVENT:
            process_counter--;
            break;
        case EXCEPTION_DEBUG_EVENT:
            if(dbg_event.u.Exception.ExceptionRecord.ExceptionCode != EXCEPTION_BREAKPOINT)
            {
                dwContinueStatus = DBG_EXCEPTION_NOT_HANDLED;
            }
            exception_debug_event(&dbg_event);
            //exception_debug_event 함수에서 레지스터 값을 다시 쓸 필요가 없으므로
            //SetThreadContext 함수를 호출 할 필요가 없어지게 된다.
            break;
    }

    ContinueDebugEvent(
        dbg_event.dwProcessId, dbg_event.dwThreadId, dwContinueStatus);

}while(process_counter > 0);

CloseHandle(process_info.hThread); //Thread 종료
CloseHandle(process_info.hProcess); //Process 종료
return 0;
}

```

Windows에선 프로그램이 대상 프로세스의 디버거로 마운트되지 않은 경우에도 대상 프로세스의 핸들을 얻을 수 있는 한 프로세스의 메모리 공간을 자유롭게 읽고 쓸 수 있다.

현재 사용자에게 해당 권한이 없으면 OpenProcess 호출은 실패하지만 프로세스 핸들을 다른 메소드를 통해 가져올 수 있는 한 프로세스의 메모리 공간을 자유롭게 읽고 쓸 수 있다.

2.2 Functions

OpenProcess function

```
HANDLE WINAPI OpenProcess(
    _In_ DWORD dwDesiredAccess, // access flags
    _In_ BOOL bInheritHandle, // handle inheritance options
    _In_ DWORD dwProcessId // Process ID
);
```

exception_debug_event 함수에서 exception이 발생할때 실행 된 명령을 얻기 위해 ReadProcessMemory 함수가 필요하다.

ReadProcessMemory function

```
BOOL WINAPI ReadProcessMemory(
    _In_ HANDLE hProcess, //process handle
    _In_ LPCVOID lpBaseAddress, //Read starting address
    _Out_ LPVOID lpBuffer, // buffer for storing data
    _In_ SIZE_T nSize, //number of bytes to read
    _Out_ SIZE_T *lpNumberOfBytesRead //The actual number of bytes read
);
```

WriteProcessMemory function

```
BOOL WINAPI WriteProcessMemory(
    _In_ HANDLE hProcess, //process handle
    _In_ LPVOID lpBaseAddress, //Write start address
    _In_ LPVOID lpBuffer, //data buffer
    _In_ SIZE_T nSize, //number of bytes to write
    _Out_ SIZE_T *lpNumberOfBytesWritten //The number of bytes actually written
);
```

다음과정은 레지스터를 읽는 것 이다.

thread를 OpenThread 를 통해 레지스터를 읽고나면, GetThreadContext 와 SetThreadContext를 통해 레지스터를 Read & Write 할 수 있다.

exception이 발생하게 되면 따로 선언해둔 함수인 exception_debug_event 함수로 가게 되는데 이때 _tmain 의 DWORD dwContinueStatus = DBG_CONTINUE 명령을 통해 EXCEPTION_DEBUG_EVENT 발생 시 디버거에서 직접 예외를 처리하게 된다.

ContinueDebugEvent

⇒ WaitForDebugEvent로 멈춰진 디버그 프로세스의 Thread를 다음 동작을 하도록 지시한다.

```
BOOL WINAPI ContinueDebugEvent(
    _In_ DWORD dwProcessId, //pid
    _In_ DWORD dwThreadId, //tid
    _In_ DWORD dwContinueStatus //처리할 방법
);
```

- dwProcessId : pid
- dwThreadId : tid
- dwContinueStatus 의 경우 밑의 표와 같다.

dwContinueStatus

dwContinueStatus	설명
DBG_CONTINUE 0x00010002L	EXCEPTION_DEBUG_EVENT 발생시 디버거에서 직접 예외를 처리
DBG_EXCEPTION_NOT_HANDLED 0x80010001L	EXCEPTION_DEBUG_EVENT 발생시 대상프로세스에 설정된 예외처리 루틴을 실행
DBG_REPLY_LATER 0x40010001L	Windows 10, 버전 1507 이상에서 추가된 루틴, dwThreadId 가 target 의 event 가 계속 될 경우 이를 breaking 하게 만든다. dwThreadId 에 대해 SuspendThread API 를 호출하면 디버거가 프로세스의 다른 스레드를 다시 시작한 다음 나중에 깨는 상태로 되돌릴 수 있다.

exception case를 거친 후에 _tmain의 switch case에서 break 를 하여 case를 이탈하는 이유는 exception_debug_event 함수에서 레지스터 값을 다시 쓸 필요가 없으므로 SetThreadContext 함수를 호출 할 필요가 없어지게 되기 때문이다.

OpenThread function

```
HANDLE WINAPI OpenThread (
    _In_ DWORD dwDesiredAccess, //access flags
    _In_ BOOL bInheritHandle, // handle inheritance options
    _In_ DWORD dwThreadId //Thread ID
);
```

GetThreadContext function

```
BOOL WINAPI GetThreadContext(
    _In_ HANDLE hThread, // Thread handle with context
    _Inout_ LPVOID lpContext //The structure address of the receiving context
);
```

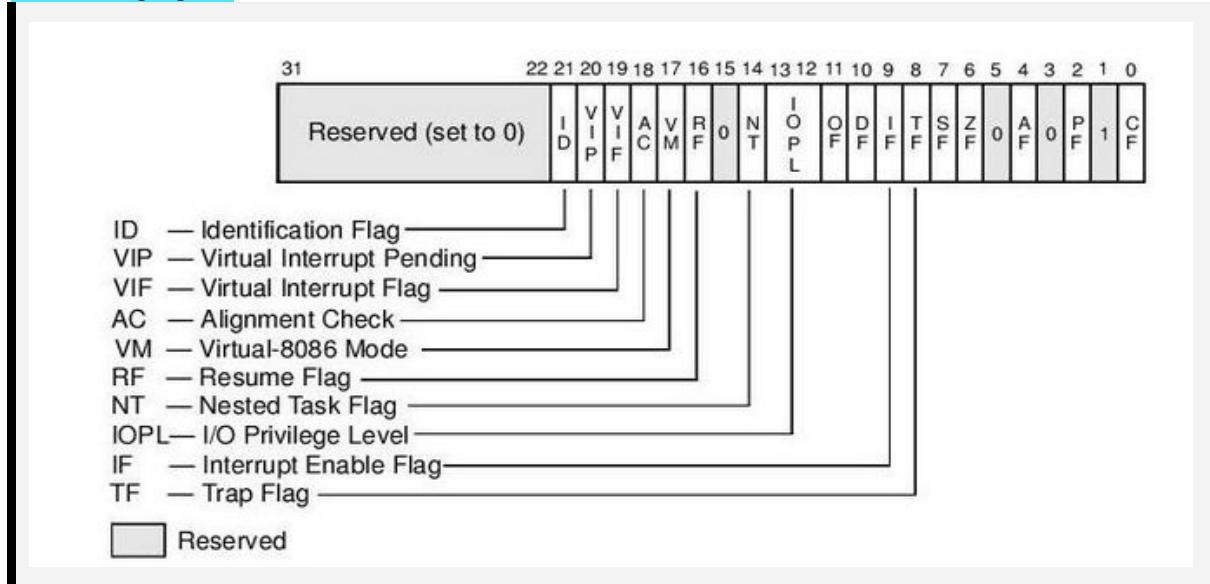
SetThreadContext

```
BOOL WINAPI SetThreadContext(
    _In_ HANDLE hThread, // Thread handle with context
    _In_ const CONTEXT *lpContext //Structure body address to store the context
);
```

2.3 Flag Register 란?

소스코드에서 GetThreadContext 와 함께 등장하는 플래그 레지스터(상태 레지스터), 플래그(Flag) 란 마이크로프로세서에서 다양한 산술 연산 결과의 상태를 알려주는 플래그 비트들이 모인 레지스터이다. 주로, 조건문과 같은 실행 순서의 분기에 사용된다. (상태 레지스터라고도 불리우며, 쉽게 말해 어떠한 상태를 깃발(Flag)을 흔들어 나타낸다고 보면 된다.) 밑의 그림은 플래그 레지스터의 구조를 나타낸 그림이며

EFLAGS : Flag register



플래그 레지스터(상태 레지스터)에는 다음과 같은 플래그들이 있다.

플래그 기호	이름	의미
Z	제로 플래그	연산 결과가 0일 경우에 참이 된다.
C	캐리 플래그	부호 없는 숫자의 연산 결과가 비트 범위를 넘어섰을 때 참이 된다.
A	보조 캐리 플래그	연산 결과 하위 니브(4bits)에서 비트 범위를 넘어섰을 때 참이 된다. 이진화 십진법(BCD) 연산에 사용된다.
V / O / W	오버플로 플래그	부호 있는 숫자의 연산 결과가 비트 범위를 넘어섰을 때 참이 된다.
N / S	네거티브 플래그, 사인플래그	연산 결과가 음수일 때 참이 된다.
I / E	인터럽트 플래그	이 플래그가 참일 경우에만 인터럽트 요구를 받아들인다. 일반적으로 관리자 모드에서만 값을 변경 할 수 있다.
P	패리티 플래그	연산 결과에서 1로된 비트의 수가 짝수일 경우 참이 된다.
D	디렉션 플래그	문자열 조작에서 참일 경우 주소 레지스터 값이 자동으로 감소하고, 거짓일 경우 자동으로 증가한다.
D / T	디버그 플래그, 트랩 플래그	참일 경우 한 명령이 실행할 때마다 인터럽트가 발생한다. 디버깅에 사용된다.

3. Version #2의 디버거에 여러분 만의 기능을 추가하시오.

3.1 추가기능 1 - 모든 EVENT CODE, STATUS 출력

```
printf(" EventCode: %d ", dbg_event.dwDebugEventCode);
fprintf(fp," EventCode: %d ", dbg_event.dwDebugEventCode);
SetConsoleTextAttribute( GetStdHandle(STD_OUTPUT_HANDLE) , 189 );
if(dbg_event.dwDebugEventCode==1){
    printf(" EXCEPTION_DEBUG_EVENT      ");
    fprintf(fp," EXCEPTION_DEBUG_EVENT      ");
}
else if(dbg_event.dwDebugEventCode==2){
    printf(" CREATE_THREAD_DEBUG_EVENT  ");
    fprintf(fp," CREATE_THREAD_DEBUG_EVENT  ");
}
else if(dbg_event.dwDebugEventCode==3){
    printf(" CREATE_PROCESS_DEBUG_EVENT ");
    fprintf(fp," CREATE_PROCESS_DEBUG_EVENT ");
}
else if(dbg_event.dwDebugEventCode==4){
    printf(" EXIT_THREAD_DEBUG_EVENT   ");
    fprintf(fp," EXIT_THREAD_DEBUG_EVENT   ");
}
else if(dbg_event.dwDebugEventCode==5){
    printf(" EXIT_PROCESS_DEBUG_EVENT  ");
    fprintf(fp," EXIT_PROCESS_DEBUG_EVENT  ");
}
else if(dbg_event.dwDebugEventCode==6){
    printf(" LOAD_DLL_DEBUG_EVENT      ");
    fprintf(fp," LOAD_DLL_DEBUG_EVENT      ");
}
else if(dbg_event.dwDebugEventCode==7){
    printf(" UNLOAD_DLL_DEBUG_EVENT    ");
    fprintf(fp," UNLOAD_DLL_DEBUG_EVENT    ");
}
else if(dbg_event.dwDebugEventCode==8){
    printf(" OUTPUT_DEBUG_STRING_EVENT ");
    fprintf(fp," OUTPUT_DEBUG_STRING_EVENT ");
}
else if(dbg_event.dwDebugEventCode==9){
    printf(" RIP_EVENT                  ");
    fprintf(fp," RIP_EVENT                  ");
}
```

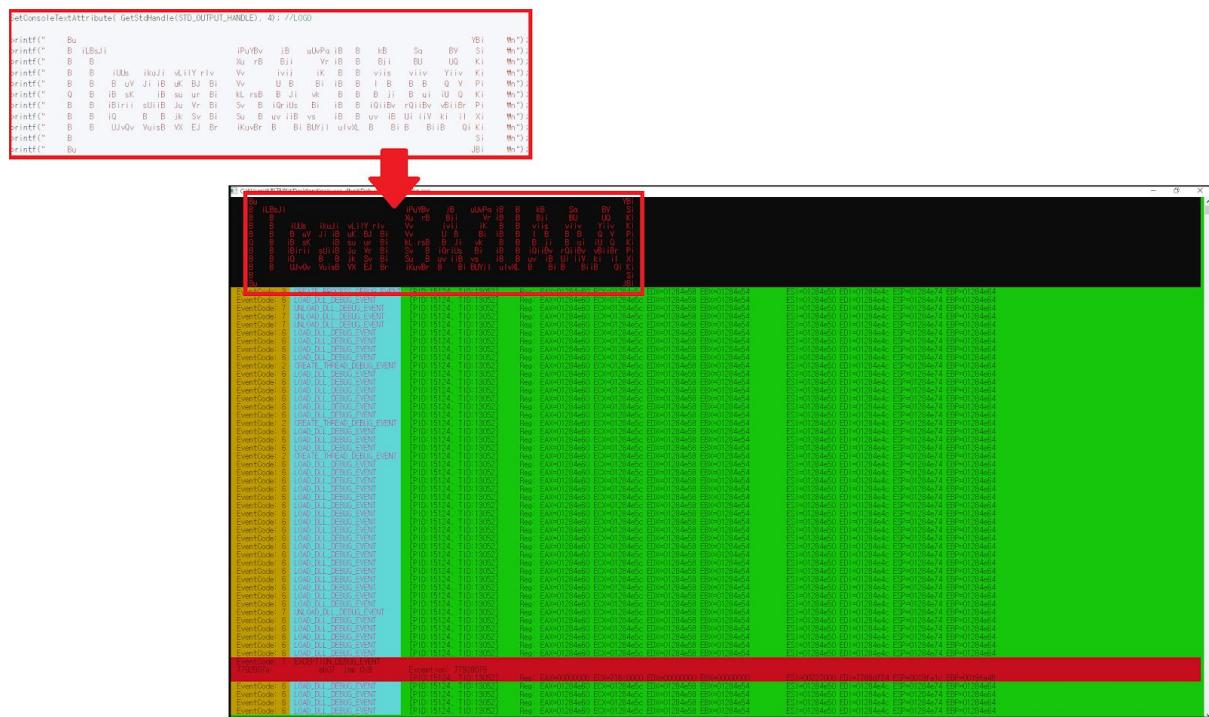
우리 조는 Version1 의 디버거 처럼 단순히 Exception case 만 보여주는 것이 아닌 모든 Event에 관한 정보를 보여주길 원했다. 그래서 dwDebugEventCode 를 통해 각각의 이벤트 코드를 출력함은 물론, 각각의 이벤트 별로 어떤 이벤트인지와 레지스트리 정보, PID, TID 등등을 보여주도록 코드를 생성하였다.

이 결과 아래와 같이 모든 EVENT에 대한 정보를 볼 수 있게 되었다.

EVENT CODE VALUES

이벤트 코드	이벤트 코드 값	유니언 u의 값
0x1	EXCEPTION_DEBUG_EVENT	u.Exception
0x2	CREATE_THREAD_DEBUG_EVENT	u.CreateThread
0x3	CREATE_PROCESS_DEBUG_EVENT	u.CreateProcessinfo
0x4	EXIT_THREAD_DEBUG_EVENT	u.ExitThread
0x5	EXIT_PROCESS_DEBUG_EVENT	u.ExitProcess
0x6	LOAD_DLL_DEBUG_EVENT	u.LoadDII
0x7	UNLOAD_DLL_DEBUG_EVENT	u.unloadDII
0x8	OUTPUT_DEBUG_STRING_EVENT	u.DebugString
0x9	RIP_EVENT	u.Ripinfo

3.2 추가기능 2 - 항목별 색 구분 및 로고 삽입



과제 이름부터 “Making Own your Debugger” 이다. 우리는 우리 조만의 상징 혹은 로고 등을 넣는 것이 좋다고 판단하여, 디버거 실행 전 위와 같이 글자를 활용한 그림 로고를 삽입하였다.

(위 글자의 경우 그림파일을 글자로 변환해 주는 프로그램인 Ascii Generator 2 를 사용하였다.)



※ <https://sourceforge.net/projects/ascgen2/>

또한 Windows.h 헤더파일에서 제공하는 콘솔 기능 중 하나인 SetConsoleTextAttribute 를 활용하여 콘솔창에 나타나는 바탕색과 글자 색 등을 보기 좋게 바꾸었다.

(Exception case의 경우 배경색이 붉은 색으로 표현하는 식이다.)

// 붉은색 바탕	SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 64);
EventCode: 6	LOAD_DLL_DEBUG_EVENT
EventCode: 1	EXCEPTION_DEBUG_EVENT
7792807a:	eb07 jmp 0x9
	Exception: 77928079
EventCode: 6	LOAD_DLL_DEBUG_EVENT
EventCode: 6	LOAD_DLL_DEBUG_EVENT

3.3 추가기능 3 - 중복없는 로그파일 저장

```
//*****File In&out control*****
time_t timer;
struct tm *t;

timer = time(NULL); // 현재 시각을 초 단위로 얻기

t = localtime(&timer); // 초 단위의 시간을 분리하여 구조체에 넣기

char file_name[256];
sprintf(file_name, "data_%d%d%d%d.txt", t->tm_year + 1900, t->tm_mon + 1, t->tm_mday, t->tm_hour, t->tm_min, t->tm_sec);
//현재시간을 변수로 받아 파일 이름에 사용
//현재_년,월,일,시,분,초
FILE *fp = fopen( file_name, "wt" );
if (fp == NULL) {
    fprintf(fp, "This is log file");
}
FILE *fpex = fopen("exception_temp.txt", "w+");
if (fpex == NULL) {
    fprintf(fpex,"Begin_to_Exception_case");
}
//*****
```



data_20186162291.txt	2018-06-16 오후 10:09	텍스트 문서	13KB
data_201861622049.txt	2018-06-16 오후 10:01	텍스트 문서	14KB
data_201861622221.txt	2018-06-16 오후 10:02	텍스트 문서	14KB
data_201861622313.txt	2018-06-16 오후 10:03	텍스트 문서	15KB
data_2018616215341.txt	2018-06-16 오후 9:53	텍스트 문서	0KB
data_2018616215537.txt	2018-06-16 오후 9:55	텍스트 문서	14KB
data_2018616215943.txt	2018-06-16 오후 9:59	텍스트 문서	14KB
exception_temp.txt	2018-06-16 오후 10:09	텍스트 문서	1KB
학점계산기.exe	2018-06-12 오후 7:30	응용 프로그램	212KB

특정 프로그램을 디버깅 할 때마다 계속해서 경로를 입력해 주고 이전 기록을 메모하려면
복사, 붙여넣기 등의 작업이 필요하다. 또한 로그파일이 없을 경우 이전 디버깅 기록과
대조하는 것 또한 쉽지 않을 것이다.

그러기에 위 소스코드와 같이 로그파일을 생성하는 코드를 넣어주었으며, 파일 이름은
생성일을 기준으로 년,월,일,시,분,초를 참조해 파일 이름 뒤에 붙여넣어 중복이 없게 만들었다.

또한 exception case의 경우 exception_temp.txt로 계속해서 덮어 씌워지는 일회성 파일이긴 하나,
exception case만을 따로 분류하여 저장하기 때문에 방금 작업한 프로세스 정보라면
로그파일을 일일이 뒤질 필요 없이 exception_temp.txt 파일만 열어봐도 된다.

위의 파일 목록을 캡쳐한 사진의 경우 테스트 프로그램 (학점계산기.exe)를 디버깅 했을 때
남는 로그 파일들이다.

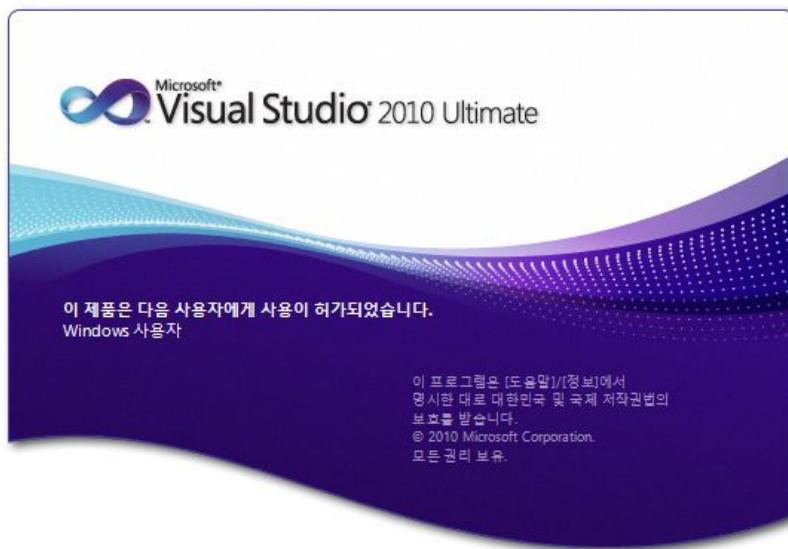
로그파일은 위의 사진과 같이 서상되며, Exception case의 경우엔 log 파일에 위, 아래에 느낌표들을 삽입하여 구분이 쉽게 저장되게 하였다.

```
printf(fp,"!!!!!!\n");
printf(fp,"!!!!!!\n");
printf(fp,"!!!!!!\n");
printf(fp,"!!!!!!\n");
printf(fp,"!!!!!!\n");
printf(fp,"!!!!!!\n");
exception_debug_event(&dbg_event); //exception 발생 시 log 파일에 위, 아래처럼 느낌표 삽입하여 구분이 쉽게함
fgets(buffer,sizeof(buffer), fpx);
fprintf(fp,buffer,sizeof(buffer));
fprintf(fp,"!!!!!!\n");
fprintf(fp,"!!!!!!\n");
fprintf(fp,"!!!!!!\n");
fprintf(fp,"!!!!!!\n");
fprintf(fp,"!!!!!!\n");
fprintf(fp,"!!!!!!\n");
```

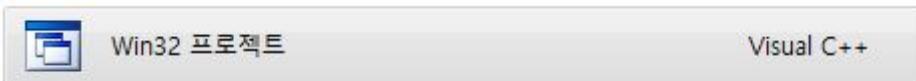
4. Version #2를 GUI로 구현하시오. (Option + 10점)

4.1 개발 환경

Compile & Build Environments : Microsoft Visual Studio 2010 Ultimate



사용 언어 : C language 기반 Win32API



4.2 GUI 화면 및 기능 설명

```

gazuaaa.dbg made by Team Gazuaaa
x

디버그 GAZUAAA!!!!
지금까지 분석한 파일의 경로
[empty]

208202501 1220800    01    22    181
 70   18      580    887    851
 18   18      18 80    016    8781
 10   18      81 8    8 15    08 01
 18  1807501  88 28    8 80    18 81
 15   18      58782807  8 8 81 01
 18   10      52    10 81    1808 81
 14  1872722  78    88 81    08 01

 728080    51    008052407  0    78    04    18    02
 0611  6      880    08 8    78    1487    808    2881
 88    18 80    58 0    74    01 8    08 02    8 14
 8     81 81    08 8    18    28 88    0 14    80 82
 10  10800  80 28    08 8    78 8 51    88 80    18 18
 81    80 48242502  88 8    18 85225284  182525081  88280088
 187  08 50    181 07    88 81 78    80 81    18 08    57
 88808481  78    88 887228087  8845081  81    8 44    55 8    75

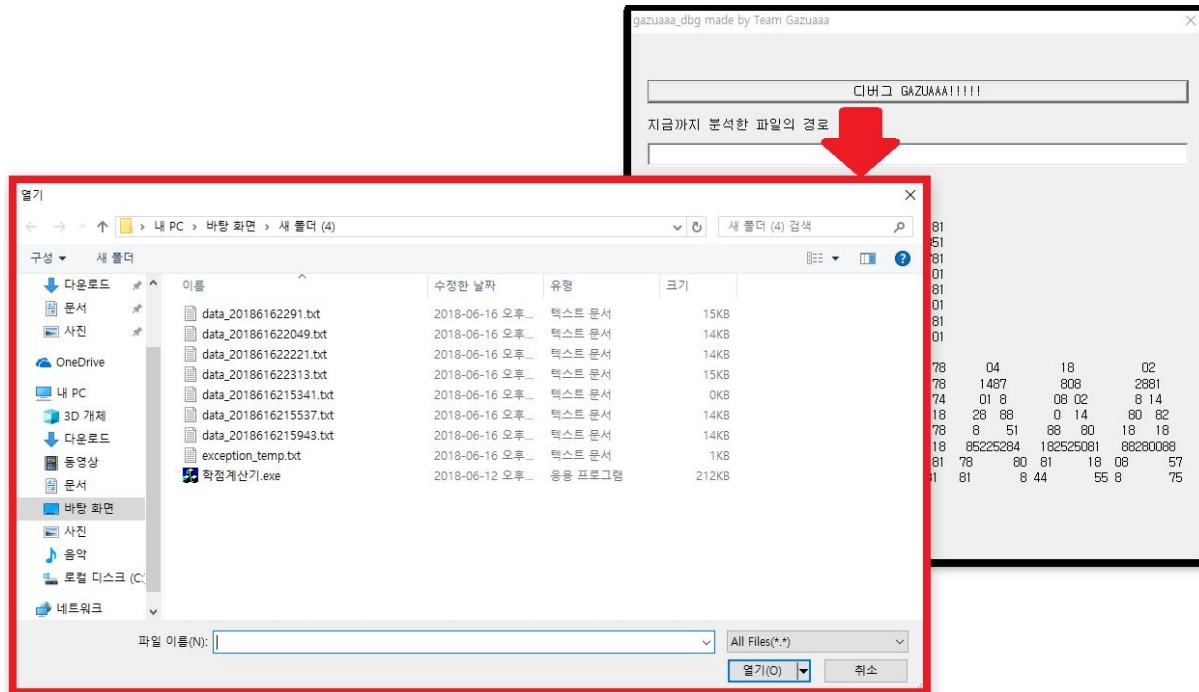
```

GUI 의 화면은 왼쪽 과 같다.

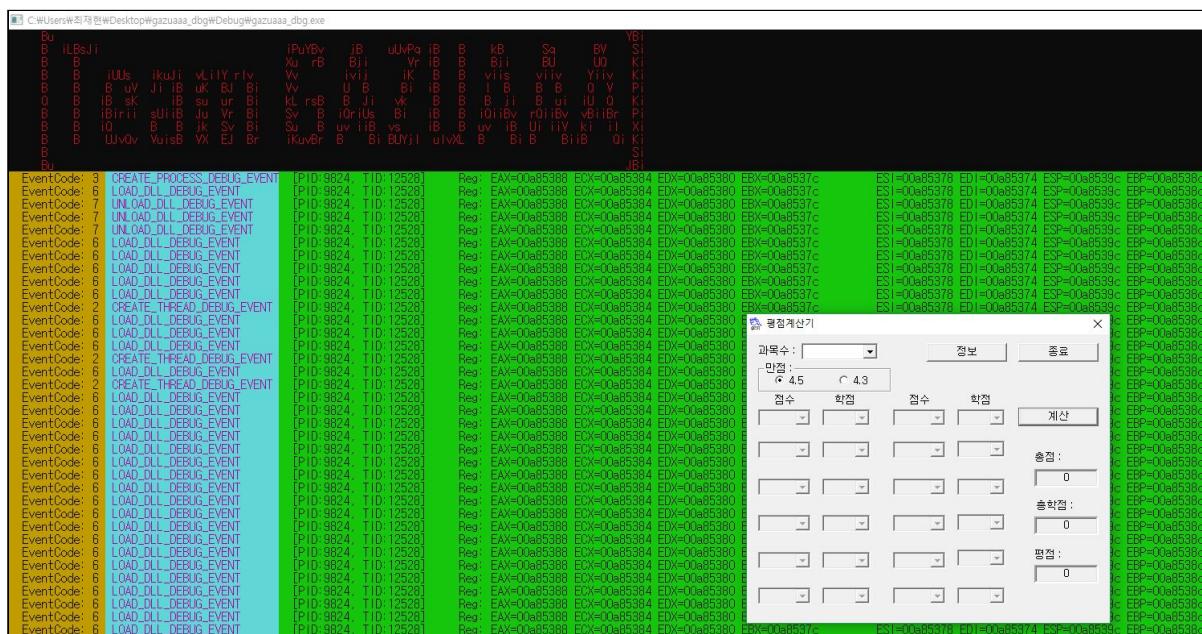
우선 우리가 GUI가 없어 가장 불편했던 점은 파일 디렉토리 기능이 없어 일일이 분석할 대상이 있는 경로를 입력해야 한다는 것 이였다.

그래서 우린 이 부분을 개선 해 보고자 하였다.

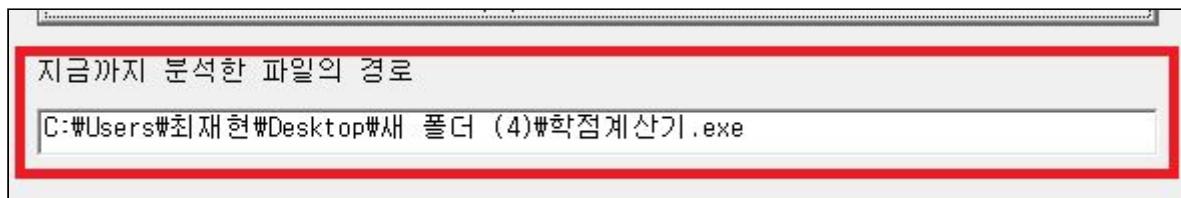
아래의 그림과 같이 디버그 GAZUAAA!!!! 를 클릭하게 되면,



그림과 같이 파일 탐색용 파일 다이얼로그가 출력되는 것을 알 수 있다.



테스트 프로그램인 '학점계산기.exe'를 다이얼로그에서 선택하게 되면 위와같이 디버깅 화면이 뜨고 디버깅을 시작하게 된다.



디버깅을 마치고 나서도 방금 내가 디버깅 했던 파일이 어느 경로에 있었는지 확인 할 수 있도록 에디트 박스를 만들었다.

4.3 Sour Code

gazaaaa_dbg.h

```
#ifndef EX_DIALOGAPP_RC_H
#define EX_DIALOGAPP_RC_H
//-----
// 대화상자 ID
#define IDD_MAIN_DIALOG      101
//-----
// 컨트롤들의 ID
#define IDC_PUSHBUTTON_OPENFILEDIALOG    201
#define IDC_LTEXT_FILETOBEOPENED        202
#define IDC_EDIT_FILETOBEOPENED         203
//-----
#endif
```

gazaaaa_dbg.cpp

```
//-----
// Header Files
#define _CRT_SECURE_NO_WARNINGS // fopen 보안 경고로 인한 컴파일 에러 방지
#include <windows.h>
#include <stdio.h>
#include "udis86.h" // 디스어셈블링을 위한 udis86.h
#include <time.h> // 로그파일 저장 시 변수로 사용하기 위한 time.h

#pragma comment(lib, "libudis86.lib") // udis86의 라이브러리 파일

#include "gazaaaa_dbg.h"

//-----
// Macro Definitions

//-----
// Global Variables

HWND hButtonOpenFileDialog;           // 파일열기 대화상을 실행하기 위한 버튼의 핸들
HWND hEditFileToBeOpened;             // 파일의 경로와 이름을 가져오는 에디트 컨트롤의 핸들

OPENFILENAME OFN;                   // 파일열기 대화상을 초기화하기 위한 변수
const UINT nFileNameMaxLen = 512;    // 다음 줄에 정의하는 szFileName 문자열의 최대 길이
WCHAR szFileName[nFileNameMaxLen]; // 파일의 경로 및 이름을 복사하기 위한 문자열

// Functions

BOOL CALLBACK MainDlgProc(HWND hDlg, UINT iMessage, WPARAM wParam, LPARAM lParam);
```

```

//=====

int sj_disassembler(unsigned char *buff, char *out, int size)
{ //기계어를 분해하는 역할을 담당한다. 여기서 udis86의 기능이 사용되게 된다.

    ud_t ud_obj;
    ud_init(&ud_obj);
    ud_set_input_buffer(&ud_obj, buff, 32);
    ud_set_mode(&ud_obj, 32);
    ud_set_syntax(&ud_obj, UD_SYN_INTEL);

    if(ud_disassemble(&ud_obj)){
        sprintf_s(out, size, "%14s %s", ud_insn_hex(&ud_obj), ud_insn_asm(&ud_obj));
    }else{
        return -1;
    }
    return (int)ud_insn_len(&ud_obj);
}

int exception_debug_event(DEBUG_EVENT *pde)
{ //exception이 발생하면 실행되며, 밑의 함수들을 호출
/* Open Process,ReadProcessMemory, OpenThread, GetThreadContext, SetThreadContext */

    // 이 함수와 WriteProcessMemory number 는 다른 프로세스에 access 하는데 필요

//+++++
    FILE *fp = fopen("exception_temp.txt", "w+");
    if (fp == NULL) {
        fprintf(fp, "This is log file for exception case\n");
    }
//+++++
    DWORD dwReadBytes;

    HANDLE process_hander= OpenProcess( // 프로세스 열기 , 헨들러에 할당
        PROCESS_VM_WRITE | PROCESS_VM_READ | PROCESS_VM_OPERATION,
        FALSE, pde->dwProcessId);

    if(!process_hander) return -1;

    HANDLE thread_handler= OpenThread(ThreadGetContext | ThreadSetContext, FALSE, pde->dwThreadId);
    //Open Thread : Register 정보 읽기위해 필요한 function

    if(!thread_handler) return -1;

    CONTEXT ctx;
    ctx.ContextFlags= CONTEXT_ALL;
    GetThreadContext(thread_handler, &ctx);
    //GetThreadContext 통해 레지스터를 Read 할 수 있다.
    char asm_string[256];
    unsigned char asm_code[32];

    ReadProcessMemory(process_hander, (VOID *)ctx.Eip, asm_code, 32, &dwReadBytes);
}

```

```
//exception_debug_event 함수에서 exception이 발생할때 실행 된 명령을 얻기 위해 필요
```

```
if(sj_disassembler(asm_code, asm_string, sizeof(asm_string)) == -1)
asm_string[0] = '\0';

// 블은색 바탕
SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 64);
printf(" EventCode: %d ", pde->dwDebugEventCode);
fprintf(fp," EventCode: %d ", pde->dwDebugEventCode);
if(pde->dwDebugEventCode==1){
    printf(" EXCEPTION_DEBUG_EVENT  ");
    fprintf(fp," EXCEPTION_DEBUG_EVENT  ");
}
else if(pde->dwDebugEventCode==2){
    printf(" CREATE_THREAD_DEBUG_EVENT ");
    fprintf(fp," CREATE_THREAD_DEBUG_EVENT ");
}
else if(pde->dwDebugEventCode==3){
    printf(" CREATE_PROCESS_DEBUG_EVENT");
    fprintf(fp," CREATE_PROCESS_DEBUG_EVENT");
}
else if(pde->dwDebugEventCode==4){
    printf(" EXIT_THREAD_DEBUG_EVENT ");
    fprintf(fp," EXIT_THREAD_DEBUG_EVENT ");
}
else if(pde->dwDebugEventCode==5){
    printf(" EXIT_PROCESS_DEBUG_EVENT ");
    fprintf(fp," EXIT_PROCESS_DEBUG_EVENT ");
}
else if(pde->dwDebugEventCode==6){
    printf(" LOAD_DLL_DEBUG_EVENT  ");
    fprintf(fp," LOAD_DLL_DEBUG_EVENT  ");
}
else if(pde->dwDebugEventCode==7){
    printf(" UNLOAD_DLL_DEBUG_EVENT  ");
    fprintf(fp," UNLOAD_DLL_DEBUG_EVENT  ");
}
else if(pde->dwDebugEventCode==8){
    printf(" OUTPUT_DEBUG_STRING_EVENT");
    fprintf(fp," OUTPUT_DEBUG_STRING_EVENT");
}
else if(pde->dwDebugEventCode==9){
    printf(" RIP_EVENT          ");
    fprintf(fp," RIP_EVENT          ");
}
printf("\n");

printf(" %08x: %s", ctx.Eip, asm_string);
fprintf(fp, " %08x: %s", ctx.Eip, asm_string);
printf("     Exception: %08x \n",pde->u.Exception.ExceptionRecord.ExceptionAddress);
fprintf(fp, "     Exception: %08x ",pde->u.Exception.ExceptionRecord.ExceptionAddress);
printf("             [PID:%d, TID:%d] ",
```

```

pde->dwProcessId, pde->dwThreadId);
fprintf(fp, " [PID:%d, TID:%d] ",
pde->dwProcessId, pde->dwThreadId);
printf(" Reg: EAX=%08x ECX=%08x EDX=%08x EBX=%08x",
ctx.Eax, ctx.Ecx, ctx.Edx, ctx.Ebx);
    fprintf(fp, " Reg: EAX=%08x ECX=%08x EDX=%08x EBX=%08x",
ctx.Eax, ctx.Ecx, ctx.Edx, ctx.Ebx);
printf(" ESI=%08x EDI=%08x ESP=%08x EBP=%08x\n",
ctx.Esi, ctx.Edi, ctx.Esp, ctx.Ebp);
    fprintf(fp, " ESI=%08x EDI=%08x ESP=%08x EBP=%08x\n",
ctx.Esi, ctx.Edi, ctx.Esp, ctx.Ebp);
SetConsoleTextAttribute( GetStdHandle(STD_OUTPUT_HANDLE), 160); //원래 색으로 되돌리기
SetThreadContext(thread_handler, &ctx);

//SetThreadContext를 통해 레지스터를 Write 할 수 있다.
CloseHandle(thread_handler);
CloseHandle(process_hander);

//>>>>>>>>>>>>>>>>>>>
fclose(fp);
//>>>>>>>>>>>>>>>>>>>
return 0;
}// End of exception_debug_event

//-----
// Main Function

int APIENTRY WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,LPSTR lpszCmdParam,int nCmdShow)
{
    DialogBox(hInstance, MAKEINTRESOURCE(IDD_MAIN_DIALOG), HWND_DESKTOP, MainDlgProc);
    return 0;
}

BOOL CALLBACK MainDlgProc(HWND hDlg, UINT iMessage, WPARAM wParam, LPARAM lParam)
{
    char buffer[500000]; //Exception case 정보 저장 용 buffer
//+++++
CONTEXT ctx;
ctx.ContextFlags= CONTEXT_ALL;

STARTUPINFO startup_info;
PROCESS_INFORMATION process_info;
    // 생성될 프로세스의 정보가 저장될 구조체를 선언
memset(&process_info, 0, sizeof(process_info));
    // startup_info 구조체의 정보를 명시
memset(&startup_info, 0, sizeof(startup_info));
    startup_info.cb= sizeof(STARTUPINFO);
    int process_counter = 0;//프로세스 카운터 선언

//+++++
switch(iMessage)

```

```

{
case WM_INITDIALOG:
    hButtonOpenFileDialog = GetDlgItem(hDlg, IDC_PUSHBUTTON_OPENFILEDIALOG);
    hEditFileToBeOpened = GetDlgItem(hDlg, IDC_EDIT_FILETOBEOPENED);
    return TRUE;

case WM_COMMAND:
    switch (LOWORD(wParam))
    {
        case IDC_PUSHBUTTON_OPENFILEDIALOG:
            //-----
            // OPENFILENAME형 변수의 멤버들의 값을 설정
            memset(&OFN, 0, sizeof(OPENFILENAME));
            OFN.lStructSize = sizeof(OPENFILENAME);
            OFN.hwndOwner = hDlg;
            OFN.lpstrFilter = L"All Files(*.*)\0*.*\0";
            OFN.lpstrFile = szFileName;
            OFN.nMaxFile = nFileNameMaxLen;
            //-----
            // 파일열기 대화상자를 열고, 선택된 파일의 이름을 에디트 박스로 복사
            if (0 != GetOpenFileName(&OFN))
            {
                SetWindowText(hEditFileToBeOpened, OFN.lpstrFile);

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
AllocConsole(); //콘솔창 open
freopen("CONIN$", "r", stdin);
freopen("CONOUT$", "w", stdout);
freopen("CONOUT$", "w", stderr);

            //++++++File In&out control+++++++
time_t timer;
struct tm *t;

timer = time(NULL); // 현재 시각을 초 단위로 얻기

            t = localtime(&timer); // 초 단위의 시간을 분리하여 구조체에 넣기

            char file_name[256];
sprintf(file_name, "data_%d%d%d%d%d.txt", t->tm_year + 1900, t->tm_mon +
1, t->tm_mday, t->tm_hour, t->tm_min, t->tm_sec);
            //현재시간을 변수로 받아 파일 이름에 사용
            //현재 년,월,일,시,분,초
            FILE *fp = fopen( file_name, "wt" );
            if (fp == NULL) {
                fprintf(fp, "This is log file");
            }
            FILE *fpex = fopen("exception_temp.txt", "w+");
            if (fpex == NULL) {
                fprintf(fpex, "Begin_to_Exception_case");
            }
            //+++++++
}

```

```

SetConsoleTextAttribute( GetStdHandle(STD_OUTPUT_HANDLE), 4); //LOGO

printf(" Bu YBi \n");
printf(" B iLBsJi iPuYBv jB uUvPq iB B kB Sq BV Si \n");
printf(" B B Xu rB Bj Vr iB B Bj BU UQ Ki \n");
printf(" B B iUUUs ikuJi vLiY rIv Vv ivij iK B B viis viiv Yiiv Ki \n");
printf(" B B B uV JiB uK BJ Bi Vv U B Bi iB B I B B B Q V Pi \n");
printf(" Q B iB sK iB su ur Bi kLrsB B Ji vk B B B ji B ui iU Q Ki \n");
printf(" B B iBirii sUiB Ju Vr Bi Sv B iQriUs Bi iB B iQiiBv rQiiBv vBiiBr Pi \n");
printf(" B B iQ B B jk Sv Bi Su B uv iiB vs iB B uv iB Ui iiV ki il Xi \n");
printf(" B B UJvQv VuisB VX EJ Br iKuvBr B Bi BUYJl ulvXL B Bi B BiiB Qi Ki \n");
printf(" B Si \n");
printf(" Bu JBi \n");

BOOL r = CreateProcess(
    NULL, OFN.lpstrFile, NULL, NULL, FALSE,
    NORMAL_PRIORITY_CLASS | CREATE_SUSPENDED | DEBUG_PROCESS,
    NULL, NULL, &startup_info, &process_info);
if(!r)
    return -1;

ResumeThread(process_info.hThread);

do{
DEBUG_EVENT dbg_event;
if(!WaitForDebugEvent(&dbg_event, INFINITE)) break;
//EXCEPTION_DEBUG_EVENT 발생 시 디버거에서 직접 예외를 처리
DWORD dwContinueStatus = DBG_CONTINUE;

switch(dbg_event.dwDebugEventCode)
{
case CREATE_PROCESS_DEBUG_EVENT://|버그 이벤트 유형으로 Switch Case 를 나눔
    SetConsoleTextAttribute( GetStdHandle(STD_OUTPUT_HANDLE), 96);
    printf(" EventCode: %d ", dbg_event.dwDebugEventCode);
    fprintf(fp," EventCode: %d ", dbg_event.dwDebugEventCode);
    SetConsoleTextAttribute( GetStdHandle(STD_OUTPUT_HANDLE), 189);

    if(dbg_event.dwDebugEventCode==1){
        printf(" EXCEPTION_DEBUG_EVENT ");
        fprintf(fp," EXCEPTION_DEBUG_EVENT ");
    }
    else if(dbg_event.dwDebugEventCode==2){
        printf(" CREATE_THREAD_DEBUG_EVENT ");
        fprintf(fp," CREATE_THREAD_DEBUG_EVENT ");
    }
    else if(dbg_event.dwDebugEventCode==3){
        printf(" CREATE_PROCESS_DEBUG_EVENT");
        fprintf(fp," CREATE_PROCESS_DEBUG_EVENT");
    }
}
}

```

```

    }

    else if(dbg_event.dwDebugEventCode==4){
        printf(" EXIT_THREAD_DEBUG_EVENT ");
        fprintf(fp," EXIT_THREAD_DEBUG_EVENT ");
    }

    else if(dbg_event.dwDebugEventCode==5){
        printf(" EXIT_PROCESS_DEBUG_EVENT ");
        fprintf(fp," EXIT_PROCESS_DEBUG_EVENT ");
    }

    else if(dbg_event.dwDebugEventCode==6){
        printf(" LOAD_DLL_DEBUG_EVENT ");
        fprintf(fp," LOAD_DLL_DEBUG_EVENT ");
    }

    else if(dbg_event.dwDebugEventCode==7){
        printf(" UNLOAD_DLL_DEBUG_EVENT ");
        fprintf(fp," UNLOAD_DLL_DEBUG_EVENT ");
    }

    else if(dbg_event.dwDebugEventCode==8){
        printf(" OUTPUT_DEBUG_STRING_EVENT ");
        fprintf(fp," OUTPUT_DEBUG_STRING_EVENT ");
    }

    else if(dbg_event.dwDebugEventCode==9){
        printf(" RIP_EVENT ");
        fprintf(fp," RIP_EVENT ");
    }

}

SetConsoleTextAttribute( GetStdHandle(STD_OUTPUT_HANDLE), 160);
printf(" [PID:%d, TID:%d] ", process_info.dwProcessId,process_info.dwThreadId);
fprintf(fp," [PID:%d, TID:%d] ", process_info.dwProcessId,process_info.dwThreadId);
printf(" Reg: EAX=%08x ECX=%08x EDX=%08x EBX=%08x",&ctx.Eax,&ctx.Ecx,&ctx.Edx,&ctx.Ebx);
fprintf(fp," Reg: EAX=%08x ECX=%08x EDX=%08x EBX=%08x",&ctx.Eax,&ctx.Ecx,&ctx.Edx,&ctx.Ebx);
printf(" ESI=%08x EDI=%08x ESP=%08x EBP=%08x\n",&ctx.Esi,&ctx.Edi,&ctx.Esp,&ctx.Ebp);
fprintf(fp," ESI=%08x EDI=%08x ESP=%08x EBP=%08x\n",&ctx.Esi,&ctx.Edi,&ctx.Esp,&ctx.Ebp);
process_counter++;
break;

case EXIT_PROCESS_DEBUG_EVENT:
    SetConsoleTextAttribute( GetStdHandle(STD_OUTPUT_HANDLE), 96);
    printf(" EventCode: %d ", dbg_event.dwDebugEventCode);
    fprintf(fp," EventCode: %d ", dbg_event.dwDebugEventCode);
    SetConsoleTextAttribute( GetStdHandle(STD_OUTPUT_HANDLE), 189);
    if(dbg_event.dwDebugEventCode==1){
        printf(" EXCEPTION_DEBUG_EVENT ");
        fprintf(fp," EXCEPTION_DEBUG_EVENT ");
    }

    else if(dbg_event.dwDebugEventCode==2){
        printf(" CREATE_THREAD_DEBUG_EVENT ");
        fprintf(fp," CREATE_THREAD_DEBUG_EVENT ");
    }

    else if(dbg_event.dwDebugEventCode==3){
        printf(" CREATE_PROCESS_DEBUG_EVENT");
        fprintf(fp," CREATE_PROCESS_DEBUG_EVENT");
    }
}

```

```

    }

    else if(dbg_event.dwDebugEventCode==4){
        printf(" EXIT_THREAD_DEBUG_EVENT ");
        fprintf(fp," EXIT_THREAD_DEBUG_EVENT ");
    }

    else if(dbg_event.dwDebugEventCode==5){
        printf(" EXIT_PROCESS_DEBUG_EVENT ");
        fprintf(fp," EXIT_PROCESS_DEBUG_EVENT ");
    }

    else if(dbg_event.dwDebugEventCode==6){
        printf(" LOAD_DLL_DEBUG_EVENT ");
        fprintf(fp," LOAD_DLL_DEBUG_EVENT ");
    }

    else if(dbg_event.dwDebugEventCode==7){
        printf(" UNLOAD_DLL_DEBUG_EVENT ");
        fprintf(fp," UNLOAD_DLL_DEBUG_EVENT ");
    }

    else if(dbg_event.dwDebugEventCode==8){
        printf(" OUTPUT_DEBUG_STRING_EVENT ");
        fprintf(fp," OUTPUT_DEBUG_STRING_EVENT ");
    }

    else if(dbg_event.dwDebugEventCode==9){
        printf(" RIP_EVENT ");
        fprintf(fp," RIP_EVENT ");
    }

}

SetConsoleTextAttribute( GetStdHandle(STD_OUTPUT_HANDLE), 160);
printf(" [PID:%d, TID:%d] ", process_info.dwProcessId,process_info.dwThreadId);
fprintf(fp," [PID:%d, TID:%d] ", process_info.dwProcessId,process_info.dwThreadId);
printf(" Reg: EAX=%08x ECX=%08x EDX=%08x EBX=%08x",&ctx.Eax,&ctx.Ecx,&ctx.Edx,&ctx.Ebx);
fprintf(fp," Reg: EAX=%08x ECX=%08x EDX=%08x EBX=%08x",&ctx.Eax,&ctx.Ecx,&ctx.Edx,&ctx.Ebx);
printf(" ESI=%08x EDI=%08x ESP=%08x EBP=%08x\n",&ctx.Esi,&ctx.Edi,&ctx.Esp,&ctx.Ebp);
fprintf(fp," ESI=%08x EDI=%08x ESP=%08x EBP=%08x\n",&ctx.Esi,&ctx.Edi,&ctx.Esp,&ctx.Ebp);
process_counter--;
break;

case EXCEPTION_DEBUG_EVENT:
    if(dbg_event.u.Exception.ExceptionRecord.ExceptionCode !=
        EXCEPTION_BREAKPOINT)
    {
        dwContinueStatus = DBG_EXCEPTION_NOT_HANDLED;
    }

    fprintf(fp,"!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
    fprintf(fp,"!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
    fprintf(fp,"!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
    fprintf(fp,"!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
    fprintf(fp,"!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
    exception_debug_event(&dbg_event);

//exception 발생 시 log 파일에 위,아래처럼 느낌표 삽입하여 구분이 쉽게함
    fgets(buffer,sizeof(buffer),fpex);
    fprintf(fp,buffer,sizeof(buffer));
    fprintf(fp,"!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
}

```

```

        fprintf(fp,"!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
        fprintf(fp,"!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
        fprintf(fp,"!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
        fprintf(fp,"!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
        //execption_debug_event 함수에서 레지스터 값을 다시 쓸 필요가 없으므로
        // SetThreadContext 함수를 호출 할 필요가 없어지게 된다.

    break;

default :
    SetConsoleTextAttribute( GetStdHandle(STD_OUTPUT_HANDLE), 96);
    printf(" EventCode: %d ", dbg_event.dwDebugEventCode);
    fprintf(fp," EventCode: %d ", dbg_event.dwDebugEventCode);
    SetConsoleTextAttribute( GetStdHandle(STD_OUTPUT_HANDLE), 189);

    if(dbg_event.dwDebugEventCode==1){
        printf(" EXCEPTION_DEBUG_EVENT  ");
        fprintf(fp," EXCEPTION_DEBUG_EVENT  ");
    }
    else if(dbg_event.dwDebugEventCode==2){
        printf(" CREATE_THREAD_DEBUG_EVENT ");
        fprintf(fp," CREATE_THREAD_DEBUG_EVENT ");
    }
    else if(dbg_event.dwDebugEventCode==3){
        printf(" CREATE_PROCESS_DEBUG_EVENT");
        fprintf(fp," CREATE_PROCESS_DEBUG_EVENT");
    }
    else if(dbg_event.dwDebugEventCode==4){
        printf(" EXIT_THREAD_DEBUG_EVENT ");
        fprintf(fp," EXIT_THREAD_DEBUG_EVENT ");
    }
    else if(dbg_event.dwDebugEventCode==5){
        printf(" EXIT_PROCESS_DEBUG_EVENT ");
        fprintf(fp," EXIT_PROCESS_DEBUG_EVENT ");
    }
    else if(dbg_event.dwDebugEventCode==6){
        printf(" LOAD_DLL_DEBUG_EVENT  ");
        fprintf(fp," LOAD_DLL_DEBUG_EVENT  ");
    }
    else if(dbg_event.dwDebugEventCode==7){
        printf(" UNLOAD_DLL_DEBUG_EVENT  ");
        fprintf(fp," UNLOAD_DLL_DEBUG_EVENT  ");
    }
    else if(dbg_event.dwDebugEventCode==8){
        printf(" OUTPUT_DEBUG_STRING_EVENT");
        fprintf(fp," OUTPUT_DEBUG_STRING_EVENT");
    }
    else if(dbg_event.dwDebugEventCode==9){
        printf(" RIP_EVENT          ");
        fprintf(fp," RIP_EVENT          ");
    }
}

```

```

SetConsoleTextAttribute( GetStdHandle(STD_OUTPUT_HANDLE), 160);
printf(" [PID:%d, TID:%d] ", process_info.dwProcessId,process_info.dwThreadId);
fprintf(fp," [PID:%d, TID:%d] ", process_info.dwProcessId,process_info.dwThreadId);
printf(" Reg: EAX=%08x ECX=%08x EDX=%08x EBX=%08x",&ctx.Eax,&ctx.Ecx,&ctx.Edx,&ctx.Ebx);
fprintf(fp," Reg: EAX=%08x ECX=%08x EDX=%08x EBX=%08x",&ctx.Eax,&ctx.Ecx,&ctx.Edx,&ctx.Ebx);
printf(" ESI=%08x EDI=%08x ESP=%08x EBP=%08x\n",&ctx.Esi,&ctx.Edi,&ctx.Esp,&ctx.Ebp);
fprintf(fp," ESI=%08x EDI=%08x ESP=%08x EBP=%08x\n",&ctx.Esi,&ctx.Edi,&ctx.Esp,&ctx.Ebp);
}

ContinueDebugEvent(
    dbg_event.dwProcessId, dbg_event.dwThreadId, dwContinueStatus);

}while(process_counter > 0);
CloseHandle(process_info.hThread); //Thread 종료
CloseHandle(process_info.hProcess); // Process 종료

fclose(fpex);
fclose(fp);

FreeConsole(); // 콘솔해제
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
}
return TRUE;

}

return FALSE;

case WM_CLOSE:
EndDialog(hDlg, IDOK);
return TRUE;
}

return FALSE;
}

```

5. References

- ❖ <http://udis86.sourceforge.net/>
- ❖ <https://espada4897.wordpress.com/2014/11/09/win32-프로그램-exe-외부-프로그램에서-호출/>
- ❖ <http://kikillers.tistory.com/2>
- ❖ <http://tapito.tistory.com/389>
- ❖ <http://zapiro.tistory.com/entry/80x86-Assembly>
- ❖ <http://blog.eairship.kr/186>
- ❖ <http://sarghis.com/blog/357/>
- ❖ <http://p3ace.tistory.com/36>
- ❖ https://ko.wikipedia.org/wiki/상태_레지스터
- ❖ <http://jeep-shoes.tistory.com/45>
- ❖ <http://wowan.tistory.com/4>
- ❖ <http://blog.naver.com/gloryo/110135646658>
- ❖ <http://ehclub.co.kr/1300>
- ❖ [https://msdn.microsoft.com/ko-kr/library/microsoft.win32.filedialog\(v=vs.110\).aspx](https://msdn.microsoft.com/ko-kr/library/microsoft.win32.filedialog(v=vs.110).aspx)
- ❖ <http://www.soulfree.net/115>