PE parser, Debugger 개발 보고

스마트보안전공 202334843 이동하

Windows API 에 대한 이해도가 없는 상태로 시작하였습니다.
따라서 비효율적인 요소가 존재하더라도, NT structure와 API 관련 공부를 바탕으로 제작하였습니다.

+특정 코드에서는 api를 사용하는 것이 적합하지만, API 이해도 이슈로 직접 구현한 부분이 있을 수 있습니다.
(발표 이후에도 이어서 개선 해볼 예정입니다)

요구사항 :

- PE parser
- Disassemble
- Memory View
- BreakPoint

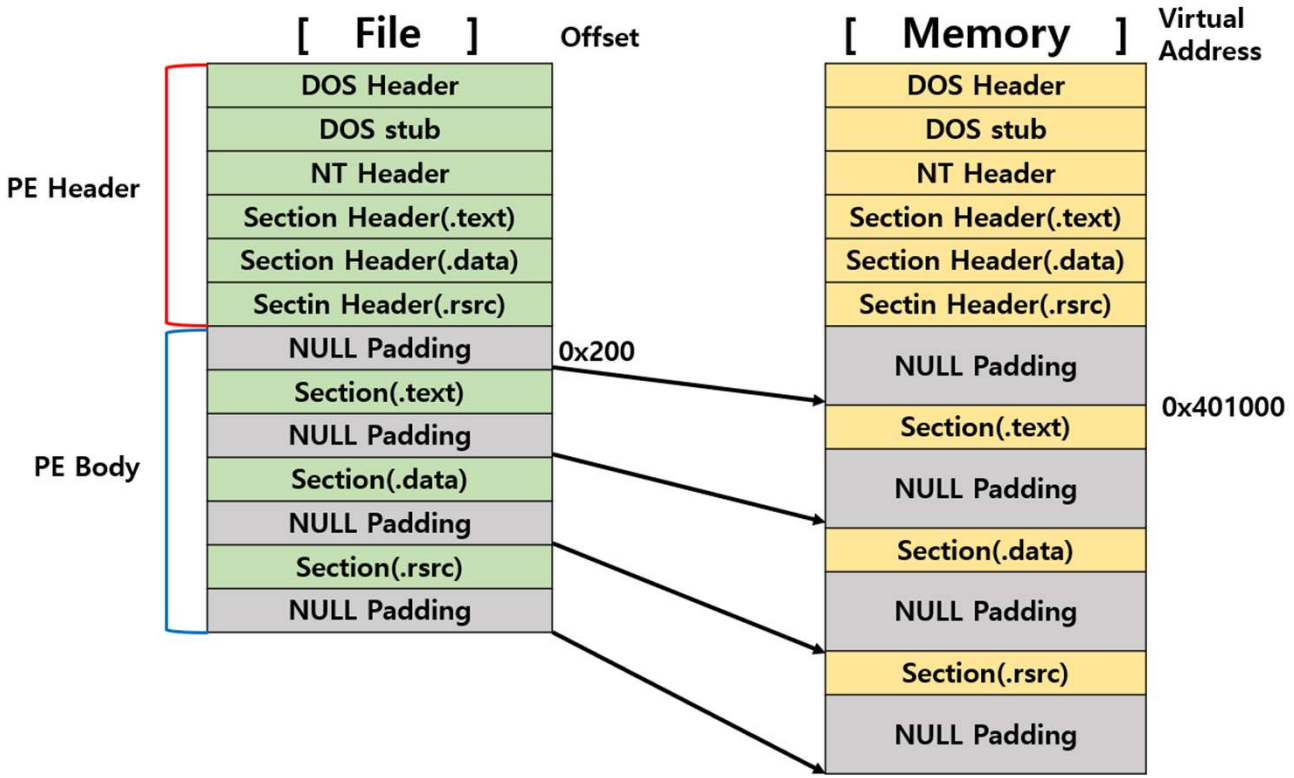
요구사항 :

- PE parser
- Disassemble
- Memory View
- BreakPoint

Plan :

- Windows Structure 를 공부하여 pe.h 제작
- Disassemble lib으로 capstone.h을 사용
- Debugee Process create, virtual memory R/W 용 exec.h
- Memory write로 int3를 injection

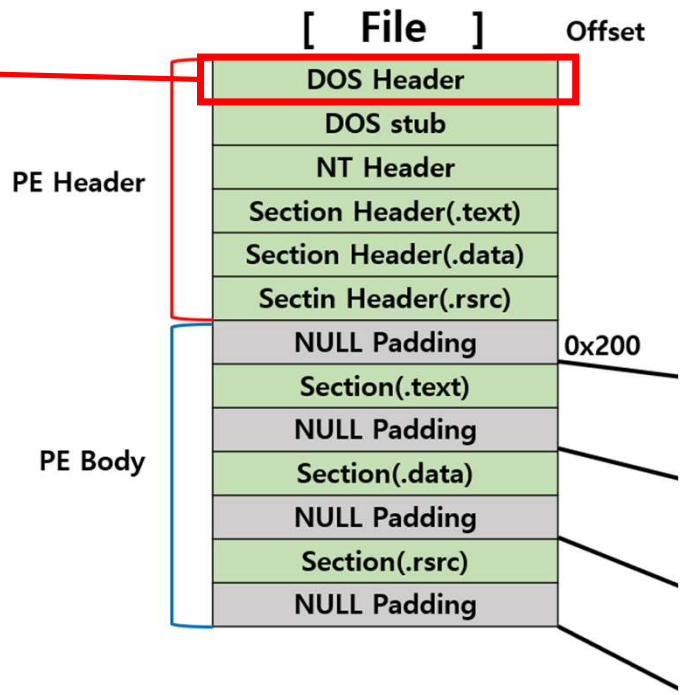
- **PE parser**



Ref : https://learn.microsoft.com/en-us/windows/win32/debug/pe-format

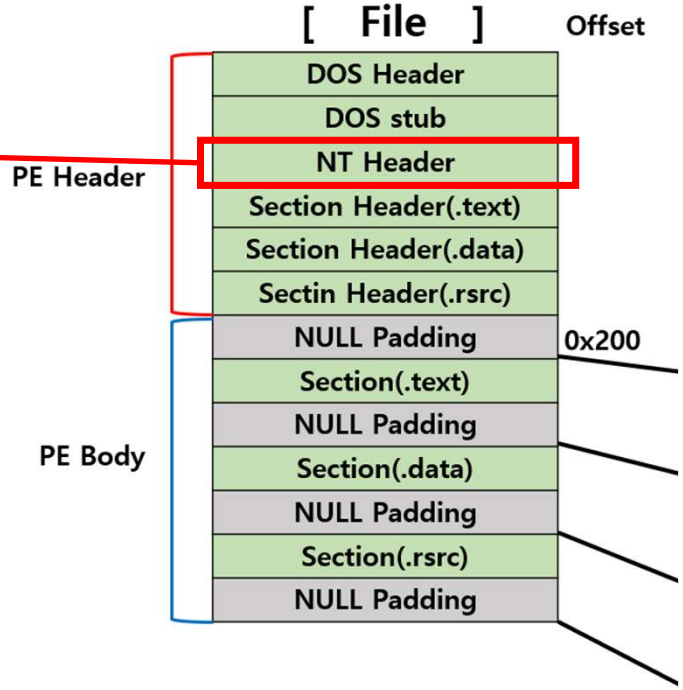
IN <winnt.h>

```
typedef struct _IMAGE_DOS_HEADER {      // DOS .EXE header
    WORD   e_magic;                     // Magic number
    WORD   e_cblp;                      // Bytes on last page of file
    WORD   e_cp;                        // Pages in file
    WORD   e_crlc;                      // Relocations
    WORD   e_cparhdr;                   // Size of header in paragraphs
    WORD   e_minalloc;                  // Minimum extra paragraphs needed
    WORD   e_maxalloc;                  // Maximum extra paragraphs needed
    WORD   e_ss;                        // Initial (relative) SS value
    WORD   e_sp;                        // Initial SP value
    WORD   e_csum;                      // Checksum
    WORD   e_ip;                        // Initial IP value
    WORD   e_cs;                        // Initial (relative) CS value
    WORD   e_lfarlc;                    // File address of relocation table
    WORD   e_ovno;                      // Overlay number
    WORD   e_res[4];                    // Reserved words
    WORD   e_oemid;                     // OEM identifier (for e_oeminfo)
    WORD   e_oeminfo;                   // OEM information; e_oemid specific
    WORD   e_res2[10];                  // Reserved words
    LONG   e_lfanew;                    // File address of new exe header
  } IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
```



PIMAGE_NT_HEADERS

```
#ifdef _WIN64
typedef IMAGE_NT_HEADERS64                  IMAGE_NT_HEADERS;
typedef PIMAGE_NT_HEADERS64                 PIMAGE_NT_HEADERS;
#else
typedef IMAGE_NT_HEADERS32                  IMAGE_NT_HEADERS;
typedef PIMAGE_NT_HEADERS32                 PIMAGE_NT_HEADERS;
#endif
```

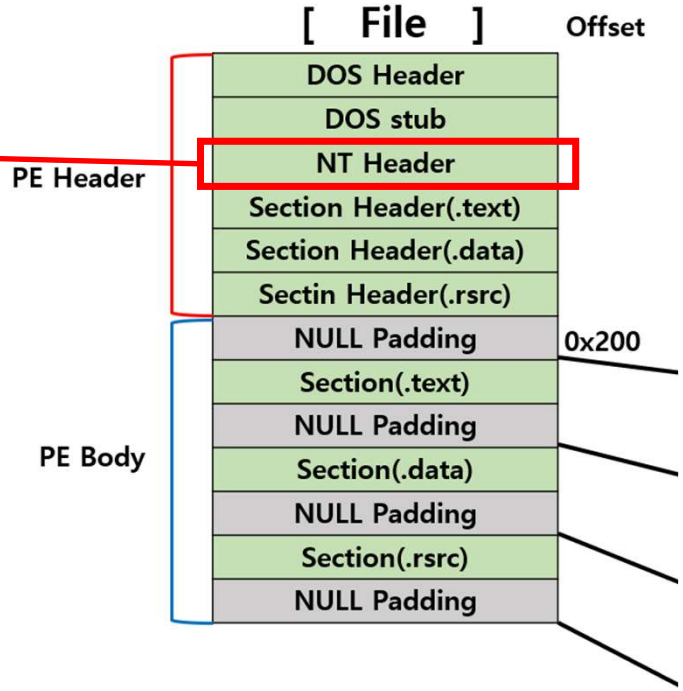


PIMAGE_NT_HEADERS

```
#ifdef _WIN64
typedef IMAGE_NT_HEADERS64                  IMAGE_NT_HEADERS;
typedef PIMAGE_NT_HEADERS64                 PIMAGE_NT_HEADERS;
#else
typedef IMAGE_NT_HEADERS32                  IMAGE_NT_HEADERS;
typedef PIMAGE_NT_HEADERS32                 PIMAGE_NT_HEADERS;
#endif
```

```
typedef struct _IMAGE_NT_HEADERS64 {
    DWORD Signature;
    IMAGE_FILE_HEADER FileHeader;
    IMAGE_OPTIONAL_HEADER64 OptionalHeader;
} IMAGE_NT_HEADERS64, *PIMAGE_NT_HEADERS64;

typedef struct _IMAGE_NT_HEADERS {
    DWORD Signature;
    IMAGE_FILE_HEADER FileHeader;
    IMAGE_OPTIONAL_HEADER32 OptionalHeader;
} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32;
```



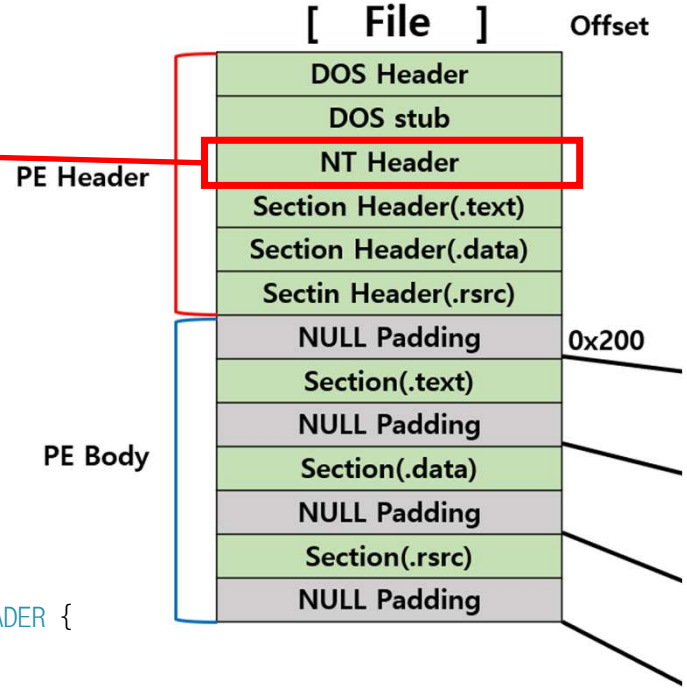
PIMAGE_NT_HEADERS

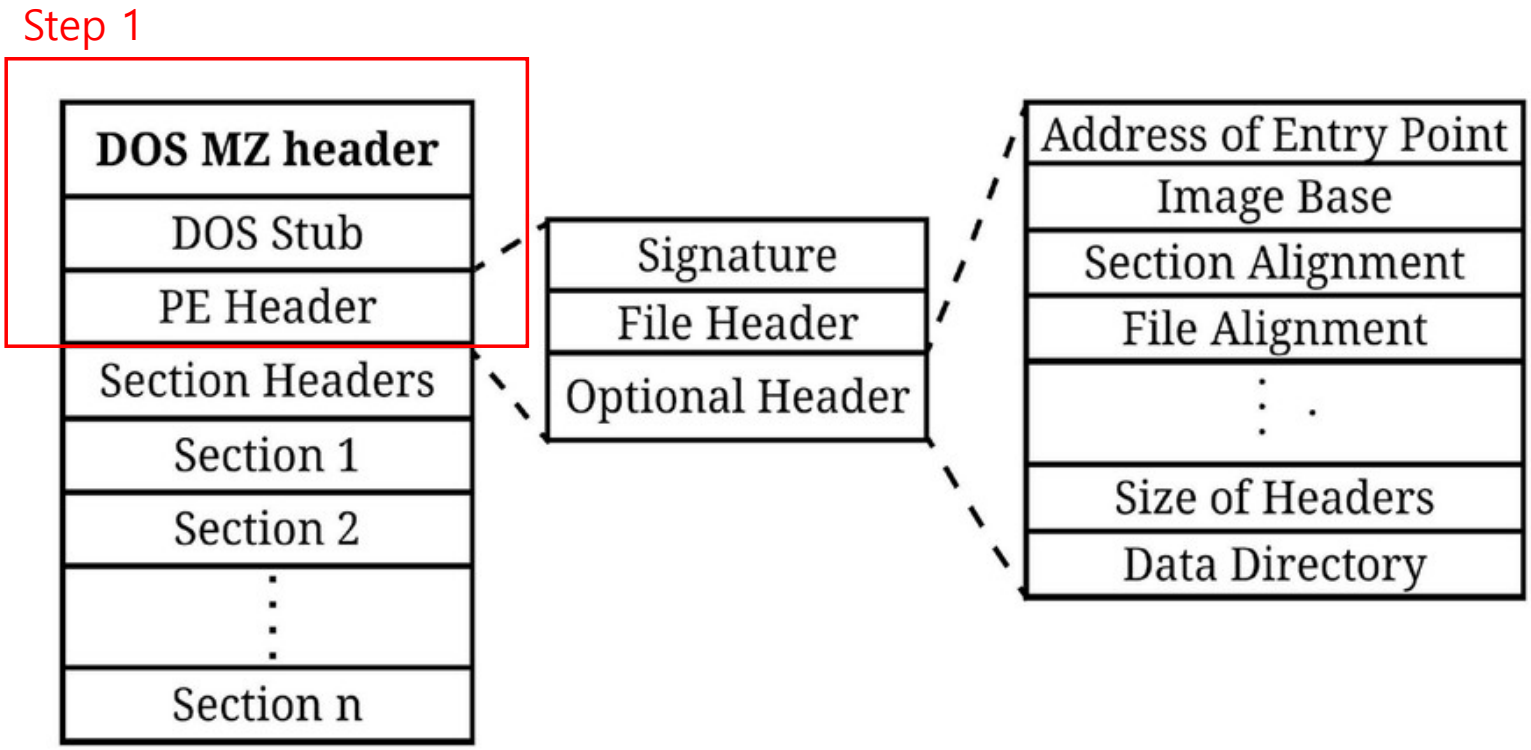
```
#ifdef _WIN64
typedef IMAGE_NT_HEADERS64                  IMAGE_NT_HEADERS;
typedef PIMAGE_NT_HEADERS64                 PIMAGE_NT_HEADERS;
#else
typedef IMAGE_NT_HEADERS32                  IMAGE_NT_HEADERS;
typedef PIMAGE_NT_HEADERS32                 PIMAGE_NT_HEADERS;
#endif
```

```
typedef struct _IMAGE_NT_HEADERS64 {
    DWORD Signature;
    IMAGE_FILE_HEADER FileHeader;
    IMAGE_OPTIONAL_HEADER64 OptionalHeader;
} IMAGE_NT_HEADERS64, *PIMAGE_NT_HEADERS64;

typedef struct _IMAGE_NT_HEADERS {
    DWORD Signature;
    IMAGE_FILE_HEADER FileHeader;
    IMAGE_OPTIONAL_HEADER32 OptionalHeader;
} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32;
```

```
typedef struct _IMAGE_FILE_HEADER {
    WORD    Machine;
    WORD    NumberOfSections;
    DWORD   TimeDateStamp;
    DWORD   PointerToSymbolTable;
    DWORD   NumberOfSymbols;
    WORD    SizeOfOptionalHeader;
    WORD    Characteristics;
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;
```




Step 1
DOS MZ header
DOS Stub
PE Header
Section Headers
Section 1
Section 2
Section n
Signature
File Header
Optional Header
Address of Entry Point
Image Base
Section Alignment
File Alignment
Size of Headers
Data Directory

<PE.h>

```
#pragma once
#include <Windows.h>

class pe
{
public:
pe(PBYTE peBuffer, SIZE_T peSize);
~pe();
bool isParse();
ULONG imageSize();
ULONG addressOfEntryPoint();
USHORT numberOfSection();
ULONG_PTR getFirstSection();
ULONG_PTR getNextSection(ULONG_PTR currentSection);
LONG_PTR getRelativeOffset(ULONG_PTR imagebase);
LONG_PTR getImageOffset();
void setImagebase(LONG_PTR imagebase);

ULONG_PTR memAlloc(SIZE_T size);

bool resolveIAT(ULONG_PTR imageBase);
void memFree(ULONG_PTR buildAddr);

PBYTE peHeader();
ULONG peHeaderSize();

private:
PBYTE m_peBuffer;
PIMAGE_DOS_HEADER m_pDosHeader;
PIMAGE_NT_HEADERS m_pNTHeader;
};
```

<PE.cpp 핵심코드>

```
m_pDosHeader = (PIMAGE_DOS_HEADER)m_peBuffer;
m_pNTHeader = (PIMAGE_NT_HEADERS)&m_peBuffer[m_pDosHeader->e_lfanew];



ULONG pe::imageSize()
{
if (!isParse())
return 0;

return (ULONG)m_pNTHeader->OptionalHeader.SizeOfImage;
}



LONG_PTR pe::getRelativeOffset(ULONG_PTR imagebase)
{
if (!isParse())
return 0;

return (LONG_PTR)imagebase - m_pNTHeader->OptionalHeader.ImageBase;
}
```

<PE.h>

```
#pragma once
#include <Windows.h>

class pe
{
public:
pe(PBYTE peBuffer, SIZE_T peSize);
~pe();
bool isParse();
ULONG imageSize();
ULONG addressOfEntryPoint();
USHORT numberOfSection();
ULONG_PTR getFirstSection();
ULONG_PTR getNextSection(ULONG_PTR currentSection);
LONG_PTR getRelativeOffset(ULONG_PTR imagebase);
LONG_PTR getImageOffset();
void setImagebase(LONG_PTR imagebase);

ULONG_PTR memAlloc(SIZE_T size);

bool resolveIAT(ULONG_PTR imageBase);
void memFree(ULONG_PTR buildAddr);

PBYTE peHeader();
ULONG peHeaderSize();

private:
PBYTE m_peBuffer;
PIMAGE_DOS_HEADER m_pDosHeader;
PIMAGE_NT_HEADERS m_pNTHeader;
};
```

<PE.cpp 핵심코드>

//winnt.h 구조체로 파싱

```
m_pDosHeader = (PIMAGE_DOS_HEADER)m_peBuffer;
m_pNTHeader = (PIMAGE_NT_HEADERS)&m_peBuffer[m_pDosHeader->e_lfanew];



ULONG pe::imageSize()
{
if (!isParse())
return 0;

return (ULONG)m_pNTHeader->OptionalHeader.SizeOfImage;
}



LONG_PTR pe::getRelativeOffset(ULONG_PTR imagebase)
{
if (!isParse())
return 0;

return (LONG_PTR)imagebase - m_pNTHeader->OptionalHeader.ImageBase;
}
```

<PE.h>

```
#pragma once
#include <Windows.h>

class pe
{
public:
pe(PBYTE peBuffer, SIZE_T peSize);
~pe();
bool isParse();
ULONG imageSize();
ULONG addressOfEntryPoint();
USHORT numberOfSection();
ULONG_PTR getFirstSection();
ULONG_PTR getNextSection(ULONG_PTR currentSection);
LONG_PTR getRelativeOffset(ULONG_PTR imagebase);
LONG_PTR getImageOffset();
void setImagebase(LONG_PTR imagebase);

ULONG_PTR memAlloc(SIZE_T size);

bool resolveIAT(ULONG_PTR imageBase);
void memFree(ULONG_PTR buildAddr);

PBYTE peHeader();
ULONG peHeaderSize();

private:
PBYTE m_peBuffer;
PIMAGE_DOS_HEADER m_pDosHeader;
PIMAGE_NT_HEADERS m_pNTHeader;
};
```

<PE.cpp 핵심코드>

//winnt.h 구조체로 파싱

```
m_pDosHeader = (PIMAGE_DOS_HEADER)m_peBuffer;
m_pNTHeader = (PIMAGE_NT_HEADERS)&m_peBuffer[m_pDosHeader->e_lfanew];
```

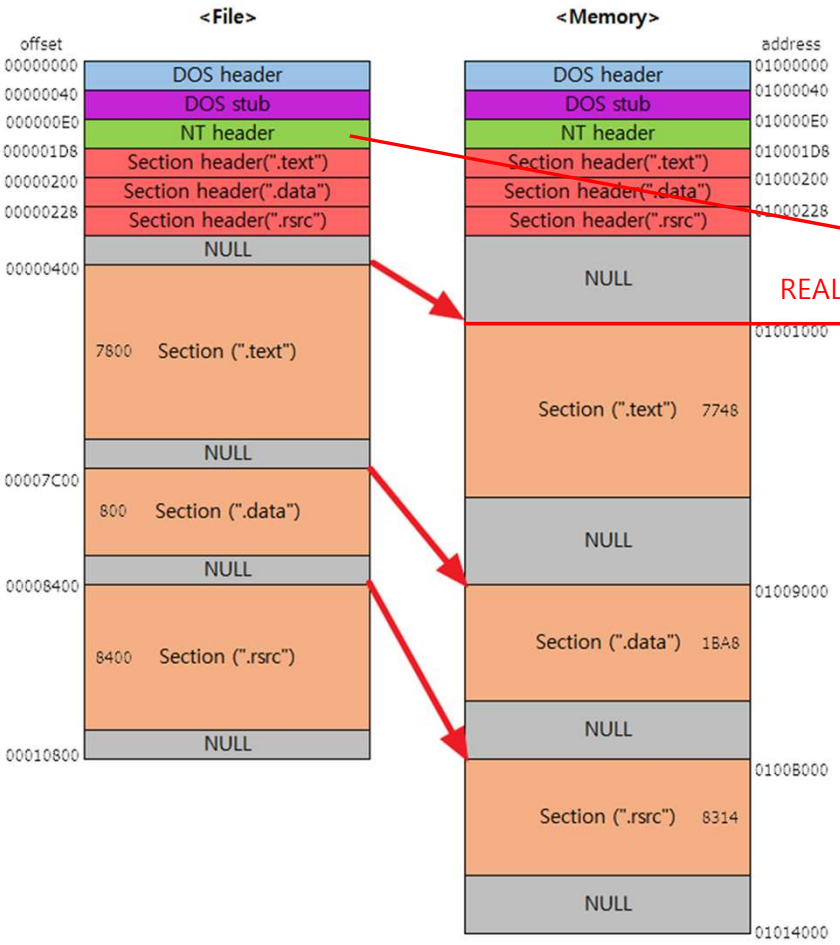
```
ULONG pe::imageSize()
{
if (!isParse())
return 0;
```

//필요한 정보를 OptionalHeader에서 load

```
return (ULONG)m_pNTHeader->OptionalHeader.SizeOfImage;
}
```

```
LONG_PTR pe::getRelativeOffset(ULONG_PTR imagebase)
{
if (!isParse())
return 0;

return (LONG_PTR)imagebase - m_pNTHeader->OptionalHeader.ImageBase;
}
```



REAL(Image base) – PE(Image base)

<PE.cpp 핵심코드>

//winnt.h 구조체로 파싱

```
m_pDosHeader = (PIMAGE_DOS_HEADER)m_peBuffer;
m_pNTHeader = (PIMAGE_NT_HEADERS)&m_peBuffer[m_pDosHeader->e_lfanew];
```

```
ULONG pe::imageSize()
{
if (!isParse())
return 0;
```

//필요한 정보를 OptionalHeader에서 load

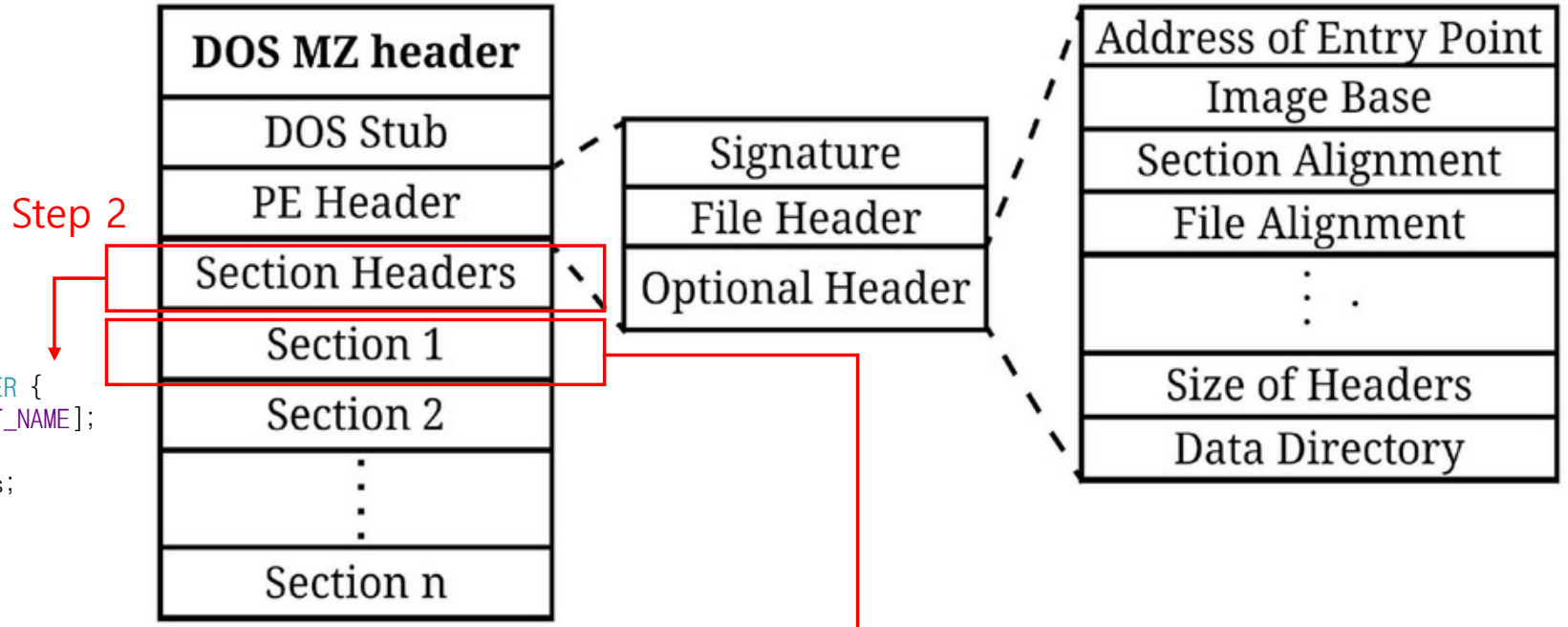
```
return (ULONG)m_pNTHeader->OptionalHeader.SizeOfImage;
}
```

//memory load 이후, breakpoint 등의 주소 연산을 위한 func

```
LONG_PTR pe::getRelativeOffset(ULONG_PTR imagebase)
{
if (!isParse())
return 0;

return (LONG_PTR)imagebase - m_pNTHeader->OptionalHeader.ImageBase;
}
```

//Offset 계산




```
typedef struct _IMAGE_SECTION_HEADER {
    BYTE    Name[IMAGE_SIZEOF_SHORT_NAME];
    union {
            DWORD   PhysicalAddress;
            DWORD   VirtualSize;
    } Misc;
    DWORD   VirtualAddress;
    DWORD   SizeOfRawData;
    DWORD   PointerToRawData;
    DWORD   PointerToRelocations;
    DWORD   PointerToLinenumbers;
    WORD    NumberOfRelocations;
    WORD    NumberOfLinenumbers;
    DWORD   Characteristics;
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

```
#define IMAGE_FIRST_SECTION( ntheader ) ((PIMAGE_SECTION_HEADER)        ₩
    ((ULONG_PTR)(ntheader) +                                            ₩
     FIELD_OFFSET( IMAGE_NT_HEADERS, OptionalHeader ) +                 ₩
     ((ntheader))->FileHeader.SizeOfOptionalHeader    ₩
    ))
```

<PE.cpp 핵심코드>

```
ULONG_PTR pe::getFirstSection()
{
if (!isParse())
return NULL;

return (ULONG_PTR)IMAGE_FIRST_SECTION(m_pNTHeader);
}
```

```
ULONG_PTR pe::getNextSection(ULONG_PTR currentSection)
{
if (!isParse())
return NULL;

return (ULONG_PTR)((PBYTE)currentSection + sizeof(IMAGE_SECTION_HEADER));
}
```

<print_func.cpp>

```
void PrintPEinfo(pe & peInstance)
{
    if (!peInstance.isParse()) {
        std::cout << "Failed to parse PE file." << std::endl;
        return;
    }
    std::cout << "==== PE File Information ====" << std::endl;
    // Basic PE information
    std::cout << "Image Size: " << peInstance.imageSize() << " bytes"
<< std::endl;
    std::cout << "Address of Entry Point: 0x" << std::hex
<< peInstance.addressOfEntryPoint() << std::dec << std::endl;
    std::cout << "Number of Sections: " << peInstance.numberOfSection()
<< std::endl;

    // PE Header information
    std::cout << "PE Header Address: "
<< reinterpret_cast<void*>(peInstance.peHeader()) << std::endl;
    std::cout << "PE Header Size: " << peInstance.peHeaderSize() << " bytes"
<< std::endl;

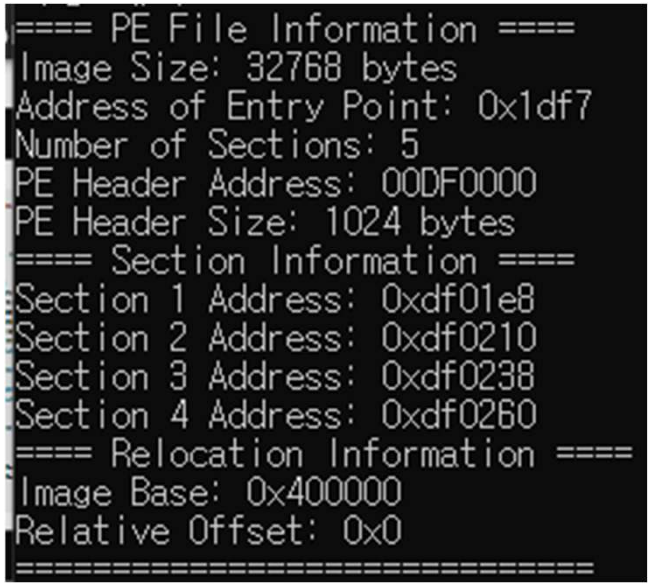
    // Section information
    ULONG_PTR firstSection = peInstance.getFirstSection();
    if (firstSection) {
        std::cout << "==== Section Information ====" << std::endl;
        ULONG_PTR currentSection = firstSection;
        int sectionIndex = 1;

        while (sectionIndex < 5) { // 너무 길어서 5줄 출력
            std::cout << "Section " << sectionIndex << " Address: 0x"
<< std::hex << currentSection << std::dec << std::endl;
            currentSection = peInstance.getNextSection(currentSection);
            ++sectionIndex;
        }
    }
    else {
        std::cout << "No sections found." << std::endl;
    }

    // Relocation information
    std::cout << "==== Relocation Information ====" << std::endl;
    ULONG_PTR imageBase = peInstance.getImageOffset();
    std::cout << "Image Base: 0x" << std::hex << imageBase << std::dec
<< std::endl;
    LONG_PTR relativeOffset = peInstance.getRelativeOffset(imageBase);
    std::cout << "Relative Offset: 0x" << std::hex << relativeOffset
<< std::dec << std::endl;

    // IAT resolution status
    //std::cout << "==== Import Address Table (IAT) ====" << std::endl;
    //if (peInstance.resolveIAT(imageBase)) { skip

    std::cout << "================================" << std::endl;
}
```


==== PE File Information ====
Image Size: 32768 bytes
Address of Entry Point: 0x1df7
Number of Sections: 5
PE Header Address: 00DF0000
PE Header Size: 1024 bytes
==== Section Information ====
Section 1 Address: 0xdf01e8
Section 2 Address: 0xdf0210
Section 3 Address: 0xdf0238
Section 4 Address: 0xdf0260
==== Relocation Information ====
Image Base: 0x400000
Relative Offset: 0x0
==============================

요구사항 :

- ~~PE parser~~
- Disassemble
- Memory View
- BreakPoint

Plan :

- ~~Windows Structure 를 공부하여 pe.h 제작~~
- Disassemble lib으로 capstone.h을 사용
- Debugee Process create, virtual memory R/W 용 exec.h
- Memory write로 int3를 injection



- **Disassemble**
  - Disassemble lib으로 capstone.h을 사용

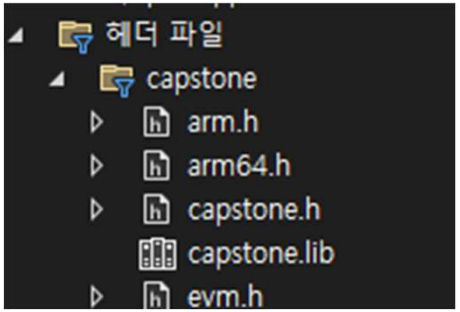


- **Disassemble**
- Disassemble lib으로 capstone.h을 사용

Capstone is a **disassembly framework** with the target of becoming the ultimate disasm engine for binary analysis and reversing in the security community.

Ref : https://github.com/capstone-engine/capstone


Git clone capstone
Build through MSVC
Get capstone DLL,LIB




```
#include "capstone/capstone.h"
#pragma comment(lib, "capstone/capstone.lib")
```

<print_func.cpp>

```
void PrintDisassembleProcessMemory(HANDLE hProcess, LPCVOID address, size_t
size) {
    BYTE* buffer = new BYTE[size];
    SIZE_T bytesRead;

    // 타 프로세스 메모리 읽기
    if (!ReadProcessMemory(hProcess, address, buffer, size, &bytesRead)) {
        printf("Error: Could not read memory at address 0x%p. Error
Code: %lu₩n", address, GetLastError());
        delete[] buffer;
        return;
    }

    // Capstone 디스어셈블러 초기화
    csh handle;
#if _M_X64
    if (cs_open(CS_ARCH_X86, CS_MODE_64, &handle) != CS_ERR_OK) {
#else
    if (cs_open(CS_ARCH_X86, CS_MODE_32, &handle) != CS_ERR_OK) {
#endif
        printf("Error: Failed to initialize Capstone disassembler₩n");
        delete[] buffer;
        return;
    }
```

```
 // 디스어셈블링
    cs_insn* insn;
    size_t count = cs_disasm(handle, buffer, bytesRead, (uint64_t)address,
0, &insn);

    if (count > 0) {
        printf("Disassembly at address 0x%p:₩n", address);
        for (size_t i = 0; i < count; i++) {
            printf("0x%llx:₩t%s₩t₩t%s₩n",
                insn[i].address,
                insn[i].mnemonic,
                insn[i].op_str);
        }
        cs_free(insn, count);
    }
    else {
        printf("Error: Failed to disassemble memory at address 0x%p₩n",
address);
    }

    // Capstone 해제
    cs_close(&handle);
    delete[] buffer;
    }
```

```
PrintDisassembleProcessMemory(targetprocess.m_process, (LPCVOID)entry_addr, 0x10);
```

<print_func.cpp>

```
void PrintDisassembleProcessMemory(HANDLE hProcess, LPCVOID address, size_t
size) {
    BYTE* buffer = new BYTE[size];
    SIZE_T bytesRead;

    // 타 프로세스 메모리 읽기
    if (!ReadProcessMemory(hProcess, address, buffer, size, &bytesRead)) {
        printf("Error: Could not read memory at address 0x%p. Error
Code: %lu₩n", address, GetLastError());
        delete[] buffer;
        return;
    }

    // Capstone 디스어셈블러 초기화
    csh handle;
#if _M_X64
    if (cs_open(CS_ARCH_X86, CS_MODE_64, &handle) != CS_ERR_OK) {
#else
    if (cs_open(CS_ARCH_X86, CS_MODE_32, &handle) != CS_ERR_OK) {
#endif
        printf("Error: Failed to initialize Capstone disassembler₩n");
        delete[] buffer;
        return;
    }
```

```
 // 디스어셈블링
    cs_insn* insn;
    size_t count = cs_disasm(handle, buffer, bytesRead, (uint64_t)address,
0, &insn);

    if (count > 0) {
        printf("Disassembly at address 0x%p:₩n", address);
        for (size_t i = 0; i < count; i++) {
            printf("0x%llx:₩t%s₩t₩t%s₩n",
                insn[i].address,
                insn[i].mnemonic,
                insn[i].op_str);
        }
        cs_free(insn, count);
    }
    else {
        printf("Error: Failed to disassemble memory at address 0x%p₩n",
address);
    }

    // Capstone 해제
    cs_close(&handle);
    delete[] buffer;
    }
```

PrintDisassembleProcessMemory(targetprocess.m_process, (LPCVOID)entry_addr, 0x10);
handle

<print_func.cpp>

```
void PrintDisassembleProcessMemory(HANDLE hProcess, LPCVOID address, size_t
size) {
    BYTE* buffer = new BYTE[size];
    SIZE_T bytesRead;

    // 타 프로세스 메모리 읽기
    if (!ReadProcessMemory(hProcess, address, buffer, size, &bytesRead)) {
        printf("Error: Could not read memory at address 0x%p. Error
Code: %lu₩n", address, GetLastError());
        delete[] buffer;
        return;
    }

    // Capstone 디스어셈블러 초기화
    csh handle;
#if _M_X64
    if (cs_open(CS_ARCH_X86, CS_MODE_64, &handle) != CS_ERR_OK) {
#else
    if (cs_open(CS_ARCH_X86, CS_MODE_32, &handle) != CS_ERR_OK) {
#endif
        printf("Error: Failed to initialize Capstone disassembler₩n");
        delete[] buffer;
        return;
    }
```

```
 // 디스어셈블링
    cs_insn* insn;
    size_t count = cs_disasm(handle, buffer, bytesRead, (uint64_t)address,
0, &insn);

    if (count > 0) {
        printf("Disassembly at address 0x%p:₩n", address);
        for (size_t i = 0; i < count; i++) {
            printf("0x%llx:₩t%s₩t₩t%s₩n",
                insn[i].address,
                insn[i].mnemonic,
                insn[i].op_str);
        }
        cs_free(insn, count);
    }
    else {
        printf("Error: Failed to disassemble memory at address 0x%p₩n",
address);
    }

    // Capstone 해제
    cs_close(&handle);
    delete[] buffer;
    }
```

```
PrintDisassembleProcessMemory(targetprocess.m_process, (LPCVOID)entry_addr, 0x10);
```

Disassemble target ADDR

<print_func.cpp>

```
void PrintDisassembleProcessMemory(HANDLE hProcess, LPCVOID address, size_t
size) {
    BYTE* buffer = new BYTE[size];
    SIZE_T bytesRead;

    // 타 프로세스 메모리 읽기
    if (!ReadProcessMemory(hProcess, address, buffer, size, &bytesRead)) {
        printf("Error: Could not read memory at address 0x%p. Error
Code: %lu₩n", address, GetLastError());
        delete[] buffer;
        return;
    }

    // Capstone 디스어셈블러 초기화
    csh handle;
#if _M_X64
    if (cs_open(CS_ARCH_X86, CS_MODE_64, &handle) != CS_ERR_OK) {
#else
    if (cs_open(CS_ARCH_X86, CS_MODE_32, &handle) != CS_ERR_OK) {
#endif
        printf("Error: Failed to initialize Capstone disassembler₩n");
        delete[] buffer;
        return;
    }
```

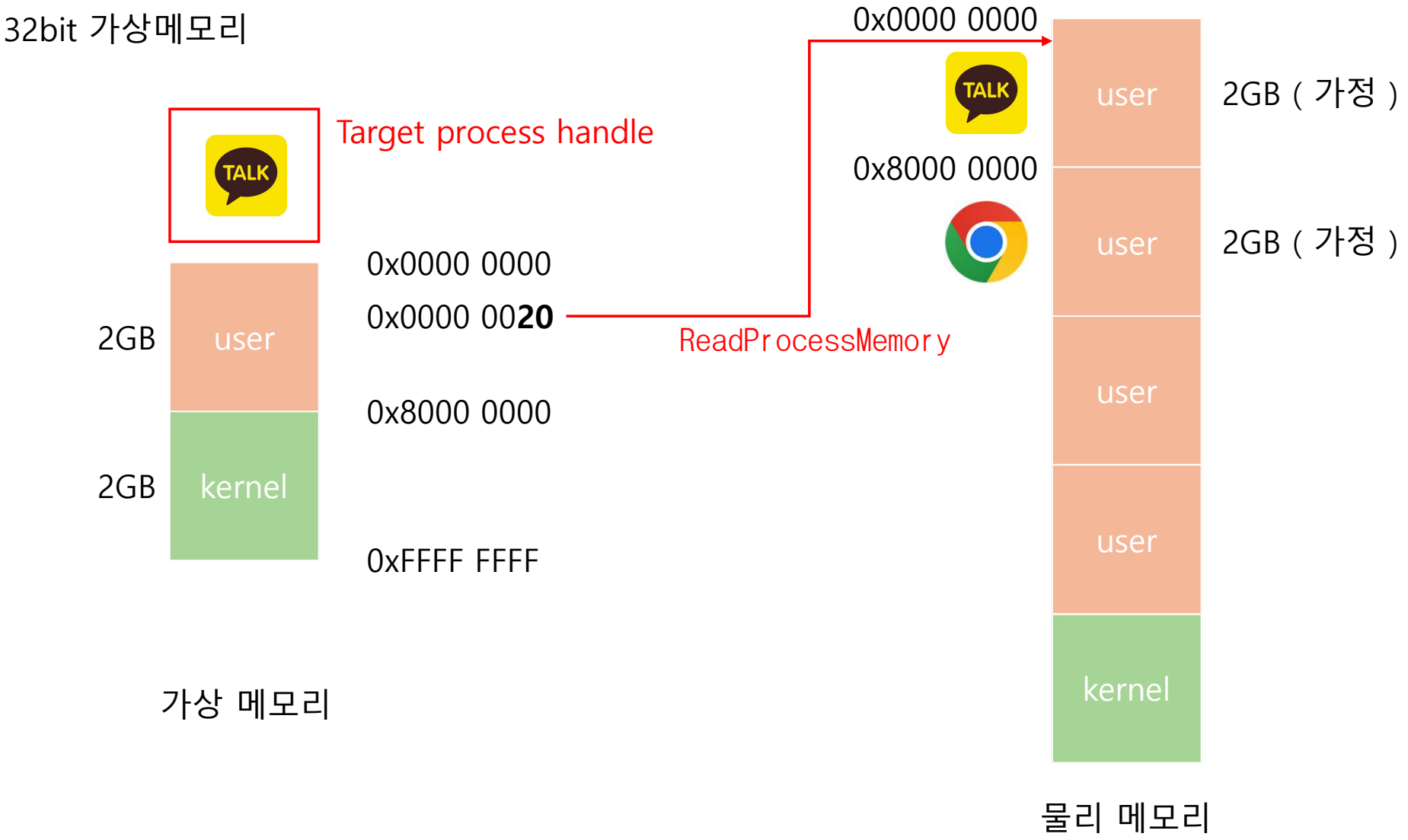
```
 // 디스어셈블링
    cs_insn* insn;
    size_t count = cs_disasm(handle, buffer, bytesRead, (uint64_t)address,
0, &insn);

    if (count > 0) {
        printf("Disassembly at address 0x%p:₩n", address);
        for (size_t i = 0; i < count; i++) {
            printf("0x%llx:₩t%s₩t₩t%s₩n",
                insn[i].address,
                insn[i].mnemonic,
                insn[i].op_str);
        }
        cs_free(insn, count);
    }
    else {
        printf("Error: Failed to disassemble memory at address 0x%p₩n",
address);
    }

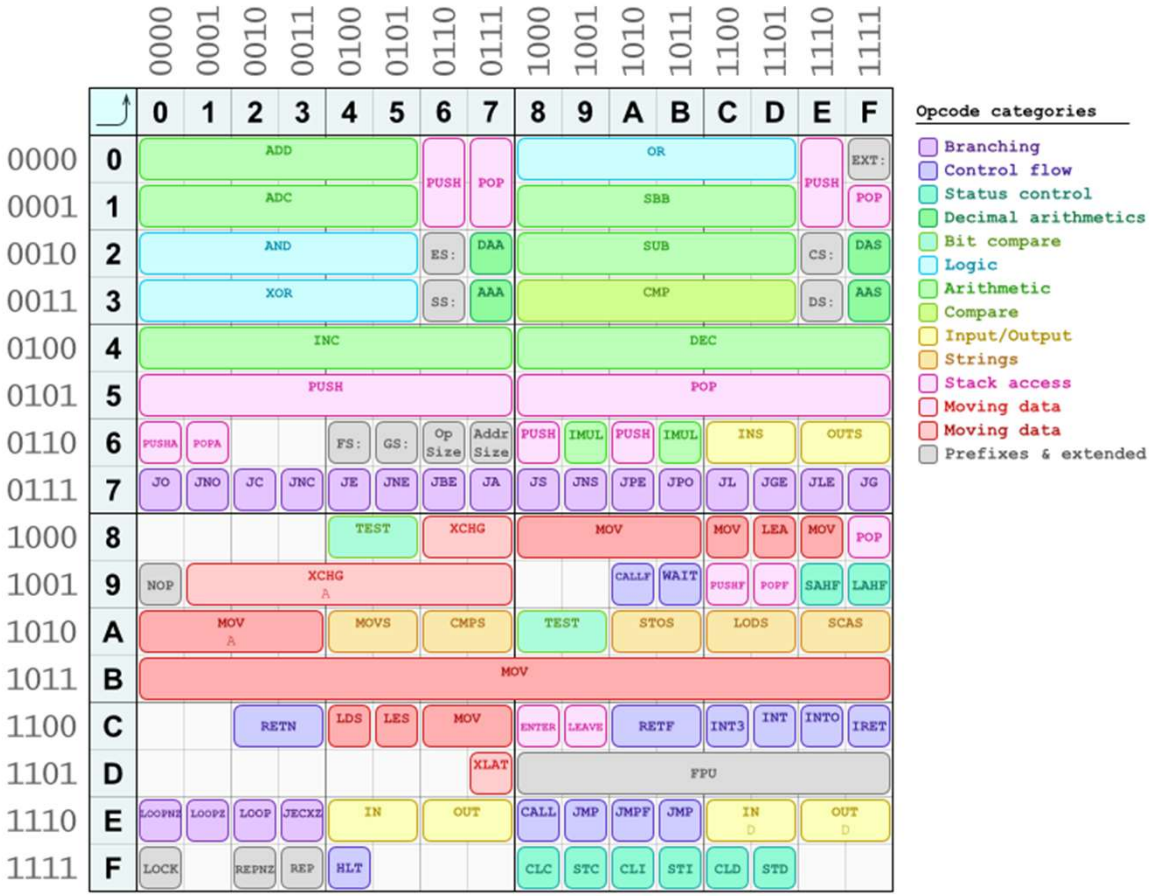
    // Capstone 해제
    cs_close(&handle);
    delete[] buffer;
    }
```

```
PrintDisassembleProcessMemory(targetprocess.m_process, (LPCVOID)entry_addr, 0x10);
```

Disassemble 영역


32bit 가상메모리
Target process handle
0x0000 0000
0x0000 0020
0x8000 0000
0xFFFF FFFF
2GB
user
2GB
kernel
가상 메모리
ReadProcessMemory
0x0000 0000
0x8000 0000
user
user
user
user
kernel
2GB (가정)
2GB (가정)
물리 메모리

32bit, 64bit ?



https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html

<print_func.cpp>

```
void PrintDisassembleProcessMemory(HANDLE hProcess, LPCVOID address, size_t size) {
    BYTE* buffer = new BYTE[size];
    SIZE_T bytesRead;

    // 타 프로세스 메모리 읽기
    if (!ReadProcessMemory(hProcess, address, buffer, size, &bytesRead)) {
        printf("Error: Could not read memory at address 0x%p. Error Code: %lu₩n", address, GetLastError());
        delete[] buffer;
        return;
    }

    // Capstone 디스어셈블러 초기화
    csh handle;
#if _M_X64
    if (cs_open(CS_ARCH_X86, CS_MODE_64, &handle) != CS_ERR_OK) {
#else
    if (cs_open(CS_ARCH_X86, CS_MODE_32, &handle) != CS_ERR_OK) {
#endif
        printf("Error: Failed to initialize Capstone disassembler₩n");
        delete[] buffer;
        return;
    }
```

```
    // 디스어셈블링
    cs_insn* insn;
    size_t count = cs_disasm(handle, buffer, bytesRead, (uint64_t)address, 0, &insn);

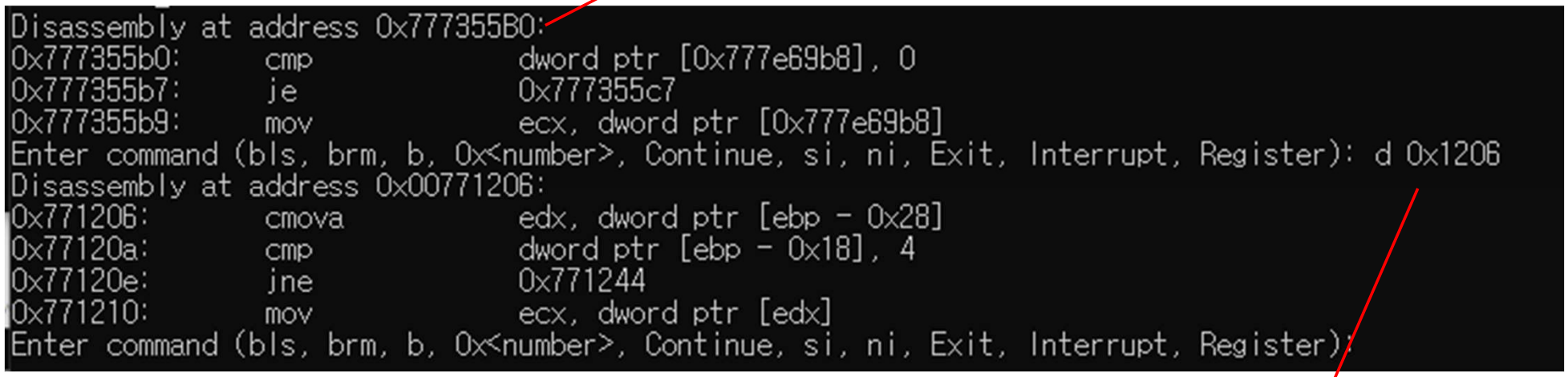
    if (count > 0) {
        printf("Disassembly at address 0x%p:₩n", address);
        for (size_t i = 0; i < count; i++) {
            printf("0x%llx:₩t%s₩t₩t%s₩n",
                insn[i].address,
                insn[i].mnemonic,
                insn[i].op_str);
        }
        cs_free(insn, count);
    }
    else {
        printf("Error: Failed to disassemble memory at address 0x%p₩n", address);
    }

    // Capstone 해제
    cs_close(&handle);
    delete[] buffer;
    }
```

- `_M_X64` x64 프로세서 또는 ARM64EC 대상으로 하는 컴파일의 정수 리터럴 값 100으로 정의됩니다. 그 이외의 경우에는 정의되지 않습니다.

Ref : https://learn.microsoft.com/ko-kr/cpp/preprocessor/predefined-macros?view=msvc-170

Initialize 시 entry point를 기준으로 disassemble



명령어"d 0x주소 "를 통해 임의 주소 disassemble

요구사항 :

- ~~PE parser~~
- ~~Disassemble~~
- Memory View
- BreakPoint

Plan :

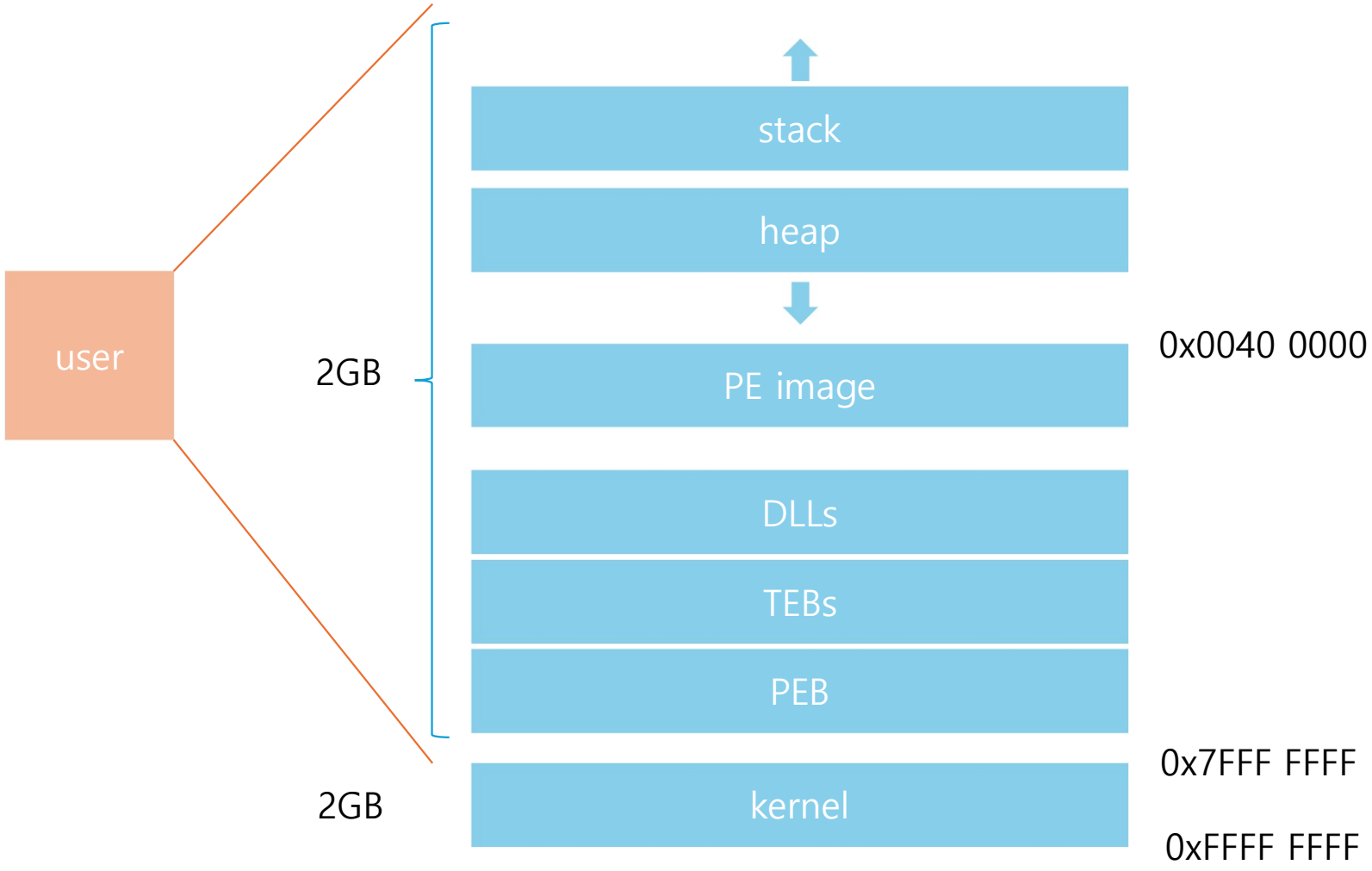
- ~~Windows Structure 를 공부하여 pe.h 제작~~
- ~~Disassemble lib으로 capstone.h을 사용~~
- Debugee Process create, virtual memory R/W 용 exec.h
- Memory write로 int3를 injection

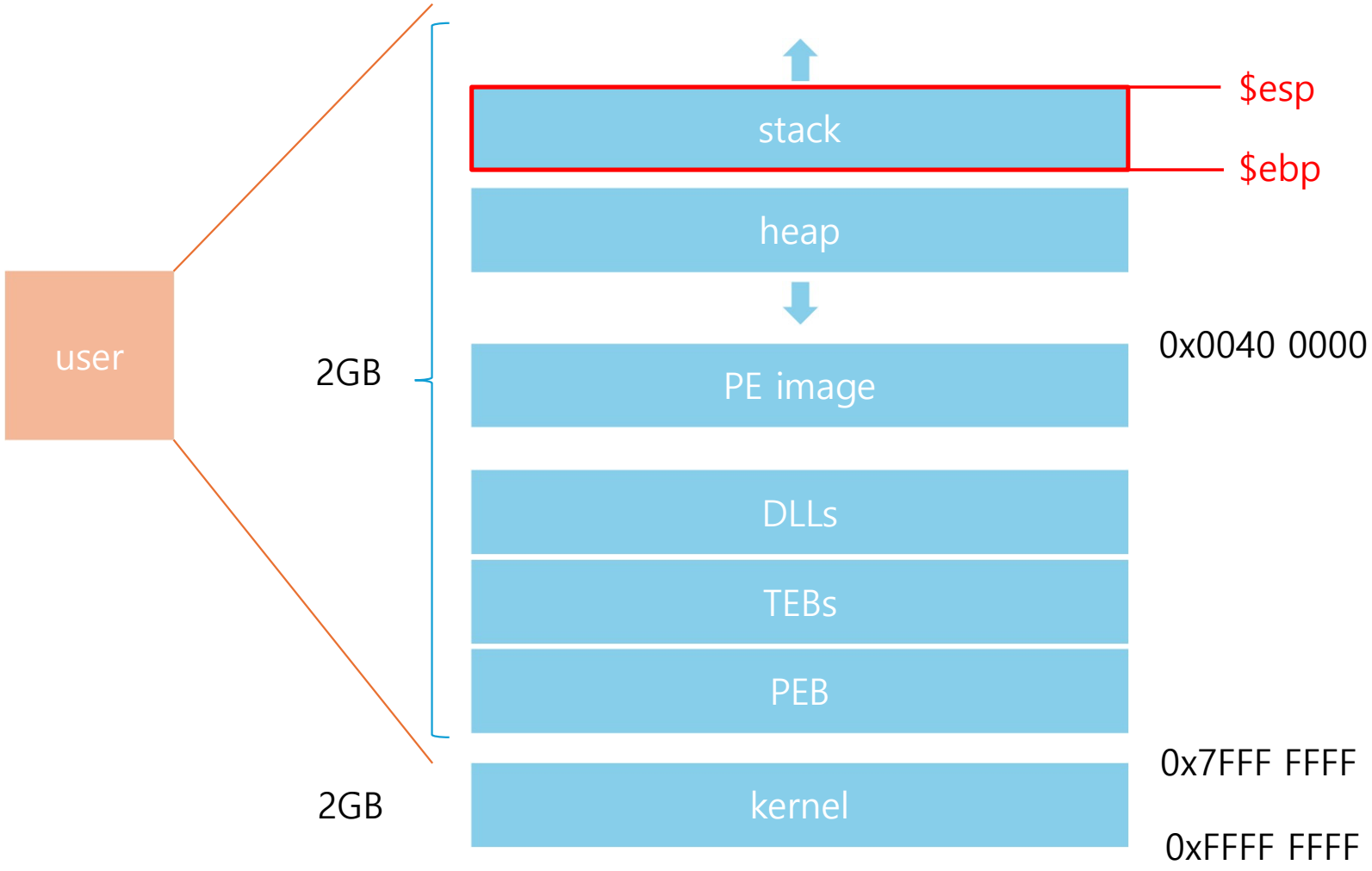
요구사항 :

- **Memory View**

Plan :

- Debugee Process create, virtual memory R/W 용 exec.h


user
2GB
stack
heap
PE image
DLLs
TEBs
PEB
kernel
2GB
0x0040 0000
0x7FFF FFFF
0xFFFF FFFF


user
2GB
2GB
stack
heap
PE image
DLLs
TEBs
PEB
kernel
$esp
$ebp
0x0040 0000
0x7FFF FFFF
0xFFFF FFFF

<print_func.cpp>

```
void PrintRegisters(CONTEXT & context) {
#ifdef _M_X64
    // 64-bit 레지스터
    std::cout << "RAX: " << std::hex << context.Rax << "₩n";
    std::cout << "RBX: " << std::hex << context.Rbx << "₩n";
    std::cout << "RCX: " << std::hex << context.Rcx << "₩n";
    std::cout << "RDX: " << std::hex << context.Rdx << "₩n";
    std::cout << "RIP: " << std::hex << context.Rip << "₩n";
    std::cout << "RSP: " << std::hex << context.Rsp << "₩n";
    std::cout << "RBP: " << std::hex << context.Rbp << "₩n";
    std::cout << "RSI: " << std::hex << context.Rsi << "₩n";
    std::cout << "RDI: " << std::hex << context.Rdi << "₩n";
#else
    // 32-bit 레지스터
    std::cout << "EAX: " << std::hex << context.Eax << "₩n";
    std::cout << "EBX: " << std::hex << context.Ebx << "₩n";
    std::cout << "ECX: " << std::hex << context.Ecx << "₩n";
    std::cout << "EDX: " << std::hex << context.Edx << "₩n";
    std::cout << "EIP: " << std::hex << context.Eip << "₩n";
    std::cout << "ESP: " << std::hex << context.Esp << "₩n";
    std::cout << "EBP: " << std::hex << context.Ebp << "₩n";
    std::cout << "ESI: " << std::hex << context.Esi << "₩n";
    std::cout << "EDI: " << std::hex << context.Edi << "₩n";
#endif
}
```

```
void PrintStackMemory(HANDLE hProcess, CONTEXT & context) {
#if _M_X64
    uint64_t bp = context.Rbp; // RBP 레지스터
    uint64_t sp = context.Rsp; // RSP 레지스터
#else
    uint32_t bp = context.Ebp; // EBP 레지스터
    uint32_t sp = context.Esp; // ESP 레지스터
#endif
<중략>
    size_t count = (bp - sp) / sizeof(uint32_t);

    for (size_t i = 0; i < count; i++) {
        uint32_t value;
        if (!ReadProcessMemory(hProcess, (LPCVOID)(sp + i *
sizeof(uint32_t)), &value, sizeof(uint32_t), NULL)) {
            printf("0x%lx: Error reading memory₩n", (unsigned long)(sp + i
* sizeof(uint32_t)));
        }
        else {
            printf("0x%lx: 0x%x₩n", (unsigned long)(sp + i *
sizeof(uint32_t)), value);
        }
    }
#endif
}
```

<print_func.cpp>

```cpp
void PrintRegisters(CONTEXT & context) {
#ifdef _M_X64
    // 64-bit 레지스터
    std::cout << "RAX: " << std::hex << context.Rax << "₩n";
    std::cout << "RBX: " << std::hex << context.Rbx << "₩n";
    std::cout << "RCX: " << std::hex << context.Rcx << "₩n";
    std::cout << "RDX: " << std::hex << context.Rdx << "₩n";
    std::cout << "RIP: " << std::hex << context.Rip << "₩n";
    std::cout << "RSP: " << std::hex << context.Rsp << "₩n";
    std::cout << "RBP: " << std::hex << context.Rbp << "₩n";
    std::cout << "RSI: " << std::hex << context.Rsi << "₩n";
    std::cout << "RDI: " << std::hex << context.Rdi << "₩n";
#else
    // 32-bit 레지스터
    std::cout << "EAX: " << std::hex << context.Eax << "₩n";
    std::cout << "EBX: " << std::hex << context.Ebx << "₩n";
    std::cout << "ECX: " << std::hex << context.Ecx << "₩n";
    std::cout << "EDX: " << std::hex << context.Edx << "₩n";
    std::cout << "EIP: " << std::hex << context.Eip << "₩n";
    std::cout << "ESP: " << std::hex << context.Esp << "₩n";
    std::cout << "EBP: " << std::hex << context.Ebp << "₩n";
    std::cout << "ESI: " << std::hex << context.Esi << "₩n";
    std::cout << "EDI: " << std::hex << context.Edi << "₩n";
#endif
}
```

```cpp
void PrintStackMemory(HANDLE hProcess, CONTEXT & context) {
#if _M_X64
    uint64_t bp = context.Rbp; // RBP 레지스터
    uint64_t sp = context.Rsp; // RSP 레지스터
#else
    uint32_t bp = context.Ebp; // EBP 레지스터
    uint32_t sp = context.Esp; // ESP 레지스터
#endif
<중략>
    size_t count = (bp - sp) / sizeof(uint32_t);

    for (size_t i = 0; i < count; i++) {
        uint32_t value;
        if (!ReadProcessMemory(hProcess, (LPCVOID)(sp + i *
sizeof(uint32_t)), &value, sizeof(uint32_t), NULL)) {
            printf("0x%lx: Error reading memory₩n", (unsigned long)(sp + i
* sizeof(uint32_t)));
        }
        else {
            printf("0x%lx: 0x%x₩n", (unsigned long)(sp + i *
sizeof(uint32_t)), value);
        }
    }
#endif
}
```

Get REGISTER

```cpp
targetprocess.getThreadContext( );
PrintStackMemory(targetprocess.m_process,*targetprocess.m_pthreadContext );
```

<print_func.cpp>

```cpp
void PrintRegisters(CONTEXT & context) {
#ifdef _M_X64
    // 64-bit 레지스터
    std::cout << "RAX: " << std::hex << context.Rax << "₩n";
    std::cout << "RBX: " << std::hex << context.Rbx << "₩n";
    std::cout << "RCX: " << std::hex << context.Rcx << "₩n";
    std::cout << "RDX: " << std::hex << context.Rdx << "₩n";
    std::cout << "RIP: " << std::hex << context.Rip << "₩n";
    std::cout << "RSP: " << std::hex << context.Rsp << "₩n";
    std::cout << "RBP: " << std::hex << context.Rbp << "₩n";
    std::cout << "RSI: " << std::hex << context.Rsi << "₩n";
    std::cout << "RDI: " << std::hex << context.Rdi << "₩n";
#else
    // 32-bit 레지스터
    std::cout << "EAX: " << std::hex << context.Eax << "₩n";
    std::cout << "EBX: " << std::hex << context.Ebx << "₩n";
    std::cout << "ECX: " << std::hex << context.Ecx << "₩n";
    std::cout << "EDX: " << std::hex << context.Edx << "₩n";
    std::cout << "EIP: " << std::hex << context.Eip << "₩n";
    std::cout << "ESP: " << std::hex << context.Esp << "₩n";
    std::cout << "EBP: " << std::hex << context.Ebp << "₩n";
    std::cout << "ESI: " << std::hex << context.Esi << "₩n";
    std::cout << "EDI: " << std::hex << context.Edi << "₩n";
#endif
}
```

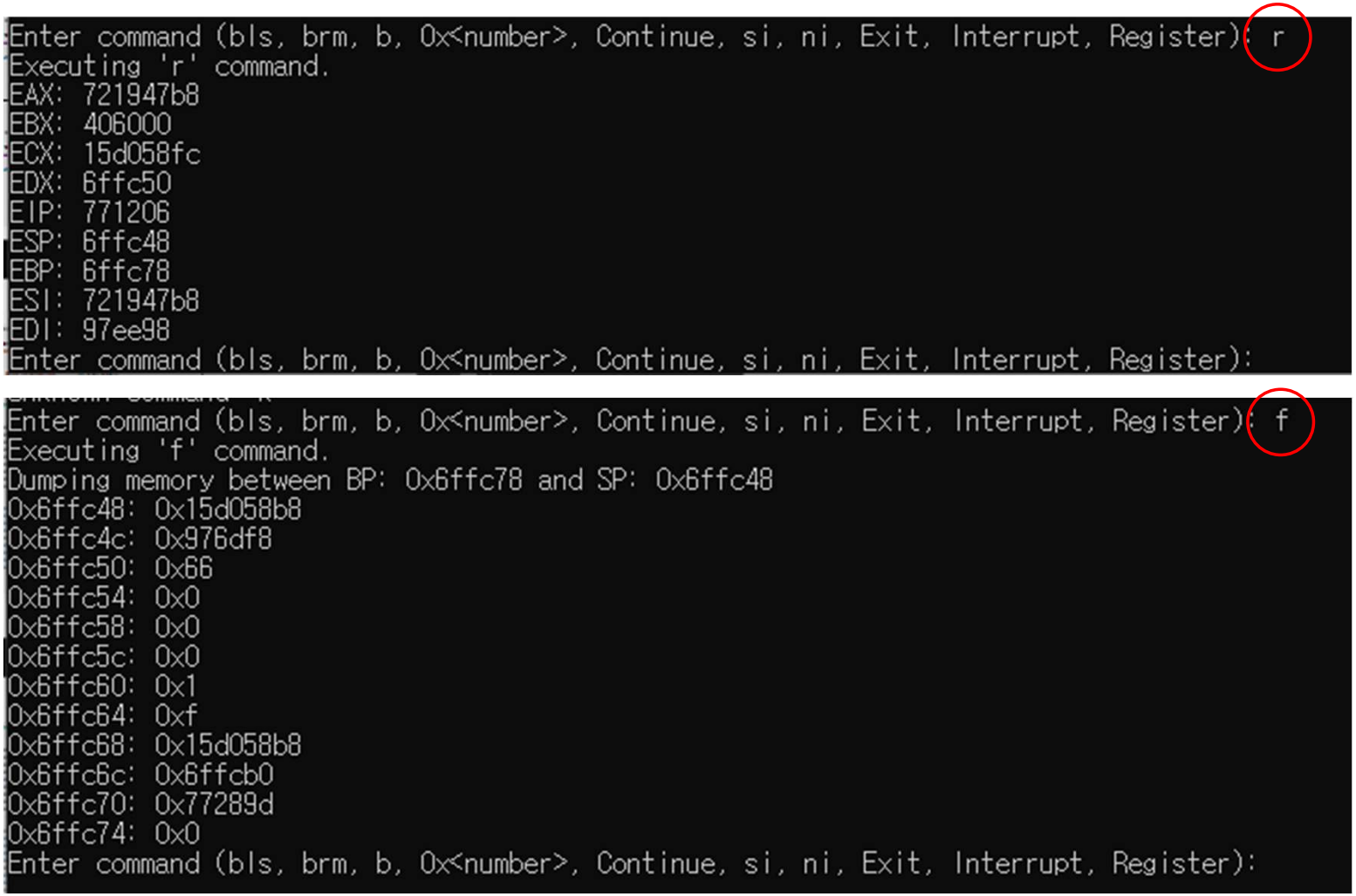
```cpp
void PrintStackMemory(HANDLE hProcess, CONTEXT & context) {
#if _M_X64
    uint64_t bp = context.Rbp; // RBP 레지스터
    uint64_t sp = context.Rsp; // RSP 레지스터
#else
    uint32_t bp = context.Ebp; // EBP 레지스터
    uint32_t sp = context.Esp; // ESP 레지스터
#endif
<중략>
    size_t count = (bp - sp) / sizeof(uint32_t);

    for (size_t i = 0; i < count; i++) {
        uint32_t value;
        if (!ReadProcessMemory(hProcess, (LPCVOID)(sp + i *
sizeof(uint32_t)), &value, sizeof(uint32_t), NULL)) {
            printf("0x%lx: Error reading memory₩n", (unsigned long)(sp + i
* sizeof(uint32_t)));
        }
        else {
            printf("0x%lx: 0x%x₩n", (unsigned long)(sp + i *
sizeof(uint32_t)), value);
        }
    }
#endif
}
```

Read Process Memory

Get REGISTER

```cpp
targetprocess.getThreadContext();
PrintStackMemory(targetprocess.m_process,*targetprocess.m_pthreadContext);
```


Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): r
Executing 'r' command.
EAX: 721947b8
EBX: 406000
ECX: 15d058fc
EDX: 6ffc50
EIP: 771206
ESP: 6ffc48
EBP: 6ffc78
ESI: 721947b8
EDI: 97ee98
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register):
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): f
Executing 'f' command.
Dumping memory between BP: 0x6ffc78 and SP: 0x6ffc48
0x6ffc48: 0x15d058b8
0x6ffc4c: 0x976df8
0x6ffc50: 0x66
0x6ffc54: 0x0
0x6ffc58: 0x0
0x6ffc5c: 0x0
0x6ffc60: 0x1
0x6ffc64: 0xf
0x6ffc68: 0x15d058b8
0x6ffc6c: 0x6ffcb0
0x6ffc70: 0x77289d
0x6ffc74: 0x0
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register):

debuggee context와 handle 관리

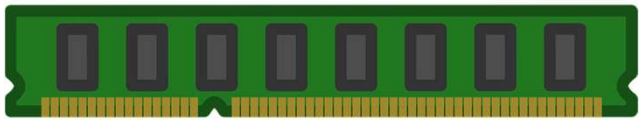
debuggee : debugger로 debugging 할 대상



debugger

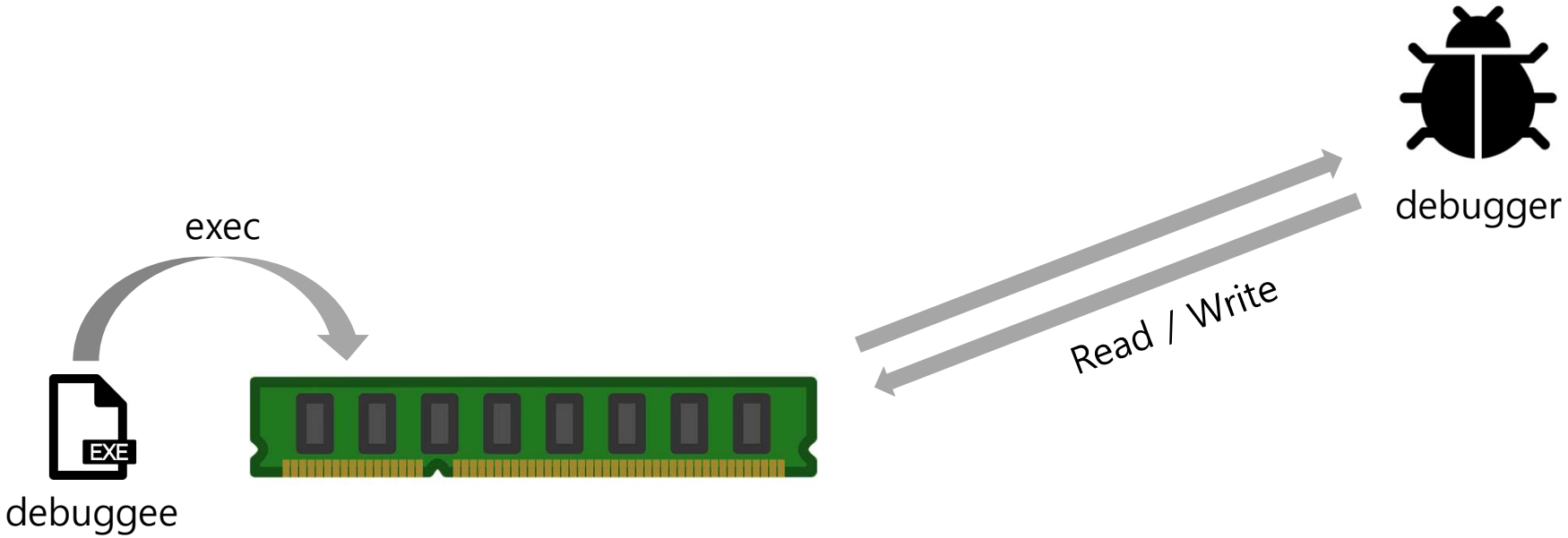


debuggee



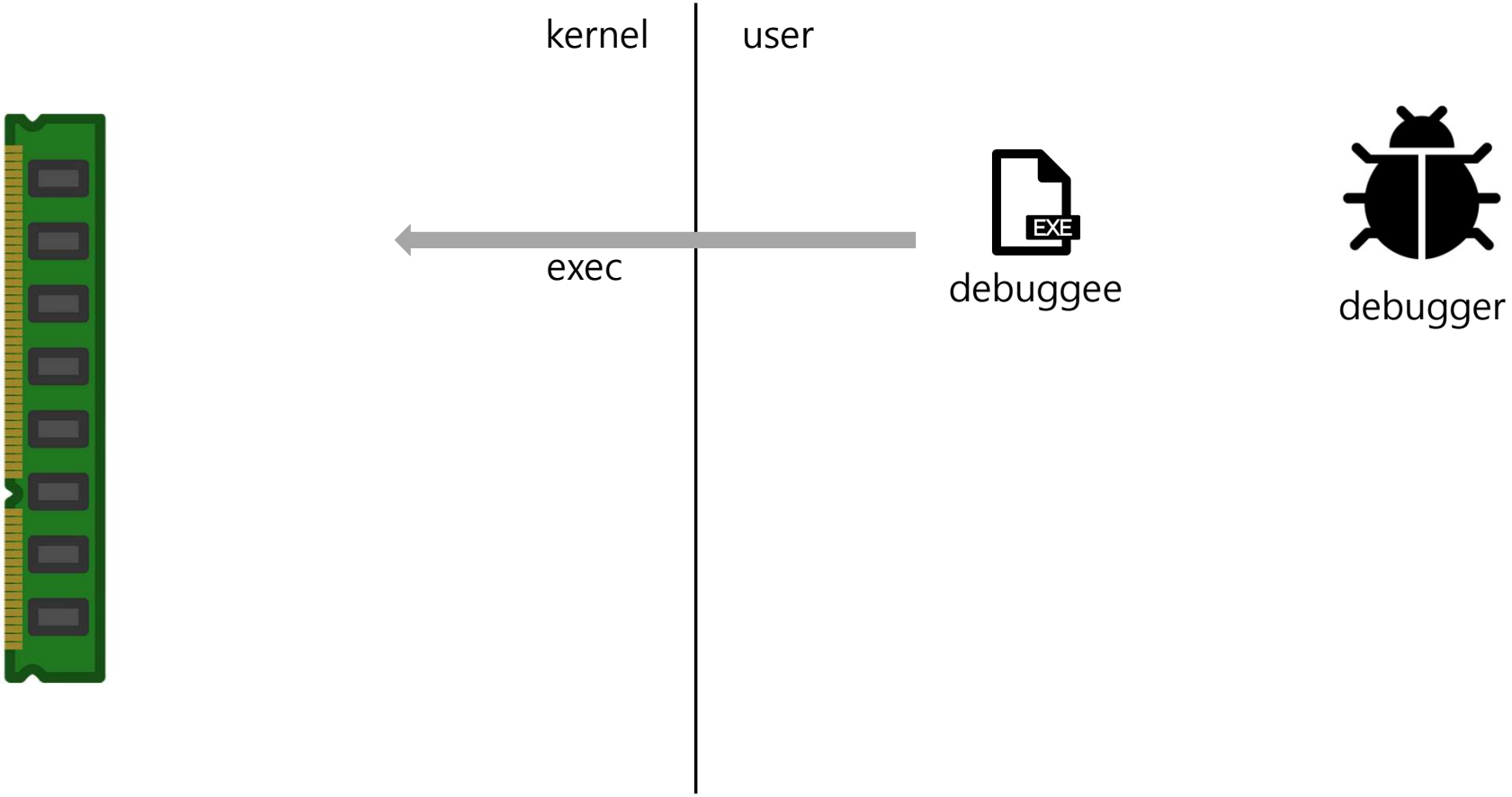
debuggee context와 handle 관리

debuggee : debugger로 debugging 할 대상



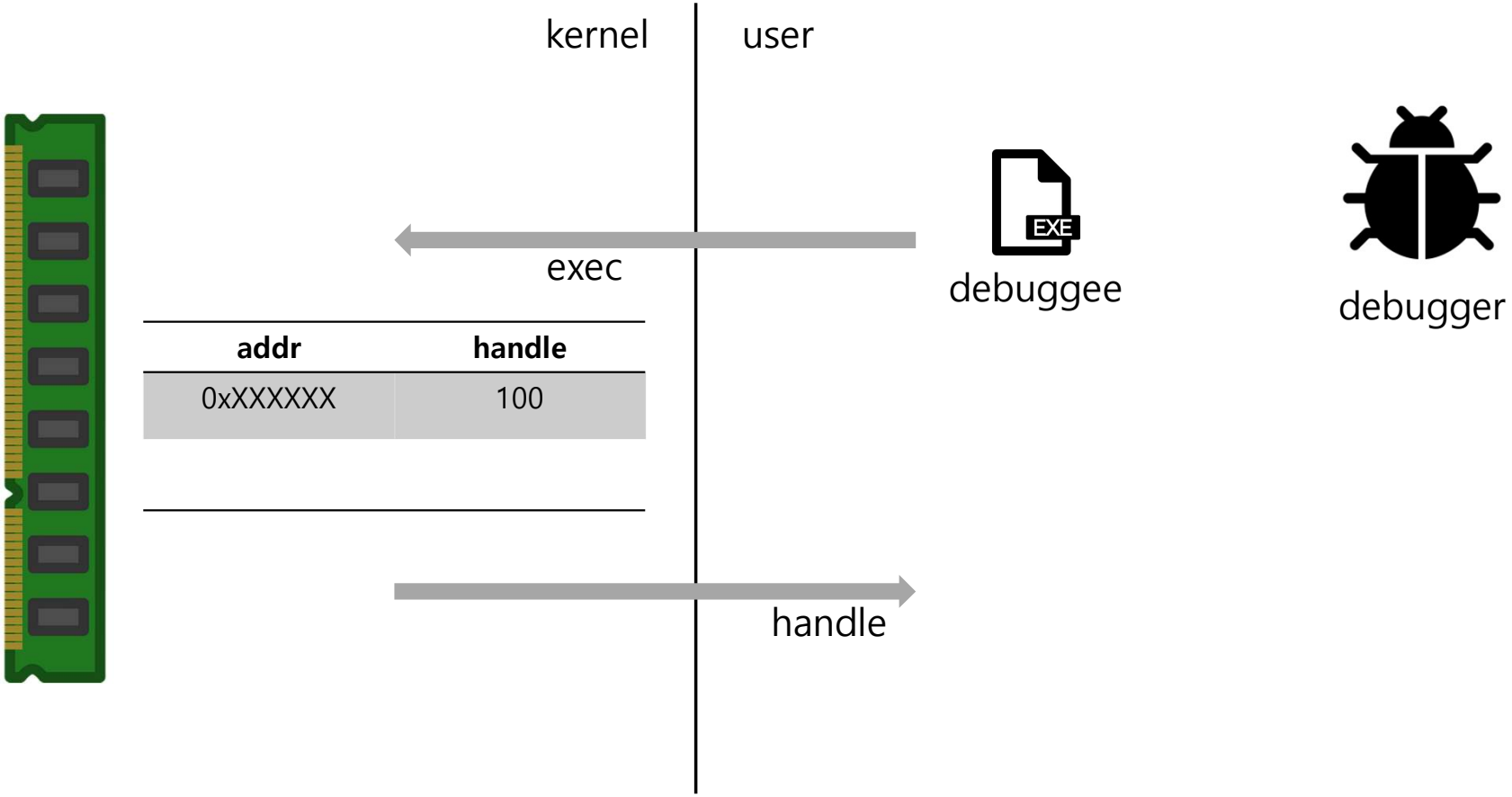
debuggee context와 handle 관리

debuggee : debugger로 debugging 할 대상



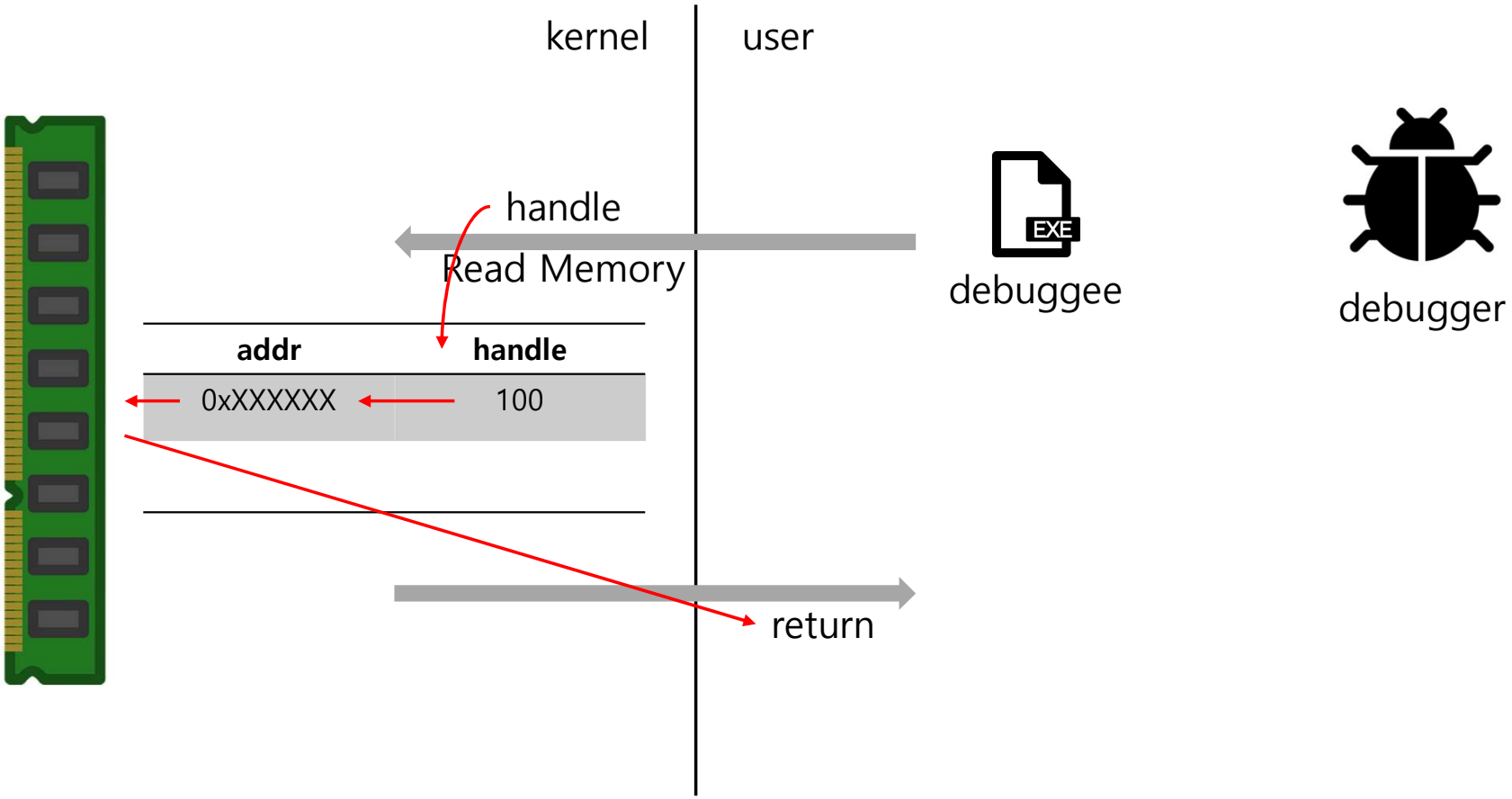
debuggee context와 handle 관리

debuggee : debugger로 debugging 할 대상



debuggee context와 handle 관리

debuggee : debugger로 debugging 할 대상



<exec.h>

```
#pragma once
#include <Windows.h>

class process
{
public:
process();
~process();

HANDLE m_thread;
HANDLE m_process;
LPCONTEXT m_pthreadContext;
ULONG_PTR m_processImageBase;
DWORD getTID();
DWORD getPID();

bool create(PWCHAR pwszProcessPath, bool
bSuspended = false);
ULONG_PTR imagebase();
void resume();

void close();
bool getThreadContext();
bool setThreadContext();

};
```

<exec.cpp 핵심 파트>

```
if (!CreateProcess(NULL, pwszProcessPath, NULL, NULL, FALSE, /*CreateFlag | */ CREATE_NEW_CONSOLE  |
DEBUG_ONLY_THIS_PROCESS | DEBUG_PROCESS, NULL, NULL, &si, &pi))
{
            return false;
}
m_thread = pi.hThread;
m_process = pi.hProcess;



bool process::getThreadContext()
{
            m_pthreadContext = (LPCONTEXT)HeapAlloc(GetProcessHeap(), 0, sizeof(CONTEXT));

            if (m_pthreadContext == NULL)
                        return false;

            memset(m_pthreadContext, 0, sizeof(CONTEXT));

            m_pthreadContext->ContextFlags = CONTEXT_FULL;
            if (!GetThreadContext(m_thread, m_pthreadContext))
            {
                        return false;
            }

            return true;
}
```

<exec.h>

```
#pragma once
#include <Windows.h>

class process
{
public:
process();
~process();

HANDLE m_thread;
HANDLE m_process;
LPCONTEXT m_pthreadContext;
ULONG_PTR m_processImageBase;
DWORD getTID();
DWORD getPID();

bool create(PWCHAR pwszProcessPath, bool
bSuspended = false);
ULONG_PTR imagebase();
void resume();

void close();
bool getThreadContext();
bool setThreadContext();

};
```

<exec.cpp 핵심 파트>

```
if (!CreateProcess(NULL, pwszProcessPath, NULL, NULL, FALSE, /*CreateFlag | */ CREATE_NEW_CONSOLE  |
DEBUG_ONLY_THIS_PROCESS | DEBUG_PROCESS, NULL, NULL, &si, &pi))
{
            return false;
}
m_thread = pi.hThread;
m_process = pi.hProcess;



bool process::getThreadContext()
{
            m_pthreadContext = (LPCONTEXT)HeapAlloc(GetProcessHeap(), 0, sizeof(CONTEXT));

            if (m_pthreadContext == NULL)
                        return false;

            memset(m_pthreadContext, 0, sizeof(CONTEXT));

            m_pthreadContext->ContextFlags = CONTEXT_FULL;
            if (!GetThreadContext(m_thread, m_pthreadContext))
            {
                        return false;
            }

            return true;
}
```

debuggee context와 handle 관리

debuggee : debugger로 debugging 할 대상

```
PROCESS_INFORMATION pi = { 0, };
```

```
<processthreadsapi.h>
typedef struct _PROCESS_INFORMATION {
    HANDLE hProcess;
    HANDLE hThread;
    DWORD dwProcessId;
    DWORD dwThreadId;
} PROCESS_INFORMATION, *PPROCESS_INFORMATION, *LPPROCESS_INFORMATION;
```

32bit env

GetProcessHeap 함수는 호출 프로세스에 대한 기본 힙에 대한 핸들을 가져옵니다.

```
LPCONTEXT m_pthreadContext;

typedef PCONTEXT LPCONTEXT;

typedef CONTEXT *PCONTEXT;
```

```
m_pthreadContext = (LPCONTEXT)HeapAlloc(GetProcessHeap(), 0, sizeof(CONTEXT));
```

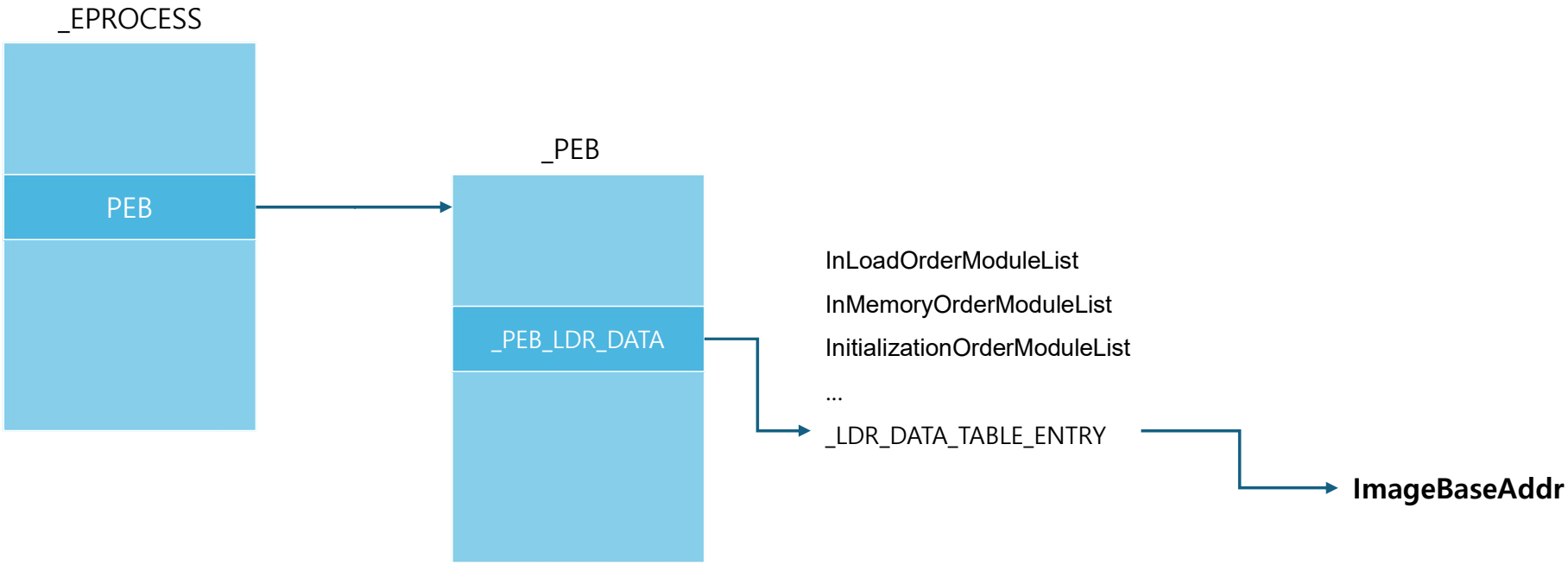
```
typedef struct DECLSPEC_NOINITALL _CONTEXT {
    DWORD ContextFlags;
    DWORD   Dr0;
    DWORD   Dr1;
    DWORD   Dr2;
    DWORD   Dr3;
    DWORD   Dr6;
    DWORD   Dr7;
    FLOATING_SAVE_AREA FloatSave;
    DWORD   SegGs;
    DWORD   SegFs;
    DWORD   SegEs;
    DWORD   SegDs;
    DWORD   Edi;
    DWORD   Esi;
    DWORD   Ebx;
    DWORD   Edx;
    DWORD   Ecx;
    DWORD   Eax;

    DWORD   Ebp;
    DWORD   Eip;
    DWORD   SegCs;              // MUST BE SANITIZED
    DWORD   EFlags;             // MUST BE SANITIZED
    DWORD   Esp;
    DWORD   SegSs;
    BYTE    ExtendedRegisters[MAXIMUM_SUPPORTED_EXTENSION];
} CONTEXT;
```

Ref : https://learn.microsoft.com/ko-kr/windows/win32/api/heapapi/nf-heapapi-getprocessheap

PEB

PEB : Process Env Block

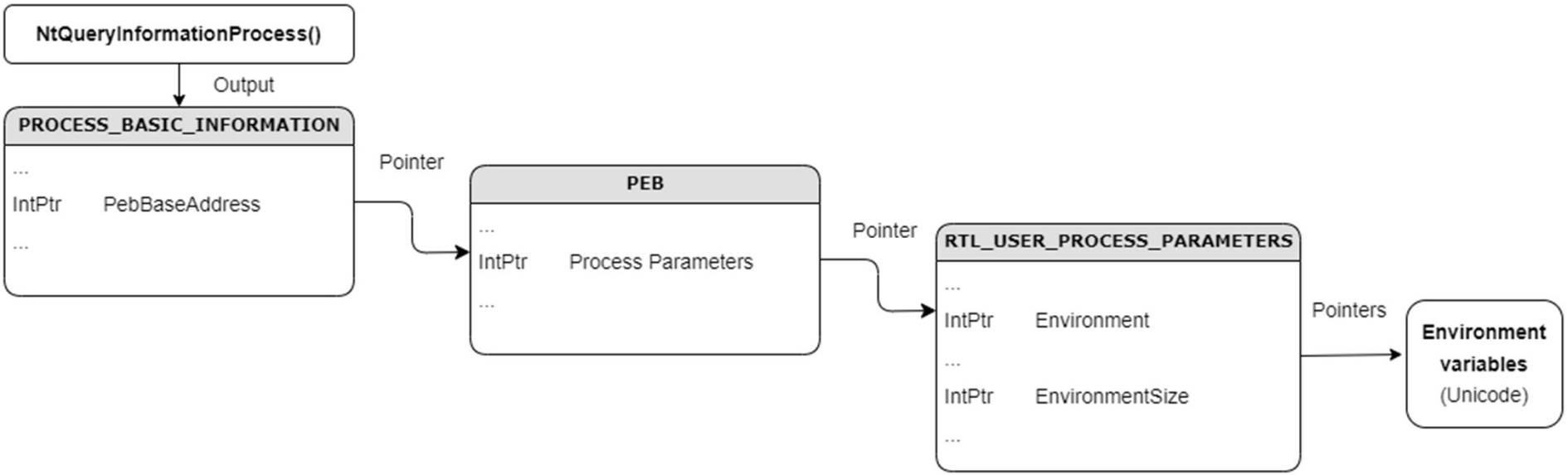


```
void* get_peb() {
    void* peb;
    __asm__(
        "movl %%fs:0x30, %0"  // TEB의 주소에서 PEB 주소를 가져옴
        : "=r"(peb)           // peb에 결과 저장
    );
    return peb;
}
```

```
void* get_peb() {
    void* peb;
    __asm__(
        "mov %%gs:0x60, %0"  // 64비트 환경에서는 GS 레지스터 사용
        : "=r"(peb)
    );
    return peb;
}
```



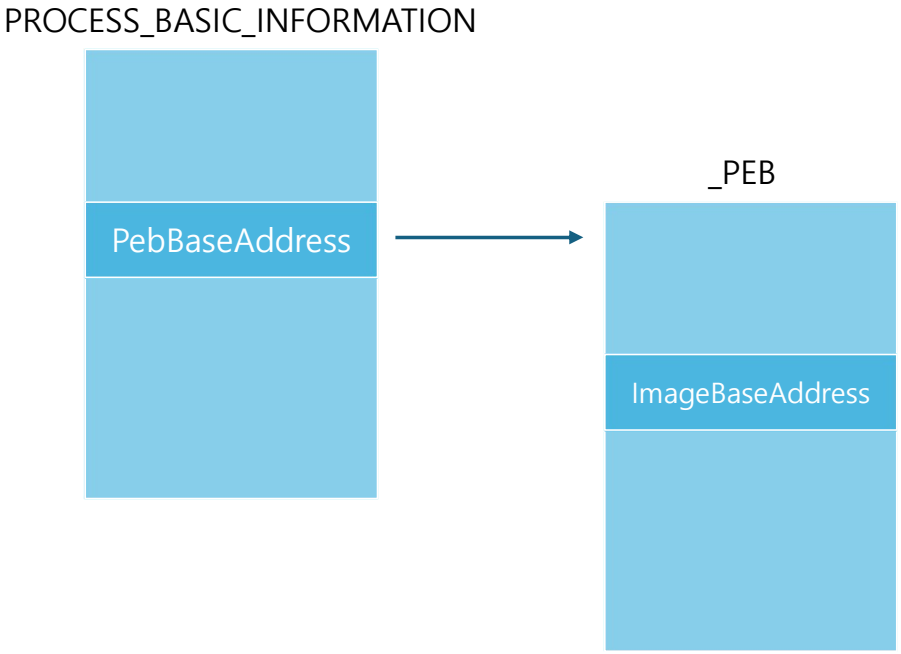
Windows API



Ref : https://learn.microsoft.com/en-us/windows/win32/api/winternl/nf-winternl-ntqueryinformationprocess

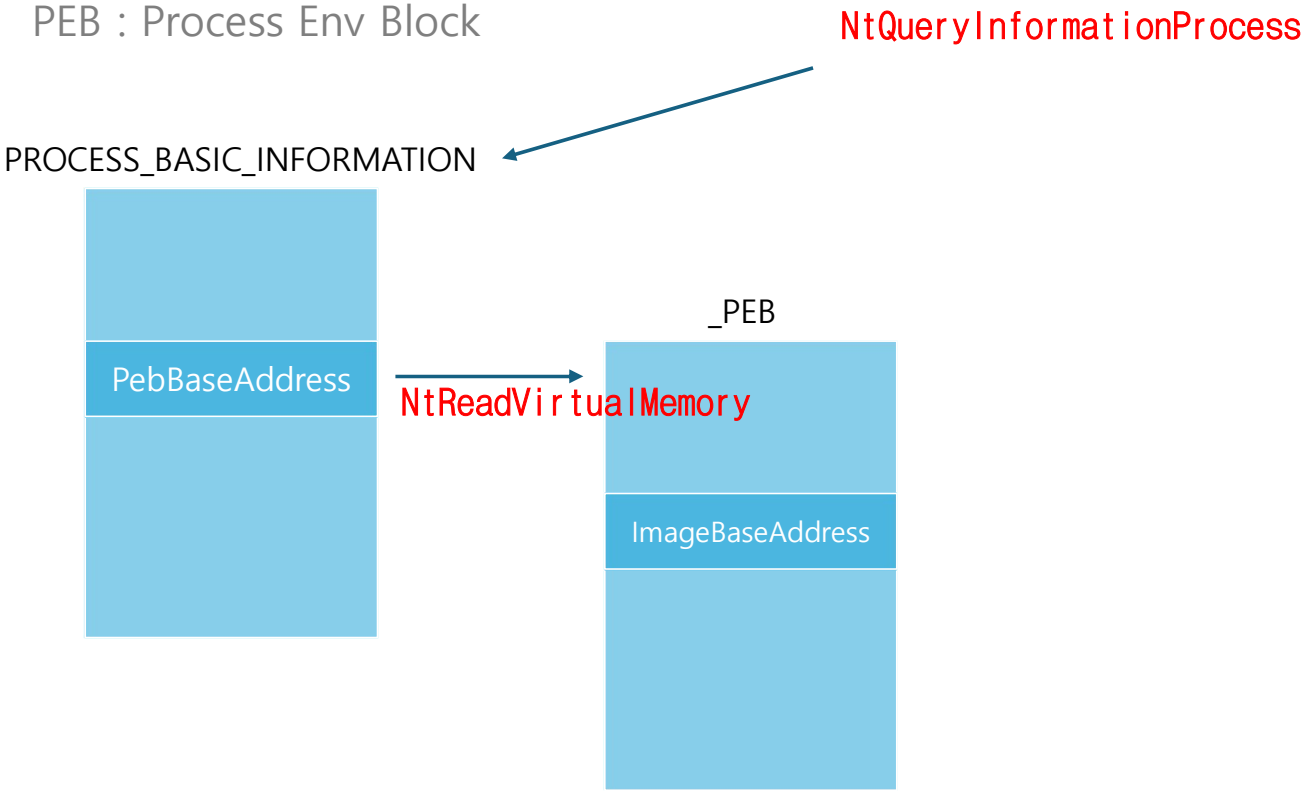
PEB

PEB : Process Env Block



PEB

PEB : Process Env Block



<exec.cpp 핵심 파트>

```
status = NtQueryInformationProcess(
            m_process,
            ProcessBasicInformation,
            &ProcessBasicInfo,
            sizeof(PROCESS_BASIC_INFORMATION),
            &ulProcessInfoLength
);

if (status != STATUS_SUCCESS)
{
            return false;
}

status = NtReadVirtualMemory(
            m_process,
            (PVOID)ProcessBasicInfo.PebBaseAddress,
            &ProcessPeb,
            sizeof(ProcessPeb),
            (PULONG)&ulPebLength
);

if (status != STATUS_SUCCESS)
{
            return false;
}

m_processImageBase = (ULONG_PTR)ProcessPeb.ImageBaseAddress;

return true;
```

요구사항 :

- ~~PE parser~~
- ~~Disassemble~~
- ~~Memory View~~
- BreakPoint

Plan :

- ~~Windows Structure 를 공부하여 pe.h 제작~~
- ~~Disassemble lib으로 capstone.h을 사용~~
- ~~Debugee Process create, virtual memory R/W 용 exec.h~~
- Memory write로 int3를 injection

요구사항 :

- **BreakPoint**
  - Memory write로 int3를 injection

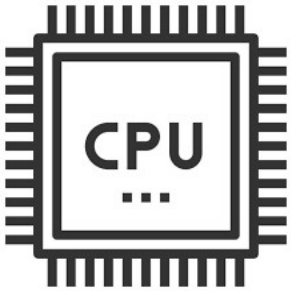
INT는 x86 중앙 처리 장치를 위한 어셈블리어로서, 인터럽트를 발생시키는 역할을 한다

INT 3 :

Opcode : CC
1byte 명령어, 1byte 이기에 기존 명령어에 patch하기 좋음(복구도 용이)
Breakpoint interrupt를 트리거 함

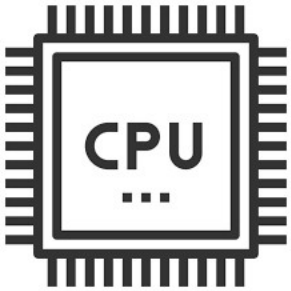
In linux,
__asm__("int3"); -> signal SIGTRAP, Trace/breakpoint trap.

In Windows?



INT 3

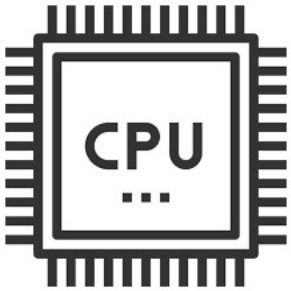
Interrupt Descriptor Table



INT 3

Interrupt Descriptor Table

Breakpoint Exception 핸들러 호출
+제어권이 커널로 넘어감



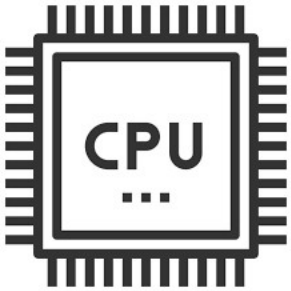
INT 3

Interrupt Descriptor Table

Breakpoint Exception 핸들러 호출
+제어권이 커널로 넘어감



STATUS_BREAKPOINT (0x800000**03**)



INT 3

Interrupt Descriptor Table

Breakpoint Exception 핸들러 호출
+제어권이 커널로 넘어감



STATUS_BREAKPOINT (0x80000003)

userland

SEH (Structured Exception Handling)

```
__try {
    __asm { int 3 } // 예외 발생 (int3 명령)
}
__except (EXCEPTION_EXECUTE_HANDLER) {
    printf("Breakpoint exception caught!₩n");
}
```

핸들링 우선순위

1.디버거가 연결된 경우:

- int3가 발생하면 Windows는 먼저 디버거로 예외를 전달합니다.
- 디버거는 이를 통해 프로그램 실행을 중단시키고 상태를 조사할 수 있습니다.
- 디버거가 예외를 처리하고 **DBG_CONTINUE**를 반환하면 프로그램은 정상적으로 계속 실행됩니다.

2.디버거가 없는 경우:

- 예외가 SEH 핸들링 체인으로 전달됩니다.
- 프로그램 내에서 SEH 핸들러를 정의하지 않으면 기본 핸들러가 호출되어 프로그램이 종료됩니다.



STATUS_BREAKPOINT (0x800000**03**)

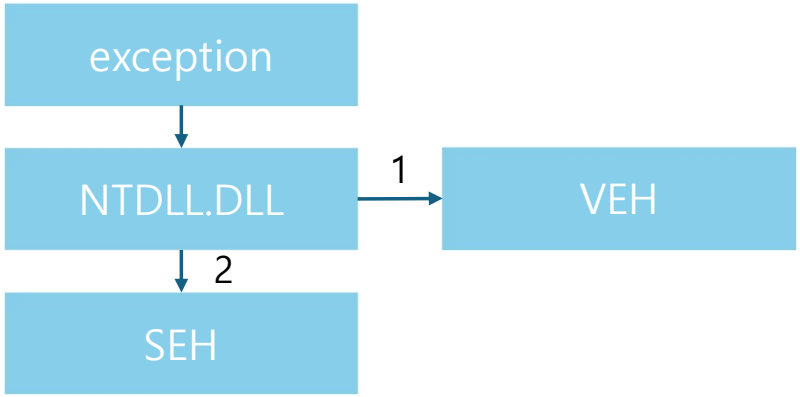
userland

~~SEH (Structured Exception Handling)~~

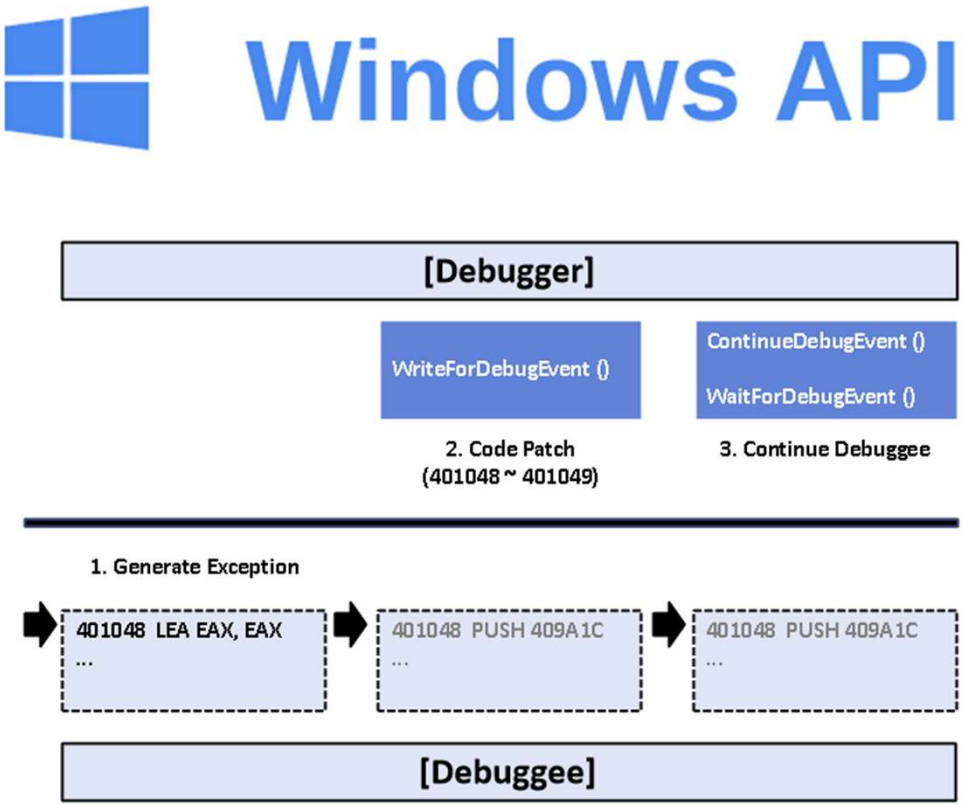
Debugger handle

1. **디버거** (디버거가 연결된 경우)
2. **VEH** (디버거가 없거나 디버거가 예외를 넘긴 경우)
3. **SEH** (VEH가 예외를 처리하지 않은 경우)

Other… (VCH, UEH, Unwind)



Ref : https://learn.microsoft.com/en-us/archive/msdn-magazine/2003/june/improved-system-info-and-kernel-debugging-security-and-ui-apis

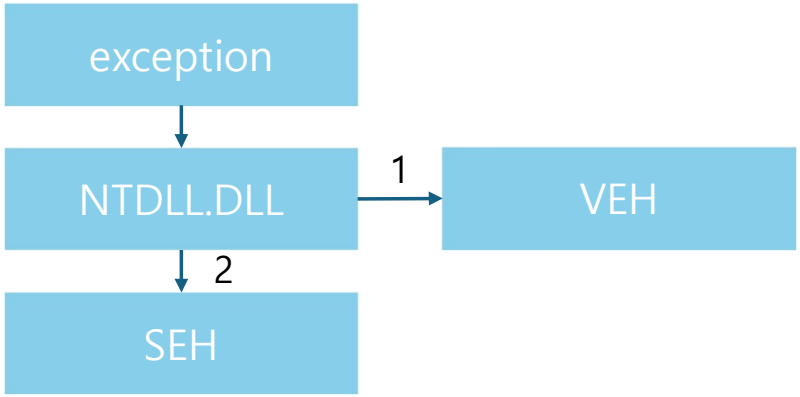


Ref : https://learn.microsoft.com/ko-kr/windows/win32/debug/debugging-events

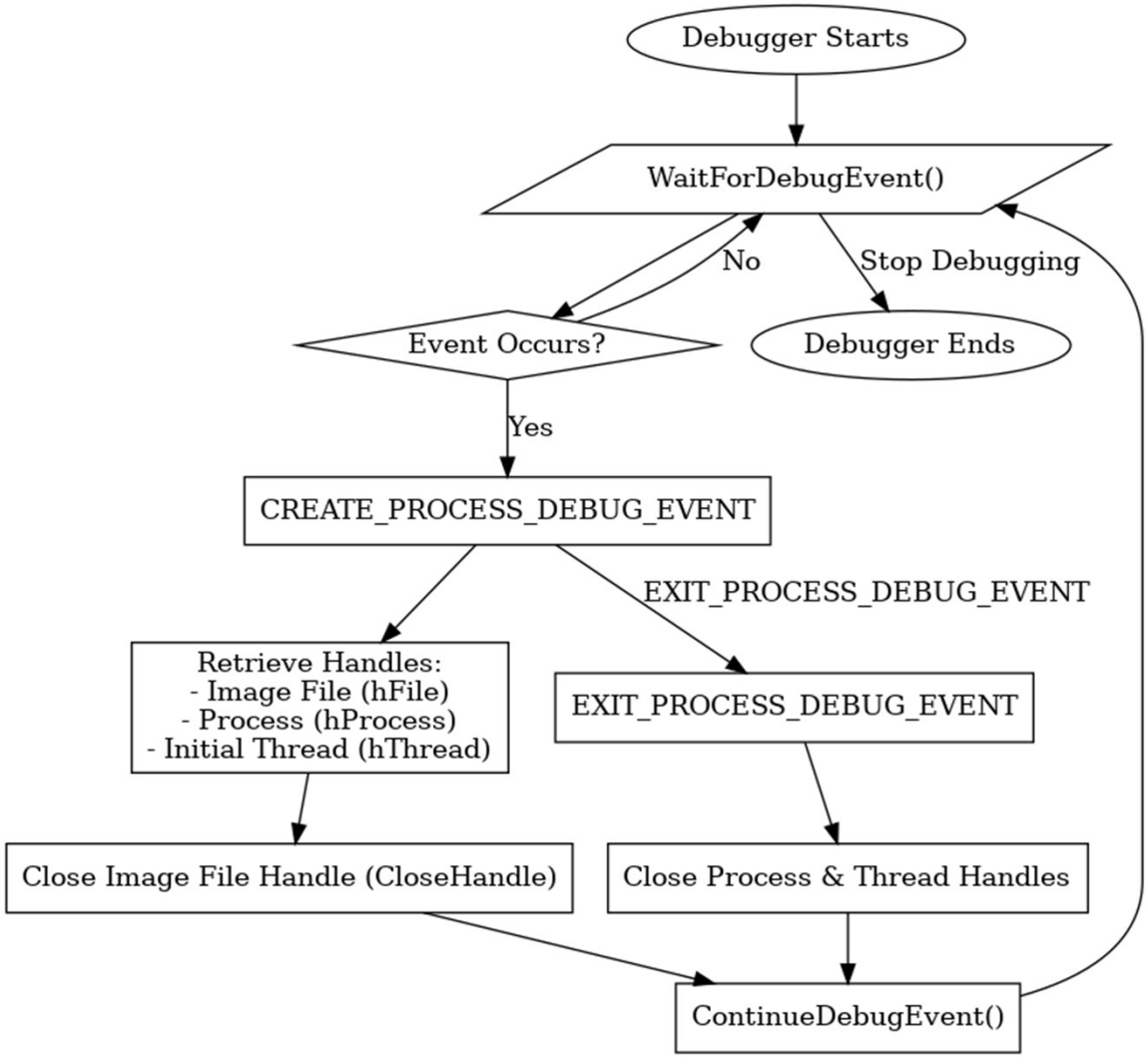
Ref : https://learn.microsoft.com/en-us/windows/win32/api/debugapi/nf-debugapi-waitfordebugevent

1. **디버거 (디버거가 연결된 경우)**
2. **VEH** (디버거가 없거나 디버거가 예외를 넘긴 경우)
3. **SEH** (VEH가 예외를 처리하지 않은 경우)

Other… (VCH, UEH, Unwind)



Ref : https://learn.microsoft.com/en-us/archive/msdn-magazine/2003/june/improved-system-info-and-kernel-debugging-security-and-ui-apis

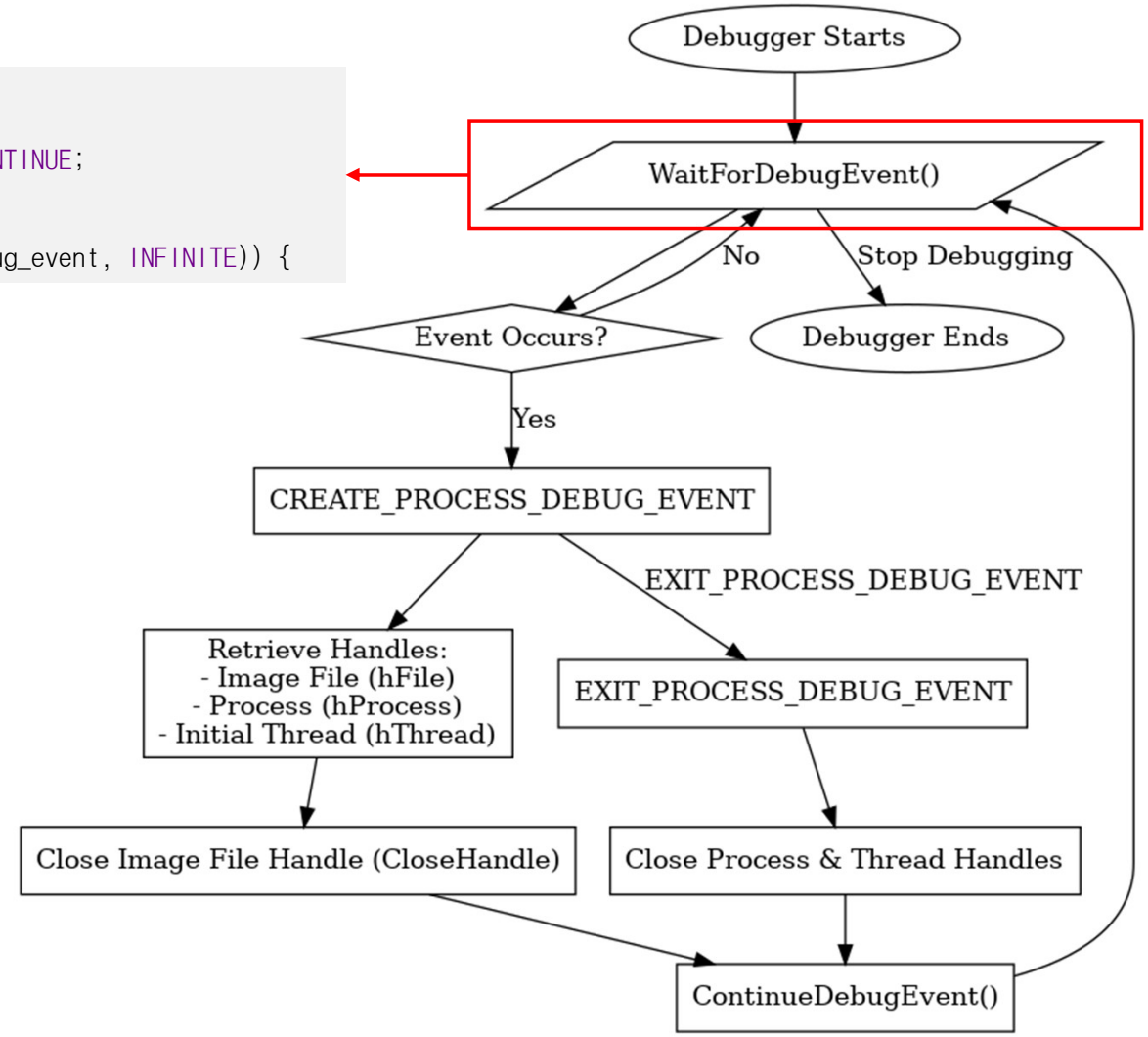

Debugger Starts
WaitForDebugEvent()
No
Stop Debugging
Event Occurs?
Debugger Ends
Yes
CREATE_PROCESS_DEBUG_EVENT
EXIT_PROCESS_DEBUG_EVENT
Retrieve Handles:
- Image File (hFile)
- Process (hProcess)
- Initial Thread (hThread)
EXIT_PROCESS_DEBUG_EVENT
Close Image File Handle (CloseHandle)
Close Process & Thread Handles
ContinueDebugEvent()

debugger.cpp

```
void init_debugger() {

    DWORD continue_status = DBG_CONTINUE;

    while (true) {
        if (WaitForDebugEvent(&debug_event, INFINITE)) {
```



debugger.cpp

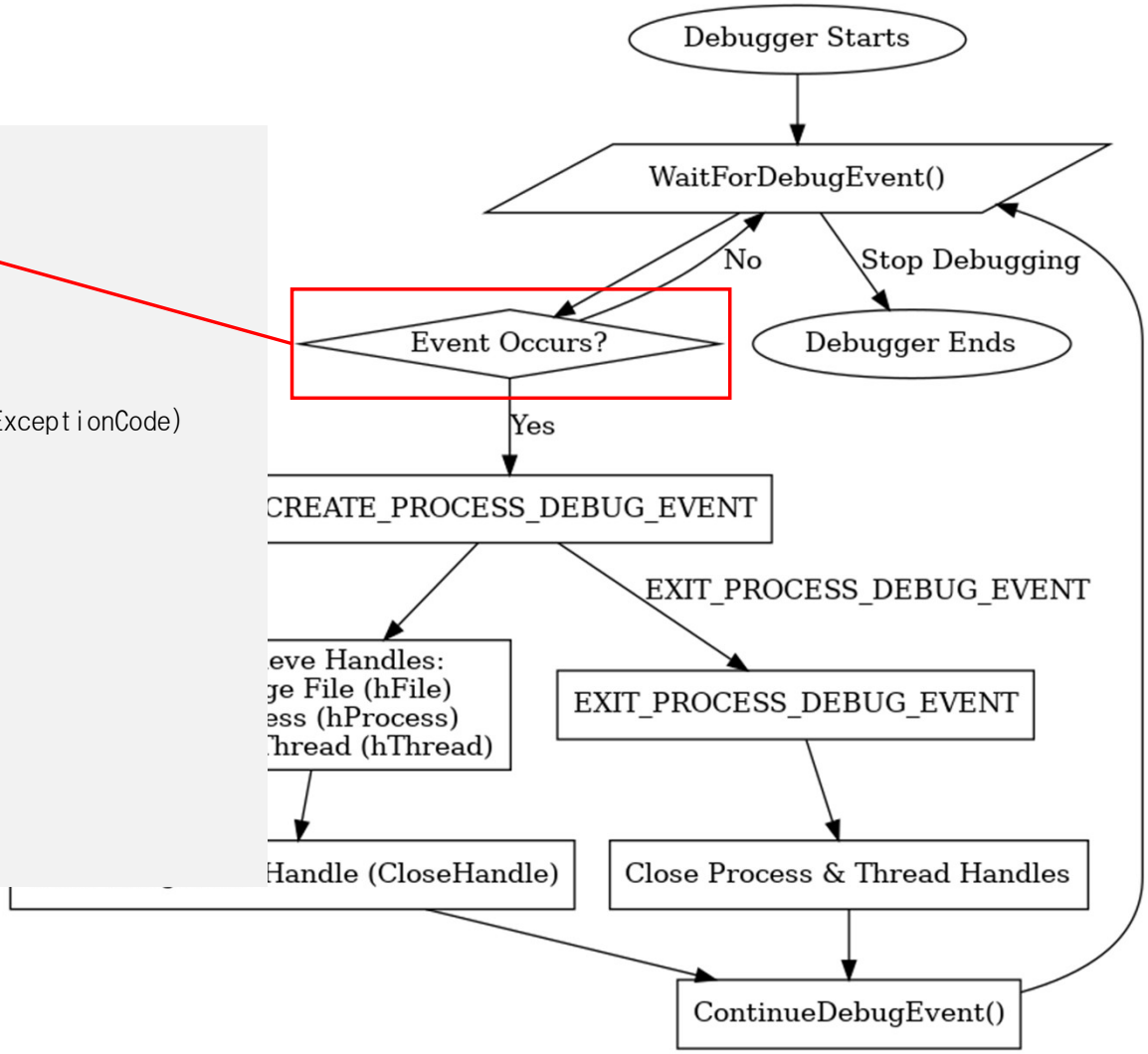
```
switch (debug_event.dwDebugEventCode)
            {
            case EXCEPTION_DEBUG_EVENT:
#ifdef DEBUG
                printf("EXCEPTION_DEBUG_EVENT₩n");
#endif



                switch (debug_event.u.Exception.ExceptionRecord.ExceptionCode)
                {
                case EXCEPTION_ACCESS_VIOLATION:
#ifdef DEBUG
                    printf("EXCEPTION_ACCESS_VIOLATION₩n");
#endif


                    break;

                case EXCEPTION_BREAKPOINT:
                    printf("EXCEPTION_BREAKPOINT₩n");
                    is_break = 1;
                    return;
                    //break;
```

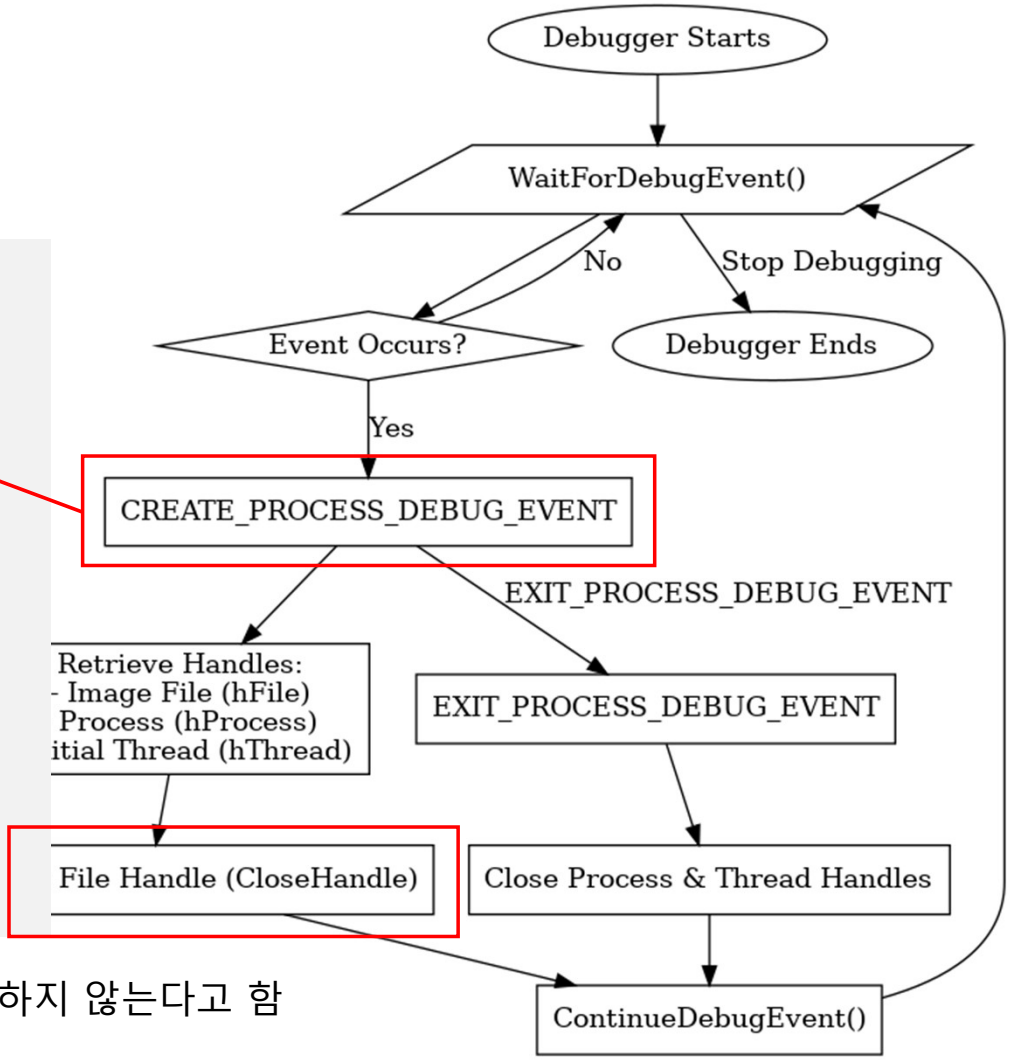


•**hFile**: 디버깅 대상 프로세스의 이미지 파일 핸들.
•**hProcess**: 디버깅 대상 프로세스 핸들.
•**hThread**: 디버깅 대상 초기 스레드 핸들.

```
 case CREATE_PROCESS_DEBUG_EVENT:
                printf("CREATE_PROCESS_DEBUG_EVENT₩n");
                printf("Base of Image: %p₩n",
debug_event.u.CreateProcessInfo.lpBaseOfImage);
#ifdef DEBUG

                // 프로세스 정보 출력
                printf("Process Handle: %p₩n",
debug_event.u.CreateProcessInfo.hProcess);
                printf("Thread Handle: %p₩n",
debug_event.u.CreateProcessInfo.hThread);

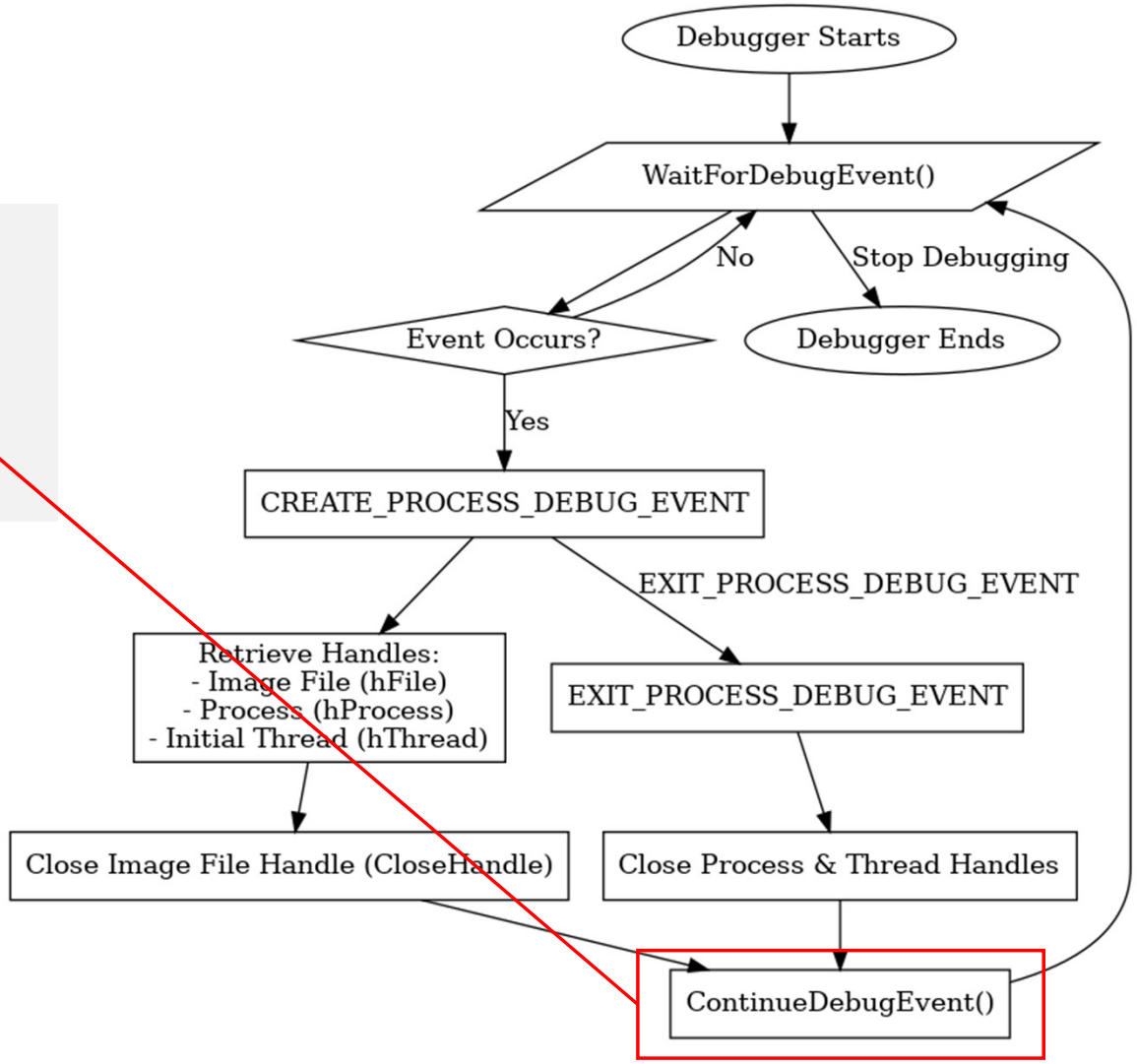
                printf("Debug Info File Handle: %p₩n",
debug_event.u.CreateProcessInfo.hFile);
#endif
                // 디버깅에 사용된 파일 핸들 닫기
                if (debug_event.u.CreateProcessInfo.hFile) {
                    CloseHandle(debug_event.u.CreateProcessInfo.hFile);
                }
                break;
```



hProcess, hThread 와 달리 hFile은 직접 close 해야 누수가 발생하지 않는다고 함
-Stack overflow 유저 의견-

debugger.cpp

```
 // Continue the process
            if (!ContinueDebugEvent(
                debug_event.dwProcessId,
                debug_event.dwThreadId,
                continue_status)) {
                printf("Failed to continue debug event.
Error: %lu₩n", GetLastError());
                break;
            }
```



🔥 DEBUG NOTE.

create process 시, DEBUG_PROCESS flag와 CREATE_SUSPENDED flag를 동시에 활성화하면
매우 신기한 일이 발생(생성된 프로세스가 hang...)

아직 분석해보진 않았으나, 추측되는 이유 :

DEBUG_PROCESS : 새로 생성된 프로세스의 디버깅 이벤트(예: 예외, 스레드 시작/종료)를 즉시 처리
CREATE_SUSPENDED : 프로세스의 주 스레드를 시작하기 전에 일시 중지 상태로 유지

WaitForDebugEvent 함수는 프로세스 초기화가 완료된 상태를 전제

*resume 이후에 wait하는 경우에 hang이 안풀리는 이유는 아직 분석 안해봤습니다.

debugger.h

```
#pragma once

// debugger.h
#ifndef DEBUGGER_H
#define DEBUGGER_H
#include "exec.h"
#include <windows.h>
#include <stdio.h>
#include <stdbool.h>
#include <map>

// External global variables
extern bool is_break;
extern DEBUG_EVENT debug_event;
extern ULONG_PTR imagebase;

extern std::map<int, ULONG_PTR> breakpoints; // Breakpoint ID와 주소를 저장
extern std::map<ULONG_PTR, BYTE> originalInstructions; // 주소와 원래 명령어 저장
extern int breakpoint_id; // Breakpoint ID를 생성하기 위한 전역 카운터

extern process targetprocess;

// Function declarations
void wait_fork();
void continue_debugger();
void init_debugger();

#endif // DEBUGGER_H
```

debugger.cpp

```
void continue_debugger() {

    ContinueDebugEvent(debug_event.dwProcessId,
debug_event.dwThreadId, DBG_CONTINUE);
    is_break = 0;
}
```

현 상황 : debuggee 의 INT 3를 탐지할 수 있다.

PLAN :

INT 3를 임의 주소에 주입할 수 있어야함.
이어서 실행되기 위해선, INT 3로 패치한 명령어를 기록해야함

PLAN :

INT 3를 임의 주소에 주입할 수 있어야함.
이어서 실행되기 위해선, INT 3로 패치한 명령어를 기록해야함

breakpoint.h

```h
#pragma once
#ifndef BREAKPOINT_H
#define BREAKPOINT_H

#include <Windows.h>
#include <iostream>
#include <map>
#include <cstdint>
#include "exec.h"
// 전역 변수
extern std::map<int, ULONG_PTR> breakpoints; // Breakpoint ID와 주소를 저장
extern std::map<ULONG_PTR, BYTE> originalInstructions; // 주소와 원래 명령어 저장
extern int breakpoint_id; // Breakpoint ID를 생성하기 위한 전역 카운터

extern process targetprocess;


// 함수 선언
void AddBreakpoint(ULONG_PTR address);
void ListBreakpoints();
void RemoveBreakpoint(int id);

#endif // BREAKPOINT_H
```

```
void AddBreakpoint(ULONG_PTR address) {
    targetprocess.getThreadContext();
    MEMORY_BASIC_INFORMATION mbi;
    if (VirtualQueryEx(targetprocess.m_process, reinterpret_cast<LPCVOID>(address), &mbi, sizeof(mbi)) == 0) {
        std::cerr << "Failed to query memory at address 0x" << std::hex << address
            << ". Error code: " << GetLastError() << std::endl;
        return;
    }
    // 메모리 상태 확인
    if (mbi.State != MEM_COMMIT) {
        std::cerr << "Address 0x" << std::hex << address << " is not committed.₩n";
        return;
    }
    if (mbi.Protect & (PAGE_NOACCESS | PAGE_GUARD)) {
        std::cerr << "Address 0x" << std::hex << address << " is protected and cannot be accessed.₩n";
        return;
    }
    // 메모리 보호 변경
    DWORD oldProtect;
    if (!VirtualProtectEx(targetprocess.m_process, reinterpret_cast<LPVOID>(address), sizeof(BYTE), PAGE_EXECUTE_READWRITE, &oldProtect)) {
        std::cerr << "Failed to change memory protection at address 0x" << std::hex << address
            << ". Error code: " << GetLastError() << std::endl;
        return;
    }
 // 기존 명령어 저장
    BYTE originalInstruction;
    SIZE_T bytesRead;
    if (!ReadProcessMemory(targetprocess.m_process, reinterpret_cast<LPCVOID>(address), &originalInstruction, sizeof(BYTE), &bytesRead) || bytesRead !=
sizeof(BYTE)) {
        std::cerr << "Failed to read original instruction at address 0x" << std::hex << address
            << ". Error code: " << GetLastError() << std::endl;
        return;
    }
    originalInstructions[address] = originalInstruction; // 저장
```

```
void AddBreakpoint(ULONG_PTR address) {
    targetprocess.getThreadContext();
    MEMORY_BASIC_INFORMATION mbi;
    if (VirtualQueryEx(targetprocess.m_process, reinterpret_cast<LPCVOID>(address), &mbi, sizeof(mbi)) == 0) {
        std::cerr << "Failed to query memory at address 0x" << std::hex << address
            << ". Error code: " << GetLastError() << std::endl;
        return;
    }
    // 메모리 상태 확인
    if (mbi.State != MEM_COMMIT) {
        std::cerr << "Address 0x" << std::hex << address << " is not committed.₩n";
        return;
    }
    if (mbi.Protect & (PAGE_NOACCESS | PAGE_GUARD)) {
        std::cerr << "Address 0x" << std::hex << address << " is protected and cannot be accessed.₩n";
        return;
    }
    // 메모리 보호 변경
    DWORD oldProtect;
    if (!VirtualProtectEx(targetprocess.m_process, reinterpret_cast<LPVOID>(address), sizeof(BYTE), PAGE_EXECUTE_READWRITE, &oldProtect)) {
        std::cerr << "Failed to change memory protection at address 0x" << std::hex << address
            << ". Error code: " << GetLastError() << std::endl;
        return;
    }
 // 기존 명령어 저장
    BYTE originalInstruction;
    SIZE_T bytesRead;
    if (!ReadProcessMemory(targetprocess.m_process, reinterpret_cast<LPCVOID>(address), &originalInstruction, sizeof(BYTE), &bytesRead) || bytesRead !=
sizeof(BYTE)) {
        std::cerr << "Failed to read original instruction at address 0x" << std::hex << address
            << ". Error code: " << GetLastError() << std::endl;
        return;
    }
    originalInstructions[address] = originalInstruction; // 저장
```

이전 보호 상태를 oldProtect에 저장하고,
1바이트를 `PAGE_EXECUTE_READWRITE`로 지정한다.
이는 RWX 모두 허용하는 속성이다.
실패 시 return 시킨다.(중단)

```
void AddBreakpoint(ULONG_PTR address) {
    targetprocess.getThreadContext();
    MEMORY_BASIC_INFORMATION mbi;
    if (VirtualQueryEx(targetprocess.m_process, reinterpret_cast<LPCVOID>(address), &mbi, sizeof(mbi)) == 0) {
        std::cerr << "Failed to query memory at address 0x" << std::hex << address
            << ". Error code: " << GetLastError() << std::endl;
        return;
    }
    // 메모리 상태 확인
    if (mbi.State != MEM_COMMIT) {
        std::cerr << "Address 0x" << std::hex << address << " is not committed.₩n";
        return;
    }
    if (mbi.Protect & (PAGE_NOACCESS | PAGE_GUARD)) {
        std::cerr << "Address 0x" << std::hex << address << " is protected and cannot be accessed.₩n";
        return;
    }
    // 메모리 보호 변경
    DWORD oldProtect;
    if (!VirtualProtectEx(targetprocess.m_process, reinterpret_cast<LPVOID>(address), sizeof(BYTE), PAGE_EXECUTE_READWRITE, &oldProtect)) {
        std::cerr << "Failed to change memory protection at ad
            << ". Error code: " << GetLastError() << std::endl
        return;
    }
 // 기존 명령어 저장
    BYTE originalInstruction;
    SIZE_T bytesRead;
    if (!ReadProcessMemory(targetprocess.m_process, reinterpret_cast<LPCVOID>(address), &originalInstruction, sizeof(BYTE), &bytesRead) || bytesRead !=
sizeof(BYTE)) {
        std::cerr << "Failed to read original instruction at address 0x" << std::hex << address
            << ". Error code: " << GetLastError() << std::endl;
        return;
    }
    originalInstructions[address] = originalInstruction; // 저장
```

이전에 권한을 부여한 1바이트를 읽고
originalInstruction에 저장한다.

```
    // 브레이크포인트 설정
    BYTE breakpointInstruction = 0xCC;
    SIZE_T bytesWritten;

    if (!WriteProcessMemory(targetprocess.m_process, reinterpret_cast<LPVOID>(address), &breakpointInstruction, sizeof(breakpointInstruction),
&bytesWritten) || bytesWritten != sizeof(breakpointInstruction)) {
        std::cerr << "Failed to write breakpoint instruction at address 0x" << std::hex << address
            << ". Error code: " << GetLastError() << std::endl;

        // 메모리 보호 원상 복구
        VirtualProtectEx(targetprocess.m_process, reinterpret_cast<LPVOID>(address), sizeof(BYTE), oldProtect, &oldProtect);
        return;
    }



    // 메모리 보호 원상 복구
    if (!VirtualProtectEx(targetprocess.m_process, reinterpret_cast<LPVOID>(address), sizeof(BYTE), oldProtect, &oldProtect)) {
        std::cerr << "Failed to restore memory protection at address 0x" << std::hex << address
            << ". Error code: " << GetLastError() << std::endl;
        return;
    }




    breakpoints[breakpoint_id] = address;
    std::cout << "Breakpoint " << breakpoint_id << " added at address: 0x" << std::hex << address << "₩n";
    ++breakpoint_id;
}
```

```
    // 브레이크포인트 설정
    BYTE breakpointInstruction = 0xCC;
    SIZE_T bytesWritten;

    if (!WriteProcessMemory(targetprocess.m_process, reinterpret_cast<LPVOID>(address), &breakpointInstruction, sizeof(breakpointInstruction),
&bytesWritten) || bytesWritten != sizeof(breakpointInstruction)) {
        std::cerr << "Failed to write breakpoint instruction at address 0x" << std::hex << address
            << ". Error code: " << GetLastError() << std::endl;

        // 메모리 보호 원상 복구
        VirtualProtectEx(targetprocess.m_process, reinterpret_cast<LPVOID>(address), sizeof(BYTE), oldProtect, &oldProtect);
        return;
    }

    // 메모리 보호 원상 복구
    if (!VirtualProtectEx(targetprocess.m_process, reinterpret_cast<
        std::cerr << "Failed to restore memory protection at address 0x" << std::hex << address
            << ". Error code: " << GetLastError() << std::endl;
        return;
    }

    breakpoints[breakpoint_id] = address;
    std::cout << "Breakpoint " << breakpoint_id << " added at address: 0x" << std::hex << address << "₩n";
    ++breakpoint_id;
}
```

1바이트 위치에 0xCC를 쓰고 권한을 oldProtect로 원상 복구한다.

```
    // 브레이크포인트 설정
    BYTE breakpointInstruction = 0xCC;
    SIZE_T bytesWritten;

    if (!WriteProcessMemory(targetprocess.m_process, reinterpret_cast<LPVOID>(address), &breakpointInstruction, sizeof(breakpointInstruction),
&bytesWritten) || bytesWritten != sizeof(breakpointInstruction)) {
        std::cerr << "Failed to write breakpoint instruction at address 0x" << std::hex << address
            << ". Error code: " << GetLastError() << std::endl;

        // 메모리 보호 원상 복구
        VirtualProtectEx(targetprocess.m_process, reinterpret_cast<LPVOID>(address), sizeof(BYTE), oldProtect, &oldProtect);
        return;
    }



    // 메모리 보호 원상 복구
    if (!VirtualProtectEx(targetprocess.m_process, reinterpret_cast<LPVOID>(address), sizeof(BYTE), oldProtect, &oldProtect)) {
        std::cerr << "Failed to restore memory protection at address 0x" << std::hex << address
            << ". Error code: " << GetLastError() << std::endl;
        return;
    }

    breakpoints[breakpoint_id] = address;
    std::cout << "Breakpoint " << breakpoint_id << " added at address: 0x" << std::hex << address << "₩n";
    ++breakpoint_id;
}
```

Write 실패하더라도 memory protection은
원상 복구 시켜야 한다.

```
    // 브레이크포인트 설정
    BYTE breakpointInstruction = 0xCC;
    SIZE_T bytesWritten;

    if (!WriteProcessMemory(targetprocess.m_process, reinterpret_cast<LPVOID>(address), &breakpointInstruction, sizeof(breakpointInstruction),
&bytesWritten) || bytesWritten != sizeof(breakpointInstruction)) {
        std::cerr << "Failed to write breakpoint instruction at address 0x" << std::hex << address
            << ". Error code: " << GetLastError() << std::endl;

        // 메모리 보호 원상 복구
        VirtualProtectEx(targetprocess.m_process, reinterpret_cast<LPVOID>(address), sizeof(BYTE), oldProtect, &oldProtect);
        return;
    }

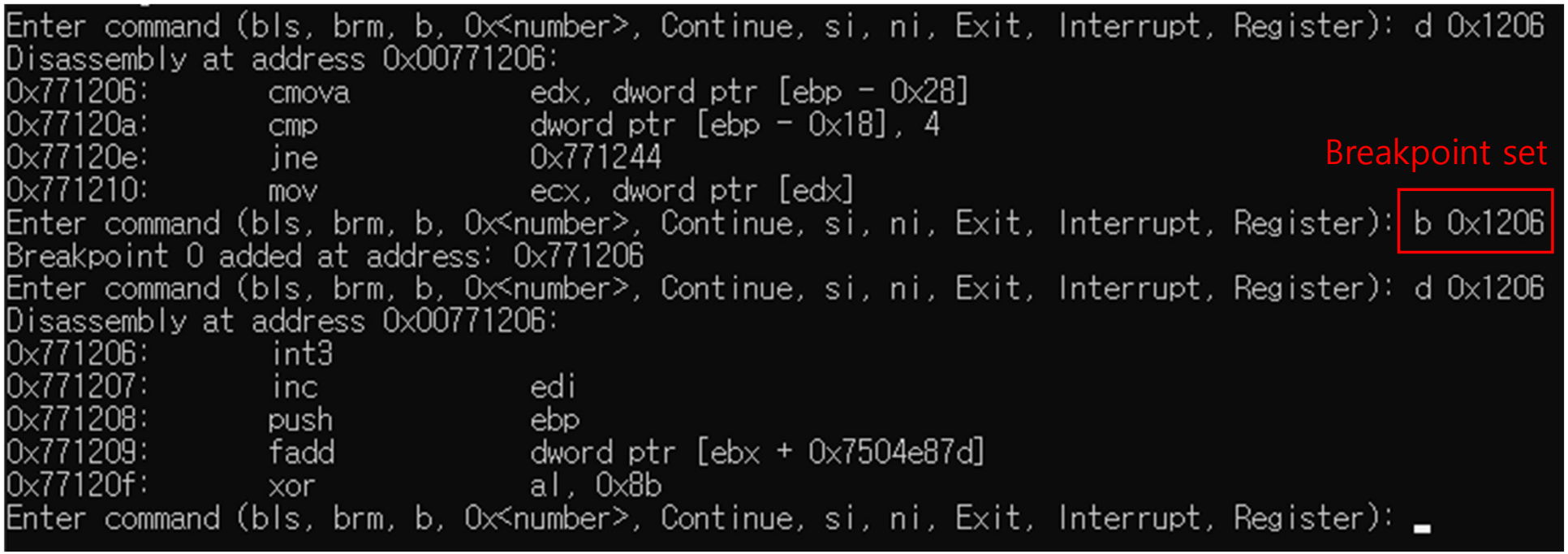


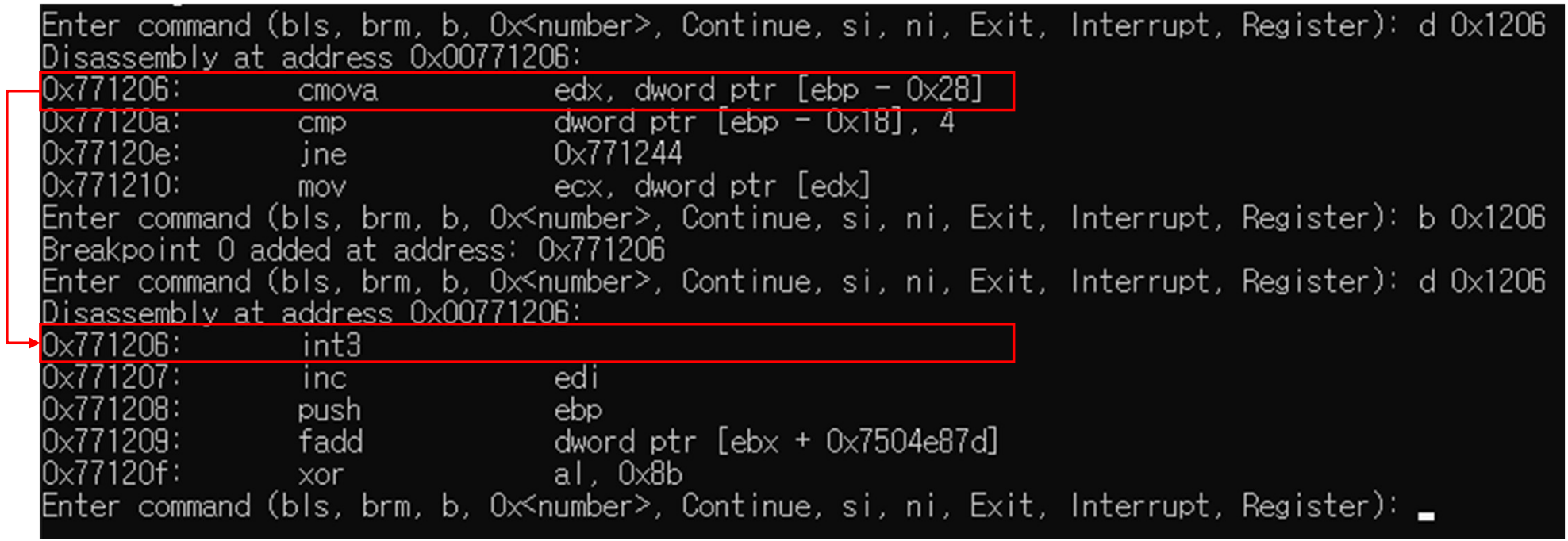
    // 메모리 보호 원상 복구
    if (!VirtualProtectEx(targetprocess.m_process, reinterpret_cast<LPVOID>(address), sizeof(BYTE), oldProtect, &oldProtect)) {
        std::cerr << "Failed to restore memory protection at address 0x" << std::hex << address
            << ". Error code: " << GetLastError() << std::endl;
        return;
    }



    breakpoints[breakpoint_id] = address;
    std::cout << "Breakpoint " << breakpoint_id << " added at address: 0x" << std::hex << address << "\n";
    ++breakpoint_id;
}
```

Breakpoints[고유 id]=<주소>
벡터로 저장한다.


Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): d 0x1206
Disassembly at address 0x00771206:
0x771206: cmova edx, dword ptr [ebp - 0x28]
0x77120a: cmp dword ptr [ebp - 0x18], 4
0x77120e: jne 0x771244
0x771210: mov ecx, dword ptr [edx]
Breakpoint set
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): b 0x1206
Breakpoint 0 added at address: 0x771206
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): d 0x1206
Disassembly at address 0x00771206:
0x771206: int3
0x771207: inc edi
0x771208: push ebp
0x771209: fadd dword ptr [ebx + 0x7504e87d]
0x77120f: xor al, 0x8b
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register):


Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): d 0x1206
Disassembly at address 0x00771206:
0x771206: cmova edx, dword ptr [ebp - 0x28]
0x77120a: cmp dword ptr [ebp - 0x18], 4
0x77120e: jne 0x771244
0x771210: mov ecx, dword ptr [edx]
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): b 0x1206
Breakpoint 0 added at address: 0x771206
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): d 0x1206
Disassembly at address 0x00771206:
0x771206: int3
0x771207: inc edi
0x771208: push ebp
0x771209: fadd dword ptr [ebx + 0x7504e87d]
0x77120f: xor al, 0x8b
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register):

main.cpp

```
 case 'c': //continue
           if (strcmp(command, "c") == 0) {
               printf("Executing 'c' command.₩n");

               is_bp = !breakpoints.empty();
               if (is_bp&&!is_break) {
                   wait_fork();
               }
               else
               {
                   continue_debugger();
               }

}
           else {
               printf("Invalid 'c' command.₩n");
           }
           break;
```

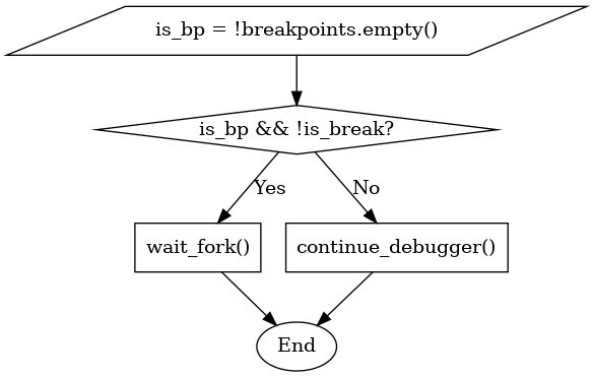
debugger.cpp

```
void continue_debugger() {

    ContinueDebugEvent(debug_event.dwProcessId,
debug_event.dwThreadId, DBG_CONTINUE);
    is_break = 0;
}
```

is_bp : 브레이크 포인트 셋팅된게 있는지?

is_break : 브레이크 포인트 HIT 된 상태인지?



브레이크 포인트가 설정된 위치가 존재하며, 브레이크 포인트에 걸려있지 않은 상태에서 CONTINUE 시 DEBUG_EVENT(INT3 HIT) 발생을 대기 -> wait_fork()

그 외, continue_debugger()

test.cpp

```cpp
#include <iostream>
#include <string>

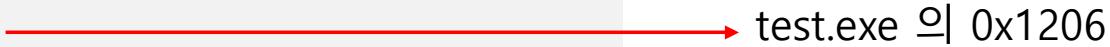
int main()
{
    std::string input; // 사용자 입력을 저장할 변수
    std::cout << "프로그램이 실행 중입니다. 종료하려면 'exit'을 입력하세요.₩n";

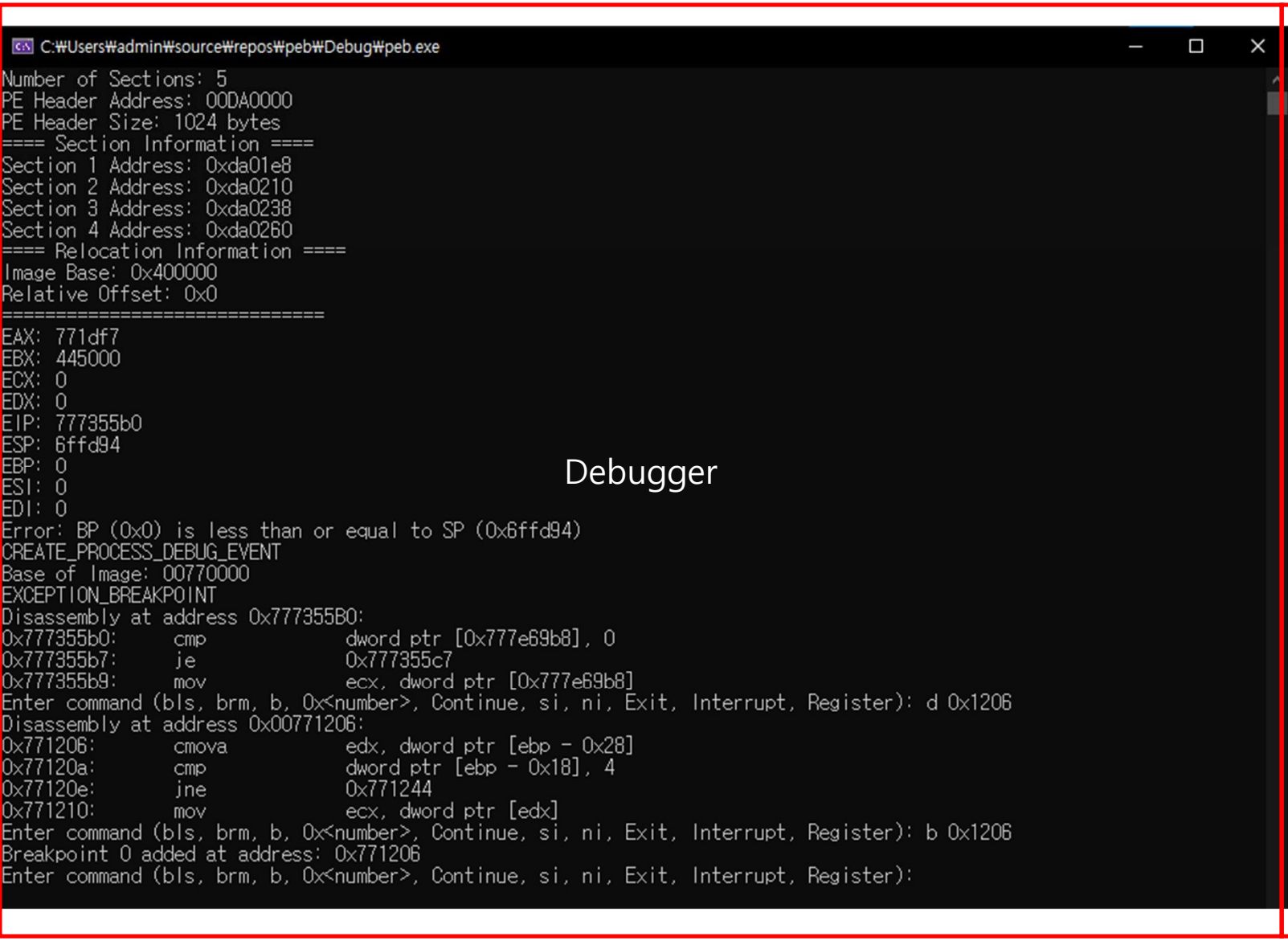
    // 입력을 지속적으로 처리
    while (true)
    {
        std::cout << "입력: ";
        std::getline(std::cin, input); // 입력 받기

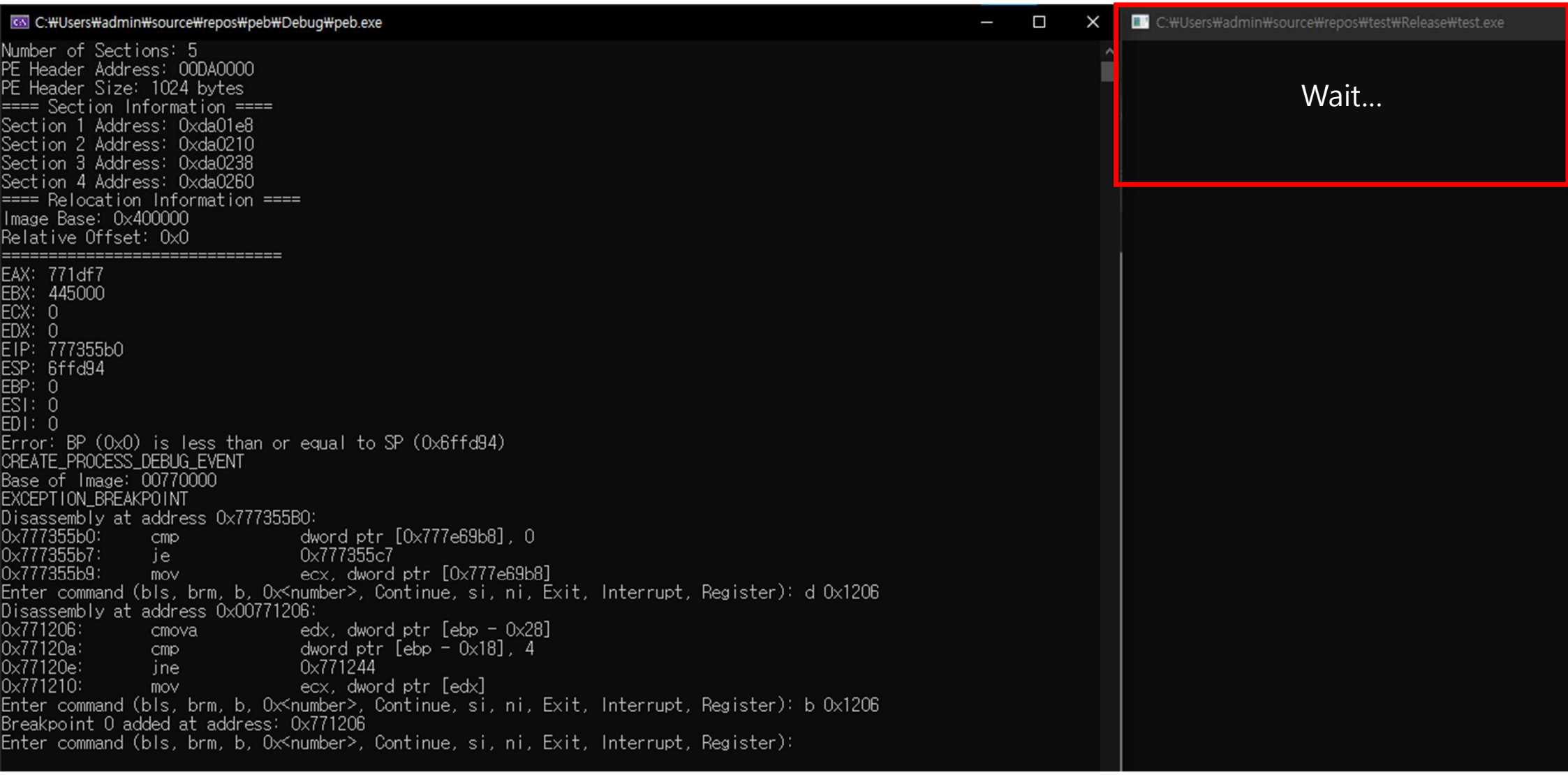
        if (input == "exit") // 'exit' 입력 시 종료
        {
            std::cout << "프로그램을 종료합니다.₩n";
            break;
        }

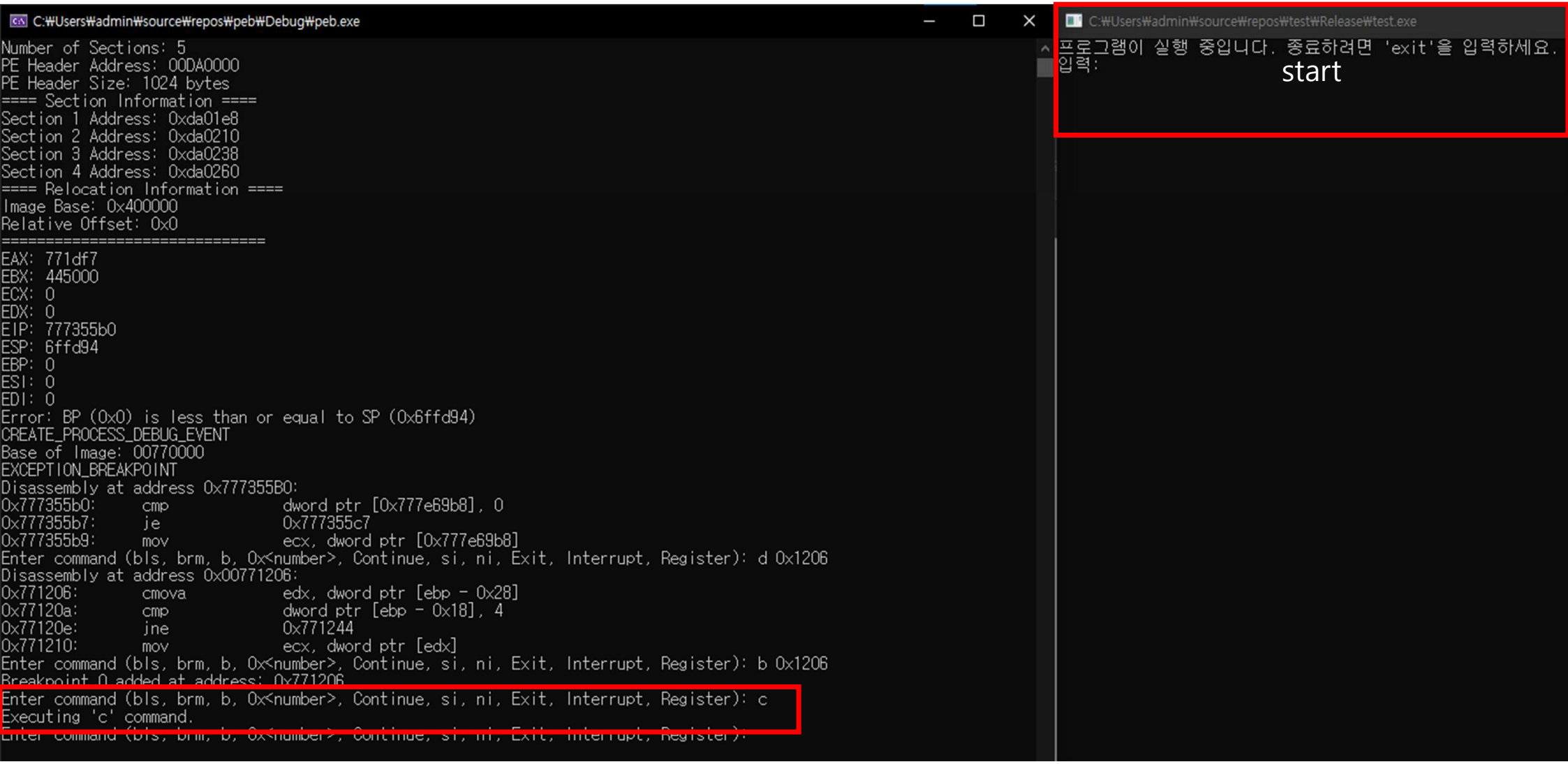
        // 입력 내용을 출력
        std::cout << "입력한 내용: " << input << "₩n";
    }

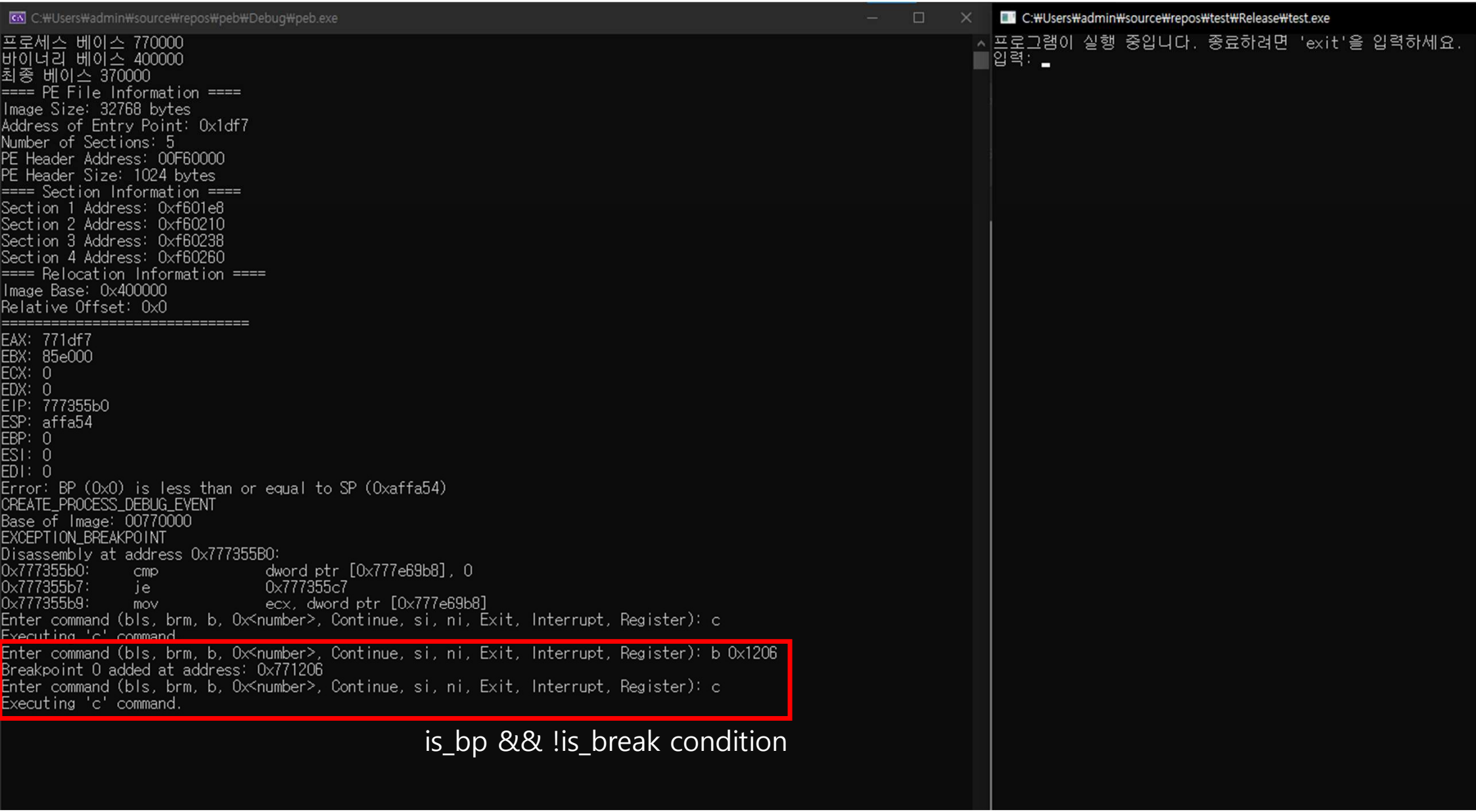
    return 0;
}
```



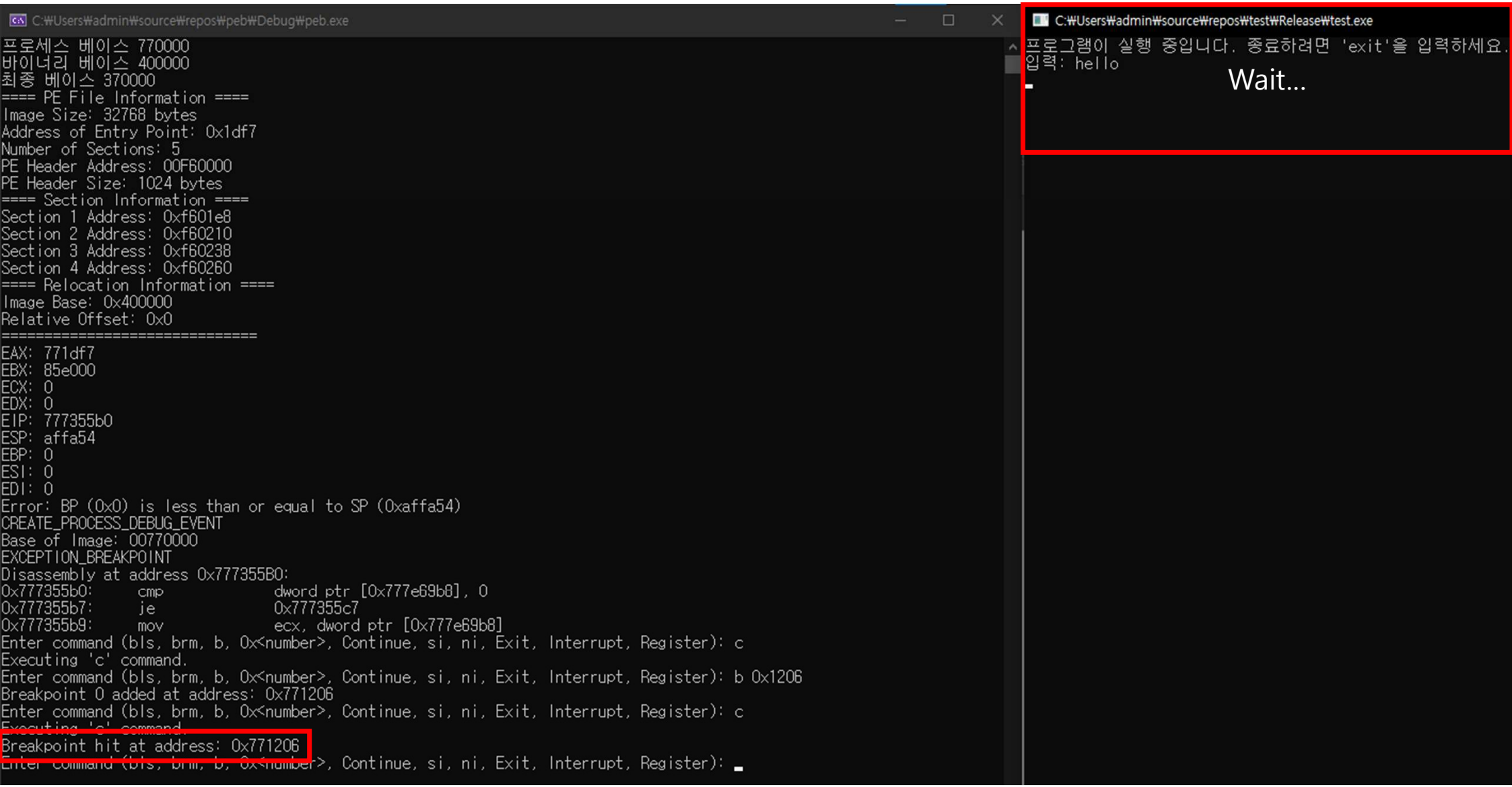

C:₩Users₩admin₩source₩repos₩peb₩Debug₩peb.exe
Number of Sections: 5
PE Header Address: 00DA0000
PE Header Size: 1024 bytes
==== Section Information ====
Section 1 Address: 0xda01e8
Section 2 Address: 0xda0210
Section 3 Address: 0xda0238
Section 4 Address: 0xda0260
==== Relocation Information ====
Image Base: 0x400000
Relative Offset: 0x0
==========
EAX: 771df7
EBX: 445000
ECX: 0
EDX: 0
EIP: 777355b0
ESP: 6ffd94
EBP: 0
ESI: 0
EDI: 0
Error: BP (0x0) is less than or equal to SP (0x6ffd94)
CREATE_PROCESS_DEBUG_EVENT
Base of Image: 00770000
EXCEPTION_BREAKPOINT
Disassembly at address 0x777355B0:
0x777355b0: cmp dword ptr [0x777e69b8], 0
0x777355b7: je 0x777355c7
0x777355b9: mov ecx, dword ptr [0x777e69b8]
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): d 0x1206
Disassembly at address 0x00771206:
0x771206: cmova edx, dword ptr [ebp - 0x28]
0x77120a: cmp dword ptr [ebp - 0x18], 4
0x77120e: jne 0x771244
0x771210: mov ecx, dword ptr [edx]
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): b 0x1206
Breakpoint 0 added at address: 0x771206
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register):
Debugger
C:₩Users₩admin₩source₩repos₩test₩Release₩test.exe
Debuggee

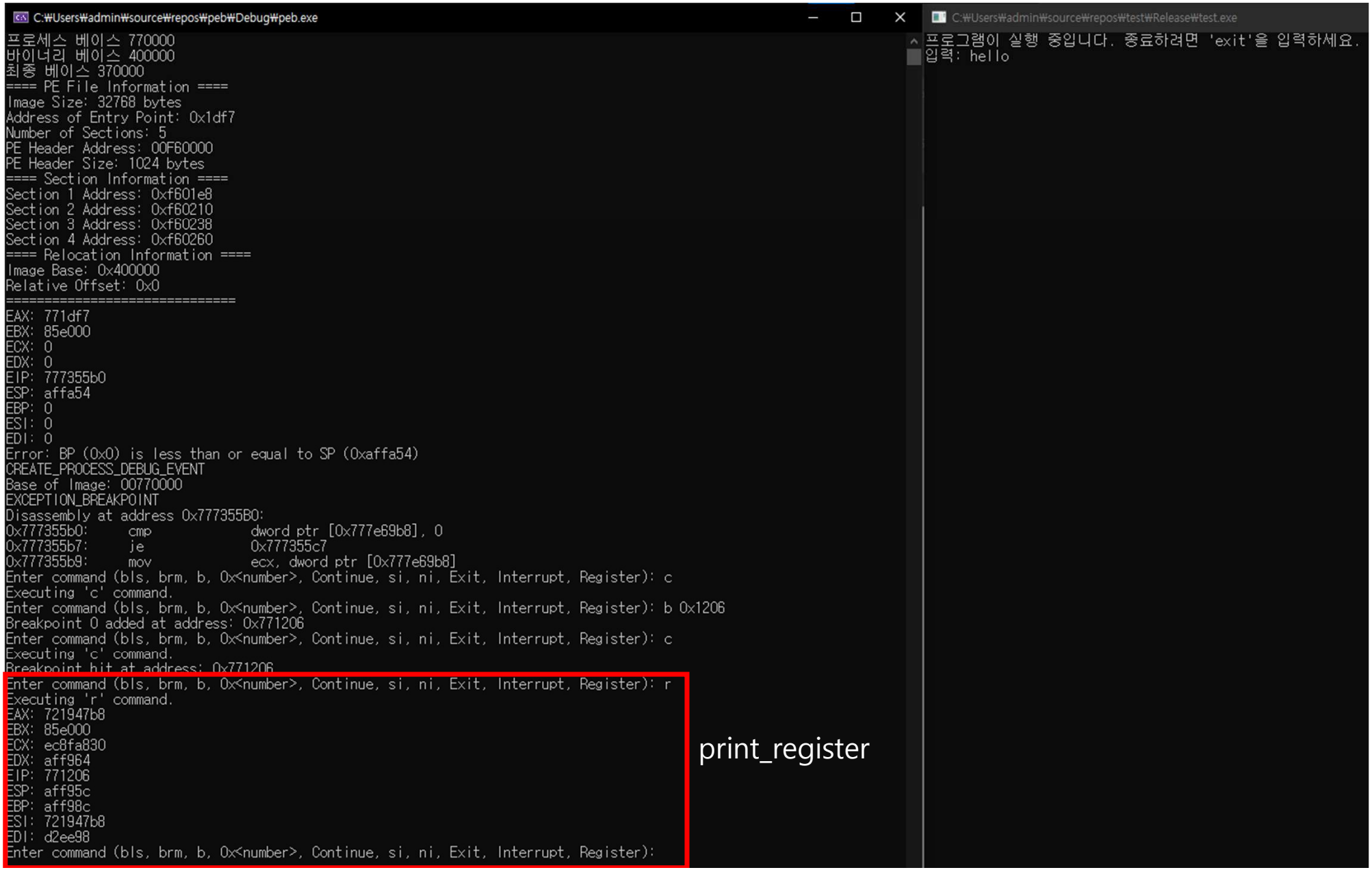

C:₩Users₩admin₩source₩repos₩peb₩Debug₩peb.exe
Number of Sections: 5
PE Header Address: 00DA0000
PE Header Size: 1024 bytes
==== Section Information ====
Section 1 Address: 0xda01e8
Section 2 Address: 0xda0210
Section 3 Address: 0xda0238
Section 4 Address: 0xda0260
==== Relocation Information ====
Image Base: 0x400000
Relative Offset: 0x0
==========
EAX: 771df7
EBX: 445000
ECX: 0
EDX: 0
EIP: 777355b0
ESP: 6ffd94
EBP: 0
ESI: 0
EDI: 0
Error: BP (0x0) is less than or equal to SP (0x6ffd94)
CREATE_PROCESS_DEBUG_EVENT
Base of Image: 00770000
EXCEPTION_BREAKPOINT
Disassembly at address 0x777355B0:
0x777355b0: cmp dword ptr [0x777e69b8], 0
0x777355b7: je 0x777355c7
0x777355b9: mov ecx, dword ptr [0x777e69b8]
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): d 0x1206
Disassembly at address 0x00771206:
0x771206: cmova edx, dword ptr [ebp - 0x28]
0x77120a: cmp dword ptr [ebp - 0x18], 4
0x77120e: jne 0x771244
0x771210: mov ecx, dword ptr [edx]
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): b 0x1206
Breakpoint 0 added at address: 0x771206
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register):
C:₩Users₩admin₩source₩repos₩test₩Release₩test.exe
Wait...

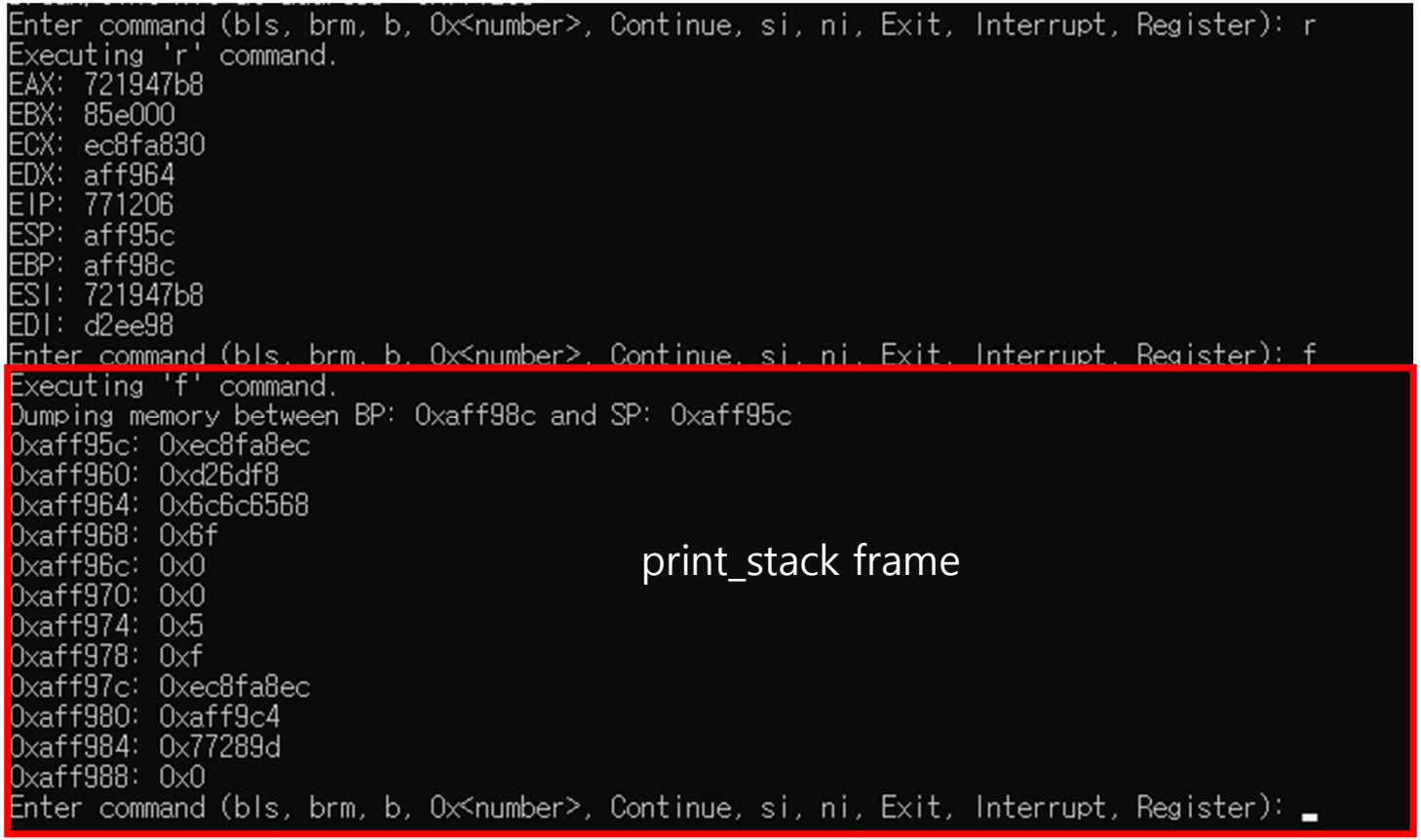

C:₩Users₩admin₩source₩repos₩peb₩Debug₩peb.exe
Number of Sections: 5
PE Header Address: 00DA0000
PE Header Size: 1024 bytes
==== Section Information ====
Section 1 Address: 0xda01e8
Section 2 Address: 0xda0210
Section 3 Address: 0xda0238
Section 4 Address: 0xda0260
==== Relocation Information ====
Image Base: 0x400000
Relative Offset: 0x0
==========
EAX: 771df7
EBX: 445000
ECX: 0
EDX: 0
EIP: 777355b0
ESP: 6ffd94
EBP: 0
ESI: 0
EDI: 0
Error: BP (0x0) is less than or equal to SP (0x6ffd94)
CREATE_PROCESS_DEBUG_EVENT
Base of Image: 00770000
EXCEPTION_BREAKPOINT
Disassembly at address 0x777355B0:
0x777355b0: cmp dword ptr [0x777e69b8], 0
0x777355b7: je 0x777355c7
0x777355b9: mov ecx, dword ptr [0x777e69b8]
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): d 0x1206
Disassembly at address 0x00771206:
0x771206: cmova edx, dword ptr [ebp - 0x28]
0x77120a: cmp dword ptr [ebp - 0x18], 4
0x77120e: jne 0x771244
0x771210: mov ecx, dword ptr [edx]
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): b 0x1206
Breakpoint 0 added at address: 0x771206
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): c
Executing 'c' command.
C:₩Users₩admin₩source₩repos₩test₩Release₩test.exe
프로그램이 실행 중입니다. 종료하려면 'exit'을 입력하세요.
입력:
start

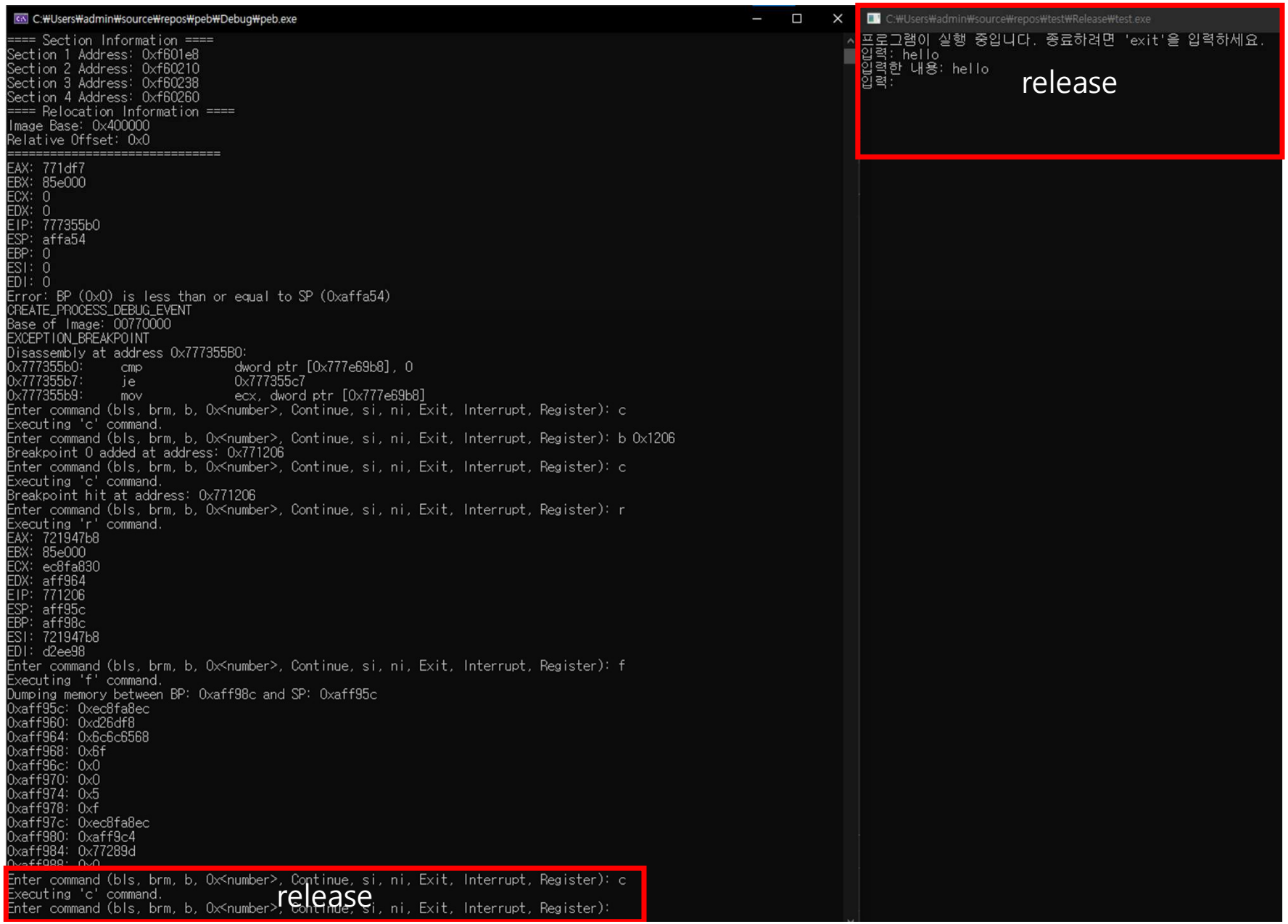


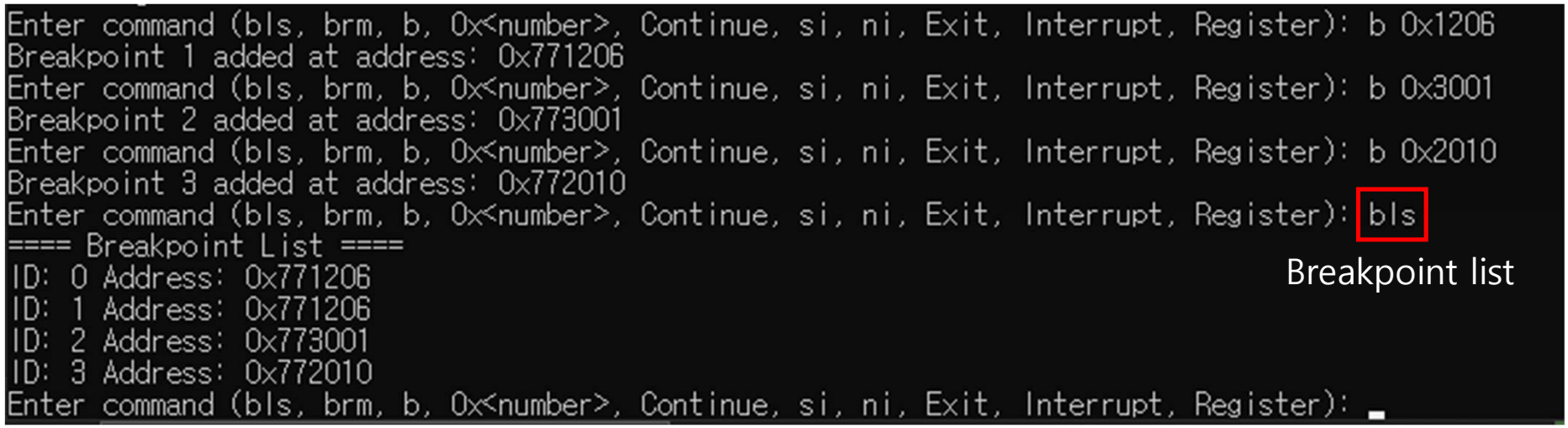
is_bp && !is_break condition

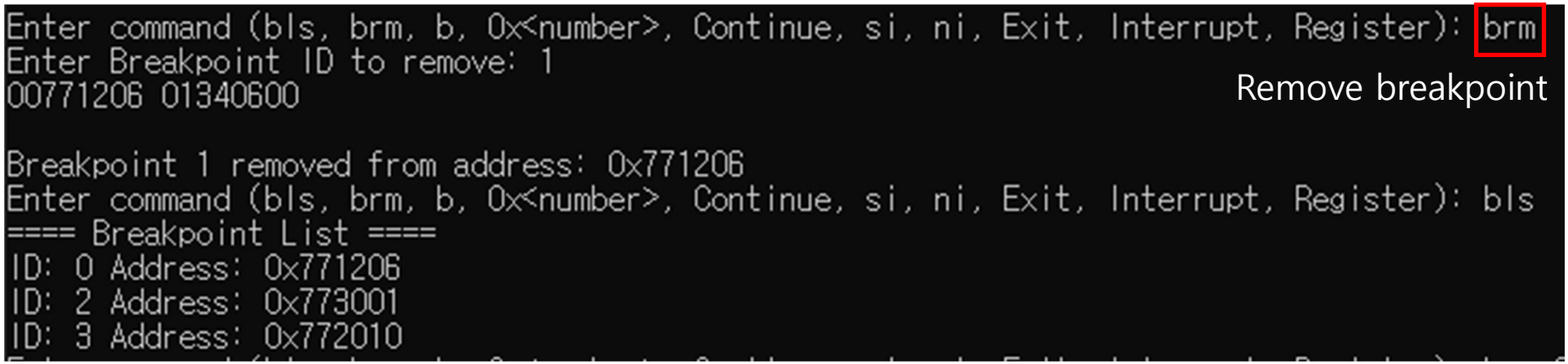

C:₩Users₩admin₩source₩repos₩peb₩Debug₩peb.exe
프로세스 베이스 770000
바이너리 베이스 400000
최종 베이스 370000
==== PE File Information ====
Image Size: 32768 bytes
Address of Entry Point: 0x1df7
Number of Sections: 5
PE Header Address: 00F60000
PE Header Size: 1024 bytes
==== Section Information ====
Section 1 Address: 0xf601e8
Section 2 Address: 0xf60210
Section 3 Address: 0xf60238
Section 4 Address: 0xf60260
==== Relocation Information ====
Image Base: 0x400000
Relative Offset: 0x0
==========
EAX: 771df7
EBX: 85e000
ECX: 0
EDX: 0
EIP: 777355b0
ESP: affa54
EBP: 0
ESI: 0
EDI: 0
Error: BP (0x0) is less than or equal to SP (0xaffa54)
CREATE_PROCESS_DEBUG_EVENT
Base of Image: 00770000
EXCEPTION_BREAKPOINT
Disassembly at address 0x777355B0:
0x777355b0: cmp dword ptr [0x777e69b8], 0
0x777355b7: je 0x777355c7
0x777355b9: mov ecx, dword ptr [0x777e69b8]
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): c
Executing 'c' command.
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): b 0x1206
Breakpoint 0 added at address: 0x771206
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): c
Executing 'c' command.
Breakpoint hit at address: 0x771206
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register):
C:₩Users₩admin₩source₩repos₩test₩Release₩test.exe
프로그램이 실행 중입니다. 종료하려면 'exit'을 입력하세요.
입력: hello
Wait...


C:₩Users₩admin₩source₩repos₩peb₩Debug₩peb.exe
프로세스 베이스 770000
바이너리 베이스 400000
최종 베이스 370000
==== PE File Information ====
Image Size: 32768 bytes
Address of Entry Point: 0x1df7
Number of Sections: 5
PE Header Address: 00F60000
PE Header Size: 1024 bytes
==== Section Information ====
Section 1 Address: 0xf601e8
Section 2 Address: 0xf60210
Section 3 Address: 0xf60238
Section 4 Address: 0xf60260
==== Relocation Information ====
Image Base: 0x400000
Relative Offset: 0x0
==========
EAX: 771df7
EBX: 85e000
ECX: 0
EDX: 0
EIP: 777355b0
ESP: affa54
EBP: 0
ESI: 0
EDI: 0
Error: BP (0x0) is less than or equal to SP (0xaffa54)
CREATE_PROCESS_DEBUG_EVENT
Base of Image: 00770000
EXCEPTION_BREAKPOINT
Disassembly at address 0x777355B0:
0x777355b0: cmp dword ptr [0x777e69b8], 0
0x777355b7: je 0x777355c7
0x777355b9: mov ecx, dword ptr [0x777e69b8]
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): c
Executing 'c' command.
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): b 0x1206
Breakpoint 0 added at address: 0x771206
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): c
Executing 'c' command.
Breakpoint hit at address: 0x771206
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): r
Executing 'r' command.
EAX: 721947b8
EBX: 85e000
ECX: ec8fa830
EDX: aff964
EIP: 771206
ESP: aff95c
EBP: aff98c
ESI: 721947b8
EDI: d2ee98
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register):
print_register
C:₩Users₩admin₩source₩repos₩test₩Release₩test.exe
프로그램이 실행 중입니다. 종료하려면 'exit'을 입력하세요.
입력: hello


Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): r
Executing 'r' command.
EAX: 721947b8
EBX: 85e000
ECX: ec8fa830
EDX: aff964
EIP: 771206
ESP: aff95c
EBP: aff98c
ESI: 721947b8
EDI: d2ee98
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): f
Executing 'f' command.
Dumping memory between BP: 0xaff98c and SP: 0xaff95c
0xaff95c: 0xec8fa8ec
0xaff960: 0xd26df8
0xaff964: 0x6c6c6568
0xaff968: 0x6f
0xaff96c: 0x0
0xaff970: 0x0
0xaff974: 0x5
0xaff978: 0xf
0xaff97c: 0xec8fa8ec
0xaff980: 0xaff9c4
0xaff984: 0x77289d
0xaff988: 0x0
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register):
print_stack frame


C:₩Users₩admin₩source₩repos₩peb₩Debug₩peb.exe
==== Section Information ====
Section 1 Address: 0xf601e8
Section 2 Address: 0xf60210
Section 3 Address: 0xf60238
Section 4 Address: 0xf60260
==== Relocation Information ====
Image Base: 0x400000
Relative Offset: 0x0
==========
EAX: 771df7
EBX: 85e000
ECX: 0
EDX: 0
EIP: 777355b0
ESP: affa54
EBP: 0
ESI: 0
EDI: 0
Error: BP (0x0) is less than or equal to SP (0xaffa54)
CREATE_PROCESS_DEBUG_EVENT
Base of Image: 00770000
EXCEPTION_BREAKPOINT
Disassembly at address 0x777355B0:
0x777355b0: cmp dword ptr [0x777e69b8], 0
0x777355b7: je 0x777355c7
0x777355b9: mov ecx, dword ptr [0x777e69b8]
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): c
Executing 'c' command.
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): b 0x1206
Breakpoint 0 added at address: 0x771206
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): c
Executing 'c' command.
Breakpoint hit at address: 0x771206
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): r
Executing 'r' command.
EAX: 721947b8
EBX: 85e000
ECX: ec8fa830
EDX: aff964
EIP: 771206
ESP: aff95c
EBP: aff98c
ESI: 721947b8
EDI: d2ee98
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): f
Executing 'f' command.
Dumping memory between BP: 0xaff98c and SP: 0xaff95c
0xaff95c: 0xec8fa8ec
0xaff960: 0xd26df8
0xaff964: 0x6c6c6568
0xaff968: 0x6f
0xaff96c: 0x0
0xaff970: 0x0
0xaff974: 0x5
0xaff978: 0xf
0xaff97c: 0xec8fa8ec
0xaff980: 0xaff9c4
0xaff984: 0x77289d
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): c
Executing 'c' command.
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register):
release
C:₩Users₩admin₩source₩repos₩test₩Release₩test.exe
프로그램이 실행 중입니다. 종료하려면 'exit'을 입력하세요.
입력: hello
입력한 내용: hello
입력:
release


Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): b 0x1206
Breakpoint 1 added at address: 0x771206
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): b 0x3001
Breakpoint 2 added at address: 0x773001
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): b 0x2010
Breakpoint 3 added at address: 0x772010
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): bls
==== Breakpoint List ====
ID: 0 Address: 0x771206
ID: 1 Address: 0x771206
ID: 2 Address: 0x773001
ID: 3 Address: 0x772010
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register):
Breakpoint list


Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): brm
Enter Breakpoint ID to remove: 1
00771206 01340600
Remove breakpoint
Breakpoint 1 removed from address: 0x771206
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): bls
==== Breakpoint List ====
ID: 0 Address: 0x771206
ID: 2 Address: 0x773001
ID: 3 Address: 0x772010

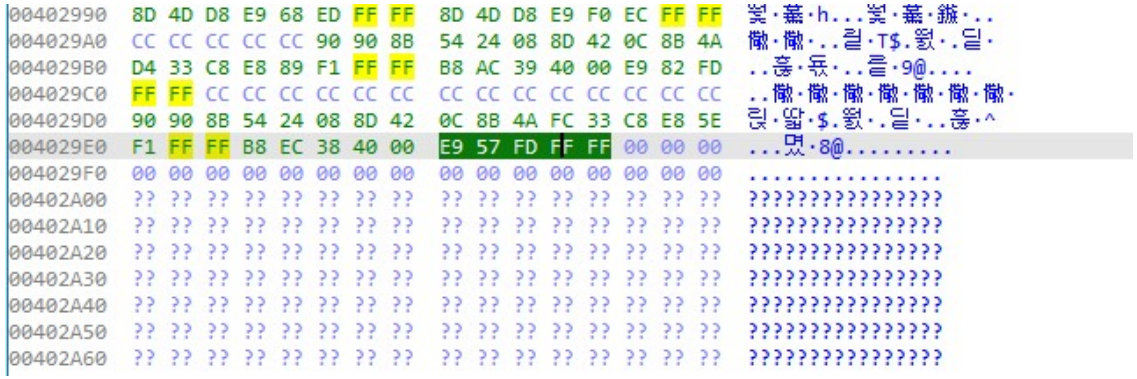
🔥 PATCH NOTE. 1

```
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): brm
Enter Breakpoint ID to remove: 0
Original instruction not found for address 0x771206.
```

같은 주소에 bp 2번 걸고 2번 삭제하면, originalInstruction 지워져서 복구 불가능
(GUI 제작하면서 패치 예정)

TODO.
Breakpoint duplication check!

PATCH NOTE. 2



```
00402990  8D 4D D8 E9 68 ED FF FF  8D 4D D8 E9 F0 EC FF FF  ?M??h???M????????
004029A0  CC CC CC CC CC 90 90 8B  54 24 08 8D 42 0C 8B 4A  ??????..T$..B..J
004029B0  D4 33 C8 E8 89 F1 FF FF  B8 AC 39 40 00 E9 82 FD  ?3??????9@.????
004029C0  FF FF CC CC CC CC CC CC  CC CC CC CC CC CC CC CC  ????????????????
004029D0  90 90 8B 54 24 08 8D 42  0C 8B 4A FC 33 C8 E8 5E  ...T$..B..J?3??^
004029E0  F1 FF FF B8 EC 38 40 00  E9 57 FD FF FF 00 00 00  ?????8@.?W??????
004029F0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ????????????????
00402A00  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
00402A10  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
00402A20  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
00402A30  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
00402A40  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
00402A50  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
00402A60  ?? ?? ?? ?? ?? ?? ?? ??  ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
```

마지막 0x00 or ??? 파트를 disassemble 로직에 들어가도록 유도하면..?

```
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): d 0x29e8
Disassembly at address 0x007729E8:
0x7729e8:       jmp             0x772744
0x7729ed:       add             byte ptr [eax], al
0x7729ef:       add             byte ptr [eax], al
0x7729f1:       add             byte ptr [eax], al
0x7729f3:       add             byte ptr [eax], al
0x7729f5:       add             byte ptr [eax], al
Enter command (bls, brm, b, 0x<number>, Continue, si, ni, Exit, Interrupt, Register): _
```

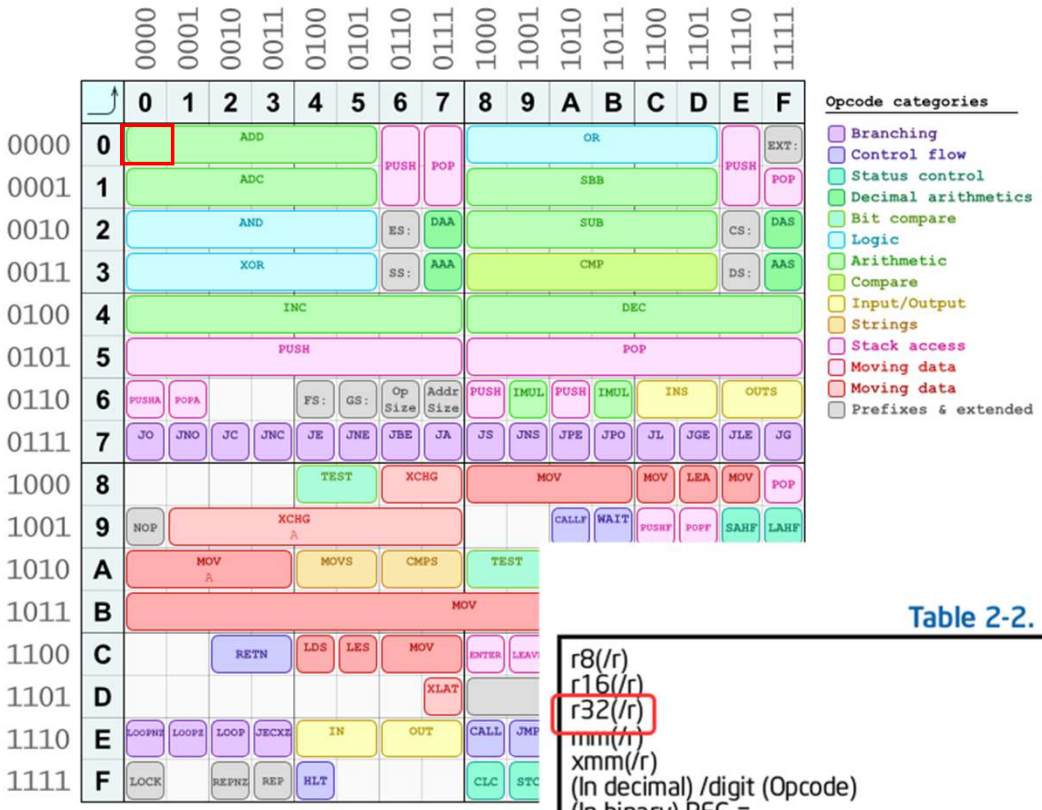


Table 2-2. 32-Bit Addressing Forms with the ModR/M Byte

r8(/r) r16(/r) r32(/r) mm(/r) xmm(/r) (In decimal) /digit (Opcode) (In binary) REG =			AL AX EAX MM0 XMM0 0 000	CL CX ECX MM1 XMM1 1 001	DL DX EDX MM2 XMM2 2 010	BL BX EBX MM3 XMM3 3 011	AH SP ESP MM4 XMM4 4 100	CH BP EBP MM5 XMM5 5 101	DH SI ESI MM6 XMM6 6 110	BH DI EDI MM7 XMM7 7 111
Effective Address	**Mod**	**R/M**	**Value of ModR/M Byte (in Hexadecimal)**							
[EAX]	00	000	00	08	10	18	20	28	30	38
[ECX]		001	01	09	11	19	21	29	31	39
[EDX]		010	02	0A	12	1A	22	2A	32	3A
[EBX]		011	03	0B	13	1B	23	2B	33	3B
[--][--][1]		100	04	0C	14	1C	24	2C	34	3C
disp32[2]		101	05	0D	15	1D	25	2D	35	3D
[ESI]		110	06	0E	16	1E	26	2E	36	3E
[EDI]		111	07	0F	17	1F	27	2F	37	3F

Ref : https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html

🔥 PATCH NOTE. 2

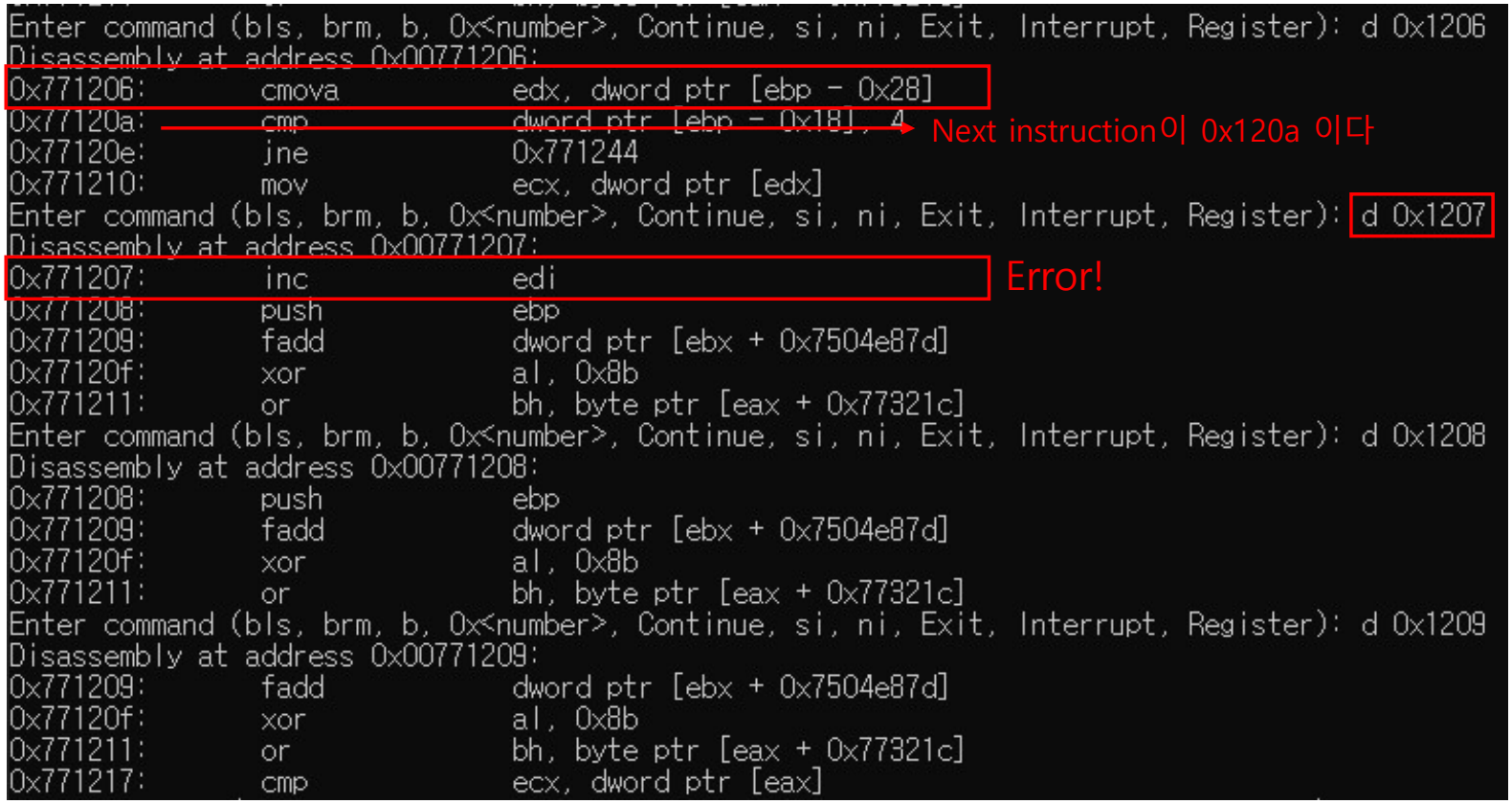
00 00 00 00 가 명령어로 인식되어

add byte ptr [eax], al

로 인식된다.

PATCH NOTE. 3

1byte 명령어가 아닌 상태에서, 명령어 중간을 disassemble한 경우 오류 발생



🔥 PATCH NOTE. 3

1byte 명령어가 아닌 상태에서, 명령어 중간을 disassemble한 경우 오류 발생

TODO.

명령어 정렬을 시도하거나, 아이디어 떠올려보기...

요구사항 :

- ~~PE parser~~
- ~~Disassemble~~
- ~~Memory View~~
- ~~BreakPoint~~

Plan :

- ~~Windows Structure 를 공부하여 pe.h 제작~~
- ~~Disassemble lib으로 capstone.h을 사용~~
- ~~Debugee Process create, virtual memory R/W 용 exec.h~~
- ~~Memory write로 int3를 injection~~