

KU - Class74 - Assignment #2

이유섭 / 2018320022

본 프로젝트의 전체 코드는 아래 링크를 통해 확인할 수 있으며, 작성된 코드의 양이 많아 본 보고서는 주요 기능에 대해서만 서술하였다.

<https://github.com/jpark-classroom/class74-LeeYuseop.git>

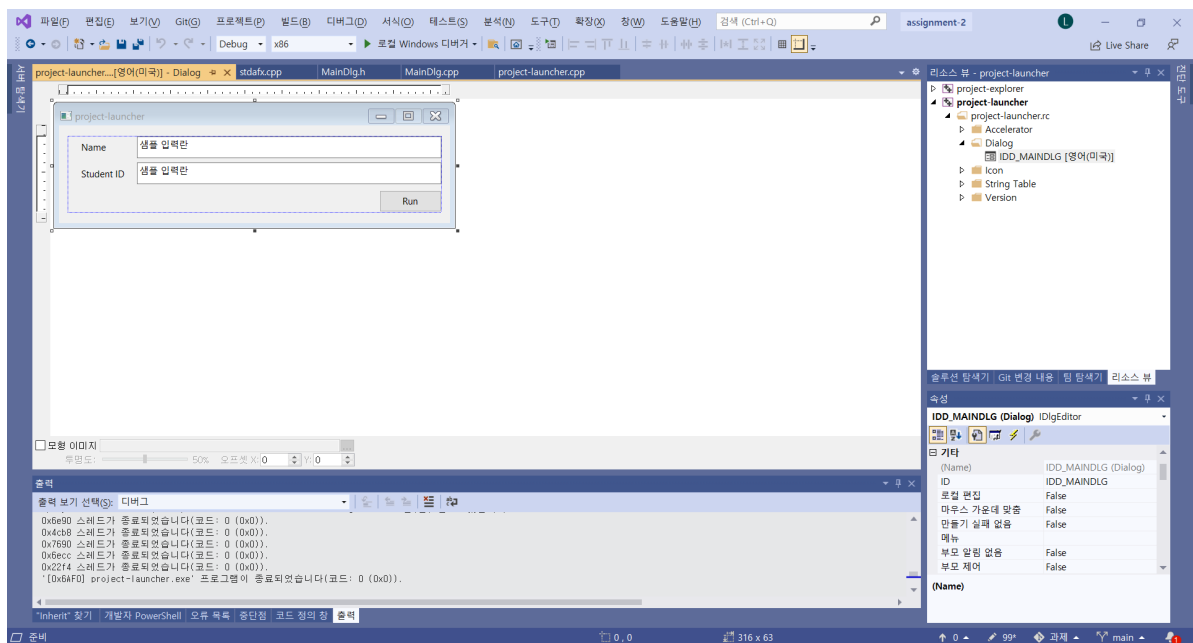
Launcher

Launcher는 다음 4가지 작업을 한다.

- 공유 메모리를 만든다.
- 사용자로부터 이름과 학번을 입력 받는다.
- 사용자가 Run 버튼을 누르면 사용자가 입력한 값들을 공유 메모리에 쓰고, Explorer를 실행시킨다. (공유 메모리도 공유해야 한다.)
- Explorer 종료 시, Explorer가 공유메모리로 넘긴 종료 시간을 화면에 출력한다.

Launcher project에서는 MainDlg.cpp 파일에 주요 기능들이 모두 구현되어 있다.

Dialog 파일



이는 project-launcher.rc 파일 중 Dialog 파일로 project-launcher 프로젝트 실행 시 제일 먼저 뜨게 될 화면이다.

사용자가 샘플 입력란에 해당하는 부분에 이름과 학번을 쓰고 우측 하단에 있는 Run 버튼을 누르면 미리 만들어 뒀던 project-explorer 실행파일이 실행된다. 이후 project-explorer 실행파일이 종료되면, 지금은 아무것도 쓰여있지 않아 보이진 않지만, Student ID 텍스트 아래에 project-explorer 실행파일이 종료된 시간이 출력된다.

OnInitDialog 함수

```
LRESULT CMainDlg::OnInitDialog(UINT /*uMsg*/, WPARAM /*wParam*/, LPARAM
/*lParam*/, BOOL& /*bHandled*/) {
    // dialog 설정
    CenterWindow();

    // 아이콘 설정
    HICON hIcon = AtlLoadIconImage(IDR_MAINFRAME, LR_DEFAULTCOLOR,
::GetSystemMetrics(SM_CXICON), ::GetSystemMetrics(SM_CYICON));
    SetIcon(hIcon, TRUE);
    HICON hIconSmall = AtlLoadIconImage(IDR_MAINFRAME, LR_DEFAULTCOLOR,
::GetSystemMetrics(SM_CXSMICON), ::GetSystemMetrics(SM_CYSMICON));
    SetIcon(hIconSmall, FALSE);

    // 공유메모리 생성
    m_hSharedMem.reset(::CreateFileMapping(INVALID_HANDLE_VALUE, nullptr,
PAGE_READWRITE, 0, 1 << 16, nullptr));
    ATLASSERT(m_hSharedMem);

    return TRUE;
}
```

OnInitDialog 함수는 CMainDlg class에서 가장 먼저 실행되는 함수로, 미리 만들어 두었던 dialog를 띄우고, 아이콘을 설정하며, 가장 중요한 공유 메모리를 생성하게 된다.

공유 메모리는 CreateFileMapping 함수를 사용하여 만들 수 있다. 해당 함수에서 반환된 handle 값은 헤더파일에서 미리 정의해 두었던 m_hSharedMem 변수에 저장하여 이후 필요할 때마다 공유 메모리의 handle 값을 얻어 사용할 수 있다.

```
HANDLE CreateFileMapping(
    [in] HANDLE hFile,
    [in, optional] LPSECURITY_ATTRIBUTES lpFileMappingAttributes,
    [in] DWORD flProtect,
    [in] DWORD dwMaximumSizeHigh,
    [in] DWORD dwMaximumSizeLow,
    [in, optional] LPCSTR lpName
);
```

CreateFileMapping 함수는 위와 같이 생겼으며, 우리는 처음 공유메모리를 만들어야 하기 때문에 hFile 인자에 INVALID_HANDLE_VALUE 값을 넣었으며, 읽기/쓰기 권한을 주기 위해 flProtect 인자에 PAGE_READWRITE 값을 넣어주었다. hFile 인자에 INVALID_HANDLE_VALUE 값을 넣게 되면 dwMaximumSizeHigh와 dwMaximumSizeLow 인자를 설정해주어야 하는데, 본 프로젝트에서는 각각 0과 1<<16() 값을 넣어주었다.

OnRun 함수

```

LRESULT CMainDlg::OnRun(WORD, WORD, HWND, BOOL &) {
    // 사용자가 입력한 값을 공유 메모리에 입력
    WriteInfo();

    // thread를 만들어 explorer 실행 및 대기
    HANDLE hThread = ::CreateThread(nullptr, 0, ThreadProc, (void*)this, 0,
    nullptr);
    CloseHandle(hThread);

    return 0;
}

```

OnRun 함수는 Run 버튼을 클릭시 실행되는 함수로, 공유 메모리에 사용자가 입력한 정보를 쓴 후, thread를 생성하여, 생성된 thread에서 explorer를 실행하고 대기하는 작업이 이루어진다.

```

HANDLE CreateThread(
    [in, optional] LPSECURITY_ATTRIBUTES   lpThreadAttributes,
    [in]           SIZE_T                   dwStackSize,
    [in]           LPTHREAD_START_ROUTINE  lpStartAddress,
    [in, optional] __drv_aliasesMem LPVOID lpParameter,
    [in]           DWORD                    dwCreationFlags,
    [out, optional] LPDWORD                 lpThreadId
);

```

thread를 생성하는 CreateThread 함수를 살펴보면, lpStartAddress 인자로 thread를 만들어 실행하고자 하는 함수의 포인터를 넘겨주게 된다. 그러나 class 내부 함수는 CreateThread 함수의 인자로 넘겨줄 수가 없다. 이를 해결하기 위해 본 프로젝트에서는 ThreadProc 이라는 외부 함수를 만들어 인자로 넘겨주었으며, ThreadProc 함수에서 class 내부 함수를 호출하도록 작성하였다. 또한 lpParameter 인자로 this를 넘겨줌으로써, 새로운 class를 만들지 않고 기존의 class의 내부 함수를 호출 할 수 있도록 하였다.

```

BOOL CloseHandle(
    [in] HANDLE hObject
);

```

CloseHandle 함수는 생성했던 프로세스 혹은 스레드의 핸들을 닫는 함수로 OnRun 함수에서는 thread를 닫는 용도로 쓰였으며, 이후 프로세스를 닫는 용도로도 쓰인다.

OnRun에서 쓰인 WriteInfo 함수와 ThreadProc 함수, 그리고 ThreadProc 함수에서 호출하는 CreateProc 함수에 대해서도 살펴보자.

WriteInfo 함수

```

void CMainDlg::WriteInfo() {
    // 공유 메모리를 쓰기 기능으로 매핑
    void* buffer = ::MapViewOfFile(m_hSharedMem.get(), FILE_MAP_WRITE, 0, 0, 0);
    if (!buffer) return;    // 공유 메모리 매핑 실패 시, return

    // Name 가져오기
    CString name;
    GetDlgItemText(IDC_NAME, name);
    ::wcscpy_s((PWSTR)buffer, name.GetLength() + 1, name);

    // '\0'로 Name과 Student ID 구분하기
}

```

```

        ::wcscpy_s((PWSTR)buffer + name.GetLength(), 1, L"\0");

        // Student ID 가져오기
        CString student_id;
        GetDlgItemText(IDC_STUDENT_ID, student_id);
        ::wcscpy_s((PWSTR)buffer + name.GetLength() + 1, student_id.GetLength() + 1,
        student_id);

        // 공유 메모리 매핑 해제
        ::UnmapViewOfFile(buffer);
    }

```

WriteInfo 함수는 사용자가 입력한 값을 공유메모리에 쓰는 작업을 하므로, MapViewOfFile 함수를 이용해 공유 메모리를 매핑하고, 마지막에 UnmapViewOfFile 함수를 이용해 공유 메모리 매핑을 해제한다. 사용자가 Edit Control 에 입력한 값은 GetDlgItemText 함수로 가져올 수 있으며, 이 값은 wcscpy_s 함수를 통해 공유 메모리에 복사할 수 있다. 본 프로젝트에서는 사용자로부터 2개의 입력을 받게되는 반면, 공유메모리는 하나를 사용하므로 이를 적절히 저장하여야 한다. 이에 이름과 학번 사이에 null 값인 '\0'를 써주어 이 둘을 구분하였다. 두 문자열을 다른 값을 이용해 두 문자열을 구분했다면, 이름을 복사할 때 이름의 길이를 같이 알아야 하거나 이름과 학번을 따로 구분하는 코드를 작성해야 하지만 '\0'로 두 문자열을 구분하게 되면 공유메모리의 값을 그대로 읽으면 이름만 읽을 수 있고, 처음으로 나타난 '\0' 이후의 값을 읽으면 학번만 읽을 수 있다는 장점이 있다.

```

LPVOID MapViewOfFile(
    [in] HANDLE hFileMappingObject,
    [in] DWORD dwDesiredAccess,
    [in] DWORD dwFileOffsetHigh,
    [in] DWORD dwFileOffsetLow,
    [in] SIZE_T dwNumberOfBytesToMap
);

```

MapViewOfFile 함수는 위와 같이 생겼으며, hFileMappingObject 인자로 매핑하고자 하는 공유 메모리의 handle 값을 넘겨주며, dwDesiredAccess 인자로 부여하고자 하는 권한을 넘겨주면 된다. dwDesiredAccess 인자에 관련한 값으로는 FILE_MAP_ALL_ACCESS와 FILE_MAP_READ, FILE_MAP_WRITE가 있으며, 이름에서 알 수 있듯이, 읽기/쓰기 권한, 읽기 권한, 쓰기 권한으로 hFileMappingObject 인자에 넘긴 handle 을 매핑하겠다는 의미이다.

```

BOOL UnmapViewOfFile(
    [in] LPCVOID lpBaseAddress
);

```

UnmapViewOfFile 함수는 MapViewOfFile 함수로부터 매핑한 공유 메모리를 해제할 때 쓰는 함수로 MapViewOfFile 함수에서 반환한 값을 인자로 넘겨주면 된다.

```

UINT GetDlgItemTextA(
    [in] int nIDDlgItem,
    [out] LPSTR lpString,
);

```

GetDlgItemText 함수는 dialog control 에 있는 값을 가져오는 함수로, nIDDlgItem에는 값을 가져오게 하는 dialog control ID를, lpString에는 값을 저장할 WCHAR 배열 주소를 넘겨주면 된다.

ThreadProc 함수

```
// thread에서 class 내부 함수를 실행할 수 없어 외부 함수 ThreadProc를 통하여 class 내부  
함수 실행  
DWORD WINAPI ThreadProc(LPVOID IParam) {  
    return ((CMainDlg*)IParam)->CreateProc();  
}
```

ThreadProc 함수는 class 내부 함수를 thread로 호출할 수 없어 형식적으로 들르는 외부 함수로, CMainDlg class 객체를 인자로 받아 CMainDlg class의 내부 함수인 CreateProc을 실행시키도록 하였다.

CreateProc 함수

```
DWORD CMainDlg::CreateProc() {  
    // 상속 가능하도록 공유 메모리 설정  
    ::SetHandleInformation(m_hSharedMem.get(), HANDLE_FLAG_INHERIT,  
        HANDLE_FLAG_INHERIT);  
  
    STARTUPINFO si = {sizeof(si)};  
    PROCESS_INFORMATION pi;  
  
    // command line 생성  
    WCHAR path[MAX_PATH] = _T("project-explorer.exe");  
    WCHAR handle[16];  
    ::_itow_s((int)(ULONG_PTR)m_hSharedMem.get(), handle, 10);  
    ::wcscat_s(path, L" ");  
    ::wcscat_s(path, handle);  
  
    // 새로운 process 실행 (explorer 실행)  
    if (::CreateProcess(nullptr, path, nullptr, nullptr, TRUE,  
        0, nullptr, nullptr, &si, &pi)) {  
        // explorer 실행 후 대기  
        ::WaitForSingleObject(pi.hProcess, INFINITE);  
        // explorer 종료 시, timestamp 출력  
        writeTimeStamp();  
        ::CloseHandle(pi.hProcess);  
        ::CloseHandle(pi.hThread);  
    }  
    else; // error  
  
    return 0;  
}
```

CreateProc 함수는 미리 만들어둔 explorer를 실행시키는 함수로, 크게 공유 메모리 핸들을 넘기고 explorer 실행 및 대기 후 종료 timestamp를 출력하는 과정으로 나뉜다. explorer는 CreateProcess 함수를 통해 실행하게 되는데, 이 때, 공유 메모리 핸들도 같이 command line으로 넘김으로써, explorer가 실행 후 공유 메모리를 사용할 수 있게 해준다. 이를 위해 우선 인자 값을 HANDLE_FLAG_INHERIT로 SetHandleInformation 함수를 실행함으로써 공유 메모리의 핸들 값을 상속 가능하게 바꿔준다. 이후 "project-explorer.exe [공유 메모리 핸들]" 형식의 command line으로 넘길 문자열을 생성해준다. 만든 command line을 CreateProcess 함수를 통해 실행시켜주면 explorer가 실행되게 된다. 이후, 성공적으로 explorer가 실행되면, 해당 프로세스가 끝날때 까지 WaitForSingleObject 함수를 통해 기다려주고,

explorer 종료 시, WriteTimeStamp 함수를 통해 explorer가 공유 메모리로 넘긴 종료 timestamp를 출력하고 explorer의 프로세스 핸들과 스레드 핸들을 닫아준다.

```
BOOL SetHandleInformation(  
    [in] HANDLE hObject,  
    [in] DWORD dwMask,  
    [in] DWORD dwFlags  
);
```

SetHandleInformation 함수로 hObject 인자로써 정보를 변경하고자 하는 핸들 값을, dwMask와 dwFlags에는 변경하고자 하는 값들을 넣어주면 된다. dwMask와 dwFlags의 인자값으로는 HANDLE_FLAG_INHERIT과 HANDLE_FLAG_PROTECT_FROM_CLOSE가 있으며, 본 프로젝트에서는 상속을 해야하기 때문에 HANDLE_FLAG_INHERIT 인자를 사용하였다.

```
BOOL CreateProcessA(  
    [in, optional] LPCSTR lpApplicationName,  
    [in, out, optional] LPSTR lpCommandLine,  
    [in, optional] LPSECURITY_ATTRIBUTES lpProcessAttributes,  
    [in, optional] LPSECURITY_ATTRIBUTES lpThreadAttributes,  
    [in] BOOL bInheritHandles,  
    [in] DWORD dwCreationFlags,  
    [in, optional] LPVOID lpEnvironment,  
    [in, optional] LPCSTR lpCurrentDirectory,  
    [in] LPSTARTUPINFOA lpStartupInfo,  
    [out] LPPROCESS_INFORMATION lpProcessInformation  
);
```

CreateProcess 함수는 다양한 인자를 통하여 다양하게 사용될 수 있으나, 모든 것을 다루기에는 보고서의 성격과 맞지 않기 때문에 본프로젝트에서 사용한 인자값만 설명하고자 한다. 프로세스를 실행하는 방법은 lpApplicationName과 lpCommandLine 인자를 사용하는 2가지 방법이 있다. 그러나 본 프로젝트에서는 explorer를 실행 할 때, 공유 메모리 핸들 값도 같이 넘겨줘야 하므로 command line을 생성하여 command line으로 프로세스를 실행시키는 방법을 사용하였다. 또한 상속을 해야하기 때문에 bInheritHandles 값을 TRUE로 넘겨주었다.

```
DWORD WaitForSingleObject(  
    [in] HANDLE hHandle,  
    [in] DWORD dwMilliseconds  
);
```

WaitForSingleObject 함수는 특정 객체의 신호를 기다리는 함수로, 기다리고자 하는 객체의 handle 값을 hHandle 인자로 넘겨주며, dwMilliseconds 인자로 기다릴 시간을 넘겨준다. 시간은 milliseconds 단위이며, 신호를 받을 때까지 기다리게 하고 싶을 때에는 인자로 INFINITE를 넘겨주면 된다. INFINITE가 아닌 다른 값을 넘겨주게 되면, 신호가 오지 않더라도 해당 시간 이후 WAIT_TIMEOUT 값을 반환하며 함수가 끝나게 된다.

WriteTimeStamp 함수

```

void CMainDlg::WriteTimeStamp() {
    // 공유 메모리를 읽기 기능으로 매핑
    void* buffer = ::MapViewOfFile(m_hSharedMem.get(), FILE_MAP_READ, 0, 0, 0);
    if (!buffer) return;    // 공유 메모리 매핑 실패 시, return

    // "terminated timestamp: [공유 메모리 값]" 으로 text 배열 설정
    WCHAR text[100] = L"terminated timestamp: ";
    ::wcscat_s(text, (PCWSTR)buffer);

    SetDlgItemText(IDC_TIMESTAMP, (PCWSTR)text);

    ::UnmapViewOfFile(buffer);
}

```

WriteTimeStamp 함수는 explorer가 실행되고 종료될 때 공유 메모리에 쓴 종료 timestamp 값을 출력하는 함수로, MapViewOfFile 함수를 통해 공유 메모리를 읽기 권한으로 매핑한다. 이후 공유 메모리 값을 이용하여 "terminated timestamp: [공유 메모리 값]" 문자열을 만들고, 이를 IDC_TIMESTAMP dialog control에 적어준다.

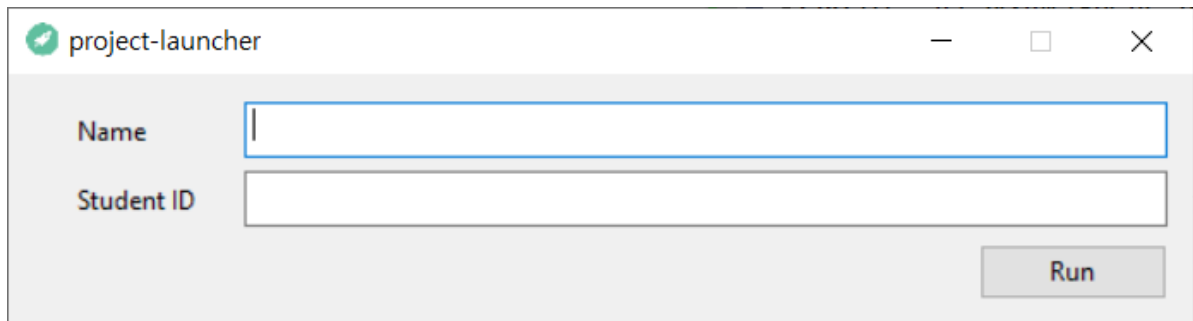
```

BOOL SetDlgItemTextA(
    [in] int    nIDDlgItem,
    [in] LPCSTR lpString
);

```

setDlgItemText 함수는 getDlgItemText 함수와 반대로 nIDDlgItem 인자로 넘겨준 dialog control 에 lpString 인자로 넘겨준 문자열을 적는 함수이다.

Launcher 실행 화면



project-launcher 프로젝트를 실행하게 되면 앞서 만들었던 Dialog 파일이 뜨는 것을 볼 수 있다.

Name과 Student ID 에 이름과 학번을 적어서 Run 버튼을 누르면 아래와 같이 미리 만들어 두었던 project-explorer 파일이 실행된다.

project-explorer

File View Help

Examiner Name: LeeYuseop
 Examiner ID: 2018320022
 Operating System: Windows 10 Home (10.0.19042)
 Computer Name: LAPTOP-A77P65TI

Name	PID	PPID	Session	User Name	Threads	Handles	Working Set	CPU Time
[Idle]	0	0	0		12	0	8 KB	20:17:28:46
System	4	0	0		276	7163	36 KB	0:01:14:59
Registry	148	4	0		4	0	62292 KB	0:00:00:03
smss.exe	536	4	0		2	53	640 KB	0:00:00:00
csrss.exe	748	652	0		14	836	2640 KB	0:00:00:04
wininit.exe	984	652	0		1	169	3068 KB	0:00:00:00
services.exe	620	984	0		7	801	10576 KB	0:00:19:41
lsass.exe	788	984	0		9	1851	17328 KB	0:00:01:01
svchost.exe	944	620	0		10	1676	28264 KB	0:00:04:07
fontdrvhost.exe	1048	984	0		5	32	2236 KB	0:00:00:00
WUDFHost.exe	1112	620	0		12	348	11520 KB	0:00:00:09
svchost.exe	1168	620	0		11	1650	18080 KB	0:00:02:34
svchost.exe	1224	620	0		4	390	5960 KB	0:00:00:35
svchost.exe	1444	620	0		2	301	5712 KB	0:00:00:01
svchost.exe	1452	620	0		2	223	6980 KB	0:00:00:00
svchost.exe	1480	620	0		7	313	9344 KB	0:00:00:04
svchost.exe	1648	620	0		2	248	7344 KB	0:00:00:09
svchost.exe	1656	620	0		4	253	6188 KB	0:00:00:02

Ready

실행된 project-explorer를 보면 앞서 launcher에서 적었던 이름과 학번이 상단에 적혀 있는 것을 확인할 수 있다.

project-launcher

Name: LeeYuseop

Student ID: 2018320022

terminated timestamp: 20:50:14

Run

project-explorer 종료 시, 종료 시간이 project-launcher에 출력되는 것을 확인할 수 있다.

S&P Explorer

Explorer는 다음 4가지 작업을 한다.

- 공유 메모리에 적힌 이름과 학번을 출력한다.
- 시스템 기본 정보를 출력한다.
- 실행 중인 프로세스를 목록화 한다.
- 종료 시, 종료 시간을 공유 메모리에 입력한다.

explorer 프로젝트의 코드는 launcher에 비해 조금 복잡하다. launcher가 explorer를 실행시키게 되면, 가장 먼저 main 함수가 있는 project-explorer 가 실행되고, project-explorer 에서는 MainForm을, MainForm이 메뉴와 툴바 등등을 호출하게 된다.

project-explorer.cpp

project-explorer 프로젝트 실행 시 가장 먼저오게 되는 cpp 파일로, tWinMain 함수가 있다. tWinMain 함수는 기본적인 설정 후, Run 함수를 호출하게 되는데 Run 함수는 아래와 같다.

Run 함수

```
int Run(LPTSTR /*lpstrCmdLine*/ = nullptr, int nCmdShow = SW_SHOWDEFAULT) {
    CMessageLoop theLoop;
    _Module.AddMessageLoop(&theLoop);

    CMainFrame wndMain;

    if (wndMain.CreateEx() == nullptr) {
        ATLTRACE(_T("Main window creation failed!\n"));
        return 0;
    }

    wndMain.ShowWindow(nCmdShow);

    int nRet = theLoop.Run();

    _Module.RemoveMessageLoop();

    wil::unique_handle m_hSharedMem;

    int count;
    PWSTR* args = ::CommandLineToArgvW(::GetCommandLine(), &count);
    if (count == 1) {
        // 단독 실행 시 공유 메모리 생성
        m_hSharedMem.reset(::CreateFileMapping(INVALID_HANDLE_VALUE, nullptr,
        PAGE_READWRITE, 0, 1 << 16, nullptr));
    }
    else {
        // launcher를 통해 실행 시, launcher에서 넘겨준 핸들 값으로 공유 메모리 설정
        m_hSharedMem.reset((HANDLE)(ULONG_PTR)::_wtoi(args[1]));
    }
    ::LocalFree(args);

    ATLASSTERT(m_hSharedMem);

    void* buffer = ::MapViewOfFile(m_hSharedMem.get(), FILE_MAP_WRITE, 0, 0, 0);
    if (!buffer) return nRet; // 공유 메모리 매핑 실패 시, return

    // 현재 시간을 가져와서 공유 메모리에 입력
    CTime dt = CTime::GetCurrentTime();
    ::wcscpy_s((PWSTR)buffer, 100, dt.Format(L"%T"));

    // 공유 메모리 매핑 해제
    ::UnmapViewOfFile(buffer);

    return nRet;
}
```

Run 함수에서도 기본적인 설정 후, 프로세스의 핵심이 되는 함수들을 호출해나간다. 이후, 프로세스가 어떤 방식으로든 종료가 되면 Run 함수가 종료되며 프로세스가 종료가 되게 되는데, Run 함수가 종료되기 전에 공유 메모리에 시간을 저장하고 종료한다. launcher에서 command line을 통해 공유 메모리 핸들값을 넘겨줬기 때문에 CommandLineToArgvW 함수를 통해 실행 시 받은 command line을 분석한다. command line을 분석하면 args[1]에 공유 메모리 핸들 값이 들어 있으므로 wil::unique_handle 형의 m_hSharedMem 변수를 만들어 handle 값을 args[1]로 설정해준다. 이후 공유 메모리를 쓰기 권한으로 매핑시키고, GetCurrentTime 함수를 통해 현재 시간 (종료 시간)을 가져와 이를 시간 형식의 문자열로

변환하여 공유 메모리에 넣어준다. 이후 Run 함수를 종료한다. launcher를 통해 explorer 가 실행되면 command line에 공유 메모리 핸들이 있지만 단독으로 실행시키는 경우 생성된 공유 메모리가 없어 새로 생성하는 코드를 작성하여 단독으로도 실행될 수 있도록 하였다.

```
CTime GetCurrentTime();
```

GetCurrentTime 은 현재 시간을 CTime 형으로 불러오는 함수이다.

View.cpp

눈에 보이는 대부분의 것들이 View.cpp 에서 설정된다.

OnCreate 함수

```
LRESULT CView::OnCreate(UINT /*uMsg*/, WPARAM /*wParam*/, LPARAM /*lParam*/,  
    BOOL& /*bHandled*/) {  
    // 정보 바 생성  
    m_InformationBar.Create(m_hwnd);  
    m_InformationBar.ShowWindow(SW_SHOW);  
  
    // 프로세스 리스트 생성  
    m_List.Create(m_hwnd, rcDefault, nullptr,  
        WS_CHILD | WS_VISIBLE | WS_CLIPCHILDREN | WS_CLIPSIBLINGS | LVS_REPORT |  
    LVS_OWNERDATA | LVS_SINGLESEL | LVS_SHOWSELALWAYS,  
        0, 123);  
    m_List.SetExtendedListViewStyle(LVS_EX_FULLROWSELECT | LVS_EX_DOUBLEBUFFER,  
    0);  
  
    struct {  
        PCWSTR Header;  
        int width;  
        int Format = LVCFMT_LEFT;  
    } columns[] = {  
        { L"Name", 200 },  
        { L"PID", 100, LVCFMT_RIGHT },  
        { L"PPID", 100, LVCFMT_RIGHT },  
        { L"Session", 80, LVCFMT_RIGHT },  
        { L"User Name", 150 },  
        { L"Threads", 80, LVCFMT_RIGHT },  
        { L"Handles", 80, LVCFMT_RIGHT },  
        { L"Working Set", 100, LVCFMT_RIGHT },  
        { L"CPU Time", 120, LVCFMT_RIGHT },  
        { L"Full Path", 250, LVCFMT_RIGHT },  
    };  
  
    int i = 0;  
    for (auto& col : columns)  
        m_List.InsertColumn(i++, col.Header, col.Format, col.Width);  
  
    // 프로세스 정보 업데이트  
    Refresh();  
    // 기본정보 업데이트  
    m_InformationBar.Init();
```

```

        SetTimer(1, 1000, nullptr);

        return 0;
    }

```

OnCreate 함수에서는 기본적인 정보(이름, 학번, 시스템 정보)를 출력할 공간과 프로세스 리스트를 출력할 공간을 만드는 함수를 호출하고, 각 정보들을 업데이트 하는 함수를 호출한다.

Refresh 함수

```

void CView::Refresh() {
    // 프로세스 갱신
    m_ProcMgr.Refresh();
    m_List.SetItemCountEx(static_cast<int>(m_ProcMgr.GetProcesses().size()),
        LVSI CF_NOINVA LIDATEALL | LVSI CF_NOSCROLL);
    m_List.RedrawItems(m_List.GetTopIndex(), m_List.GetCountPerPage() +
        m_List.GetTopIndex());
}

```

Refresh 함수는 실행 중인 프로세스들에 대한 정보를 주기적으로 갱신하는 함수이다. ProcessManager.cpp 에 정의된 Refresh 함수를 호출하여 실행 중인 프로세스 정보를 갱신하고, 리스트를 갱신한다. 이는 OnCreate 함수에 의해서도 실행되지만, OnTimer라는 함수에 의해 주기적으로 실행되어 프로세스 정보들을 주기적으로 갱신한다.

OnGetDispInfo 함수

```

LRESULT CView::OnGetDispInfo(int, LPNMHDR pnmh, BOOL&) {
    auto lv = (NMLV DISPINFO*)pnmh;
    auto& item = lv->item;

    if (lv->item.mask & LVIF_TEXT) {
        const auto& data = m_ProcMgr.GetProcesses()[item.iItem];

        switch (item.iSubItem) {
            case 0: // name
                item.pszText = (PWSTR)(PCWSTR)data->Name;
                break;

            case 1: // pid
                StringCchPrintf(item.pszText, item.cchTextMax, L"%d", data->Id);
                break;

            ...
        }
    }
}

```

OnGetDispInfo 함수는 m_List 에 저장된 정보들을 사용자가 보기 좋게 문자열 형식으로 변환하여 출력을 용이하게 해주는 함수로, 프로세스 정보에서 추가하고 싶은 정보가 있을 시, OnCreate 함수에서 columns 추가와 함께 OnGetDispInfo 함수도 수정하면 된다. 코드가 복잡하지 않고 길기 때문에 일부만 첨부하였다.

ProcessManager.cpp

프로세스 정보를 가져오는 코드를 작성한 파일이다.

Refresh 함수

```
void ProcessManager::Refresh() {
    std::unique_ptr<BYTE[]> buffer;
    ULONG size = 1 << 20;
    NTSTATUS status;
    do {
        buffer = std::make_unique<BYTE[]>(size);
        // 시스템 정보 검색
        status = ::NtQuerySystemInformation(SystemProcessInformation,
buffer.get(), size, nullptr);
        // 설정된 size보다 정보가 더 많으면 size 증가
        if (status == STATUS_INFO_LENGTH_MISMATCH) {
            size *= 2;
            continue;
        }
        break;
    } while (true);
    // 시스템 프로세스 정보를 가져오지 못했으면 return;
    if (status != STATUS_SUCCESS)
        return;

    bool first = _processes.empty();
    auto existingProcesses(_processesByKey);

    _processes.clear();
    _processes.reserve(first ? 256 : _processesByKey.size() + 10);

    if (first) {
        _processesByKey.reserve(256);
    }

    auto p = (SYSTEM_PROCESS_INFORMATION*)buffer.get();
    for (;;) {
        // (생성 시간, pid)쌍 key 생성
        std::shared_ptr<ProcessInfo> pi;
        auto pid = HandleToUlong(p->UniqueProcessId);
        ProcessKey key(p->CreateTime.QuadPart, pid);
        auto it = _processesByKey.find(key);

        // (생성 시간, pid)key가 없으면...
        if (it == _processesByKey.end()) {
            pi = std::make_unique<ProcessInfo>(key.CreateTime, pid);
            // 생성 시간, pid 입력
            pi->Ppid = HandleToUlong(p->InheritedFromUniqueProcessId);
            // ppid 입력
            pi->Name = pid == 0 ? L"[Idle]" : CString(p->ImageName.Buffer, p-
>ImageName.Length / sizeof(WCHAR));    // 이름 입력
            pi->SessionId = p->SessionId;
            // Session ID 입력
            pi->UserName = GetUserNameFromPid(pid);
            // UserName 입력
```

```

        // Full Path 입력
        HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pid);
        TCHAR exeName[MAX_PATH] = { 0, };
        DWORD dwSize = sizeof(exeName) / sizeof(TCHAR);
        bool res = QueryFullProcessImageName(hProcess, 0, exeName, &dwSize);
        pi->FullPath = exeName;

        _processesByKey.insert({ key, pi });
    }
    // (생성 시간, pid)key가 있으면 위의 정보 입력 x
    else {
        pi = it->second;
        existingProcesses.erase(key);
    }
    _processes.push_back(pi);

    // 추가 정보 갱신
    pi->HandleCount = p->HandleCount;
    pi->ThreadCount = p->NumberOfThreads;
    pi->UserTime = p->UserTime.QuadPart;
    pi->KernelTime = p->KernelTime.QuadPart;
    pi->WorkingSet = p->WorkingSetSize;
    if (p->NextEntryOffset == 0)
        break;
    p = (SYSTEM_PROCESS_INFORMATION*)((BYTE*)p + p->NextEntryOffset);
}

// remove dead processes
for (auto& [key, _] : existingProcesses)
    _processesByKey.erase(key);
}

```

ProcessManager class의 Refresh 함수는 직접 프로세스 정보를 가져오는 함수이다. NtQuerySystemInformation 함수를 통해 시스템 프로세스 정보를 가져오며, 가져온 정보들을 가공하여 의미 있는 정보들을 뽑아내고 정리한다.

```

__kernel_entry NTSTATUS NtQuerySystemInformation(
    [in]          SYSTEM_INFORMATION_CLASS SystemInformationClass,
    [in, out]     PVOID                      SystemInformation,
    [in]          ULONG                      SystemInformationLength,
    [out, optional] PULONG                   ReturnLength
);

```

NtQuerySystemInformation 함수는 시스템 정보를 가져오는 함수로, SystemInformationClass 인자로 SystemProcessInformation를 넘겨주면, 시스템에서 실행 중인 각 프로세스의 정보를 SYSTEM_PROCESS_INFORMATION 형식의 구조체 배열로 넘겨준다. SYSTEM_PROCESS_INFORMATION 구조체의 인자 값들은 아래 링크를 통해 확인해볼 수 있다.

<https://www.geoffchappell.com/studies/windows/km/ntoskrnl/api/ex/sysinfo/process.htm>

본 프로젝트에서 사용한 SYSTEM_PROCESS_INFORMATION 구조체의 인자 값을 간략히 살펴보면, pid 값은 UniqueProcessId 인자에, ppid 값은 InheritedFromUniqueProcessId 인자에, 프로세스 이름은 ImageName 인자에 있는 것을 알 수 있다. 그 외의 다른 값들도 SYSTEM_PROCESS_INFORMATION 구조체에서 확인할 수 있다. 그러나 딱 하나 SYSTEM_PROCESS_INFORMATION 구조체에서 확인할 수 없는 것이 있다. 바로 Full Path 이다. Full Path는 프로세스의 pid 값을 얻어내고, 이를 이용하여

OpenProcess 함수로 프로세스의 handle 값을 얻어낸 뒤, 프로세스의 handle 값을 이용하여 QueryFullProcessImageName 함수로 얻어낼 수 있다.

```
HANDLE OpenProcess(  
    [in] DWORD dwDesiredAccess,  
    [in] BOOL bInheritHandle,  
    [in] DWORD dwProcessId  
);
```

OpenProcess 함수는 dwDesiredAccess 인자에 권한과 bInheritHandle 인자에 상속 여부, dwProcessId 인자에 pid를 넘겨주면 해당 pid의 프로세스 handle을 반환해주는 함수로, 이를 통해 프로세스의 Full Path를 알아낼 수 있다.

```
BOOL QueryFullProcessImageNameA(  
    [in] HANDLE hProcess,  
    [in] DWORD dwFlags,  
    [out] LPSTR lpExeName,  
    [in, out] PDWORD lpdwSize  
);
```

OpenProcess 함수를 통해 프로세스의 handle을 알아냈으면, queryFullProcessImageName 함수를 hProcess 인자에 얻은 handle의 값을, dwFlags 인자에 0, lpExeName에 Full Path를 저장할 배열을 넣고 호출하면 lpExeName에 해당 프로세스의 Full Path가 저장된다.

InformationBar.cpp

InformationBar.cpp은 launcher에서 공유 메모리를 통해 넘겨준 이름, 학번과 이전 과제에서 알아냈듯이 API를 통해 알아낸 시스템 정보들을 수집하고 출력하는 코드가 적힌 파일이다.

Init 함수

```
void CInformationBar::Init() {  
    int count;  
    PWSTR* args = ::CommandLineToArgvW(::GetCommandLine(), &count);  
    if (count == 1) {  
        // 단독 실행 시 공유 메모리 생성  
        m_hSharedMem.reset(::CreateFileMapping(INVALID_HANDLE_VALUE, nullptr,  
        PAGE_READWRITE, 0, 1 << 16, nullptr));  
    }  
    else {  
        // launcher를 통해 실행 시, launcher에서 넘겨준 핸들 값으로 공유 메모리 설정  
        m_hSharedMem.reset((HANDLE)(ULONG_PTR)::wtoi(args[1]));  
    }  
    ::LocalFree(args);  
  
    ATLASSERT(m_hSharedMem);  
  
    WCHAR buffer[1000] = { 0, };  
    int start = 0;  
  
    start = readSharedMemory(buffer);  
    start += useGetProductInfoAPI(buffer + start);  
    start += useiKUSER_SHARED_DATAstruct(buffer + start);  
}
```

```

        start += useGetComputerNameAPI(buffer + start);
//      ::wcscpy_s((PWSTR)buffer, 30, args[1]);
        SetDlgItemText(IDC_INFORMATION, buffer);
    }

```

Init 함수는 View.cpp의 OnCreate 함수에서 호출되는 함수로, 기본 정보들을 가져와서 출력하는 함수이다. 공유 메모리를 설정하는 부분은 project-explorer.cpp 의 Run 함수에서 공유 메모리를 설정하는 부분과 동일하다. 여러 정보들을 하나의 dialog control에 출력하기 위해 정보를 가져오는 여러 함수들을 통해 하나의 buffer에 정보들을 쓴 후, 마지막에 SetDlgItemText 함수를 통해 수집한 정보들을 dialog control에 출력한다.

readSharedMemory 함수

```

int CInformationBar::readSharedMemory(PWSTR text) {
    // 공유 메모리 읽기 권한으로 매핑
    void* buffer = ::MapViewOfFile(m_hSharedMem.get(), FILE_MAP_READ, 0, 0, 0);
    if (!buffer) return 0; // 매핑 실패 시, return

    int start, len1, len2;
    // 공유 메모리 : "[Name]\0[Student ID]\0"

    // "Examiner Name:\t[Name]\n" 문자열 버퍼에 저장
    ::wcscpy_s((PWSTR)text, 17, L"Examiner Name: \t\0");
    start = 16;
    for (len1 = 0; *((PCWSTR)buffer + len1) != L'\0'; len1++);
    ::wcscpy_s((PWSTR)text + start, len1 + 1, (PCWSTR)buffer);
    start += len1;
    ::wcscpy_s((PWSTR)text + start, 2, L"\n\0");
    start += 1;

    // "Examiner ID:\t\t[Student ID]\n" 문자열 버퍼에 저장
    ::wcscpy_s((PWSTR)text + start, 16, L"Examiner ID: \t\t\0");
    start += 15;
    for (len2 = len1 + 1; *((PCWSTR)buffer + len2) != L'\0'; len2++);
    ::wcscpy_s((PWSTR)text + start, len2 - len1, (PCWSTR)buffer + len1 + 1);
    start += len2 - len1 - 1;
    ::wcscpy_s((PWSTR)text + start, 2, L"\n\0");
    start += 1;

    ::UnmapViewOfFile(buffer);

    return start;
}

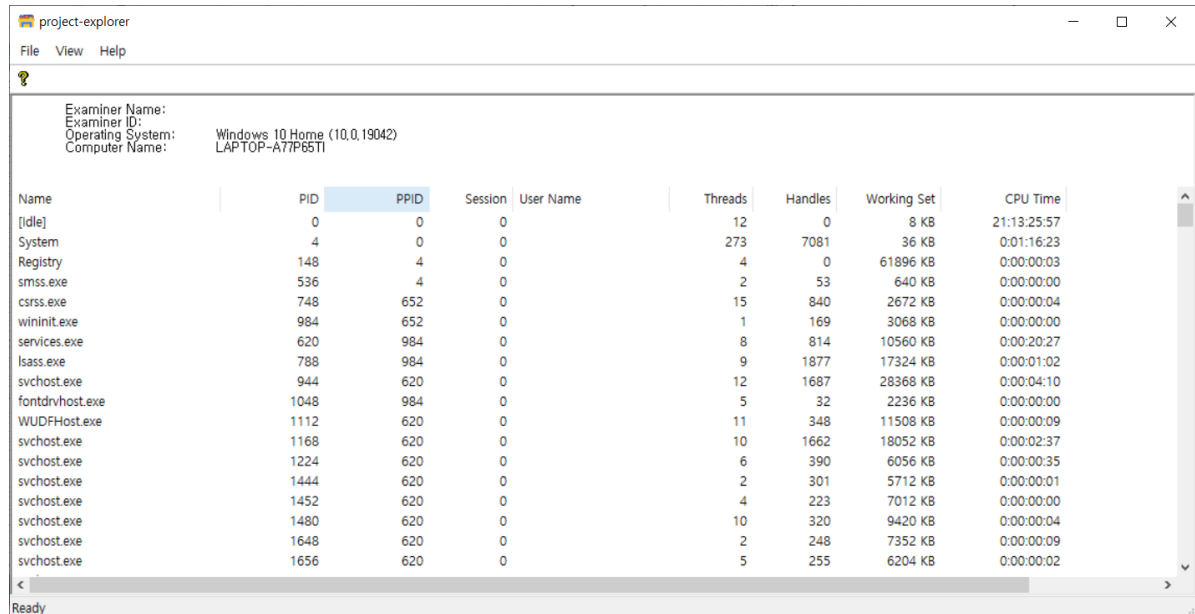
```

readSharedMemory 함수는 공유 메모리에 있는 값을 읽어서 정리하는 함수이다. 공유 메모리에 "[Name]\0[Student ID]\0" 형식으로 정보가 저장되어 있으므로, 처음으로 '\0'가 나오는 부분과 두번째로 '\0'가 나오는 부분을 찾아 문자열을 분리하고 문자열의 길이를 알 수 있다.

시스템 정보 가져오는 함수들

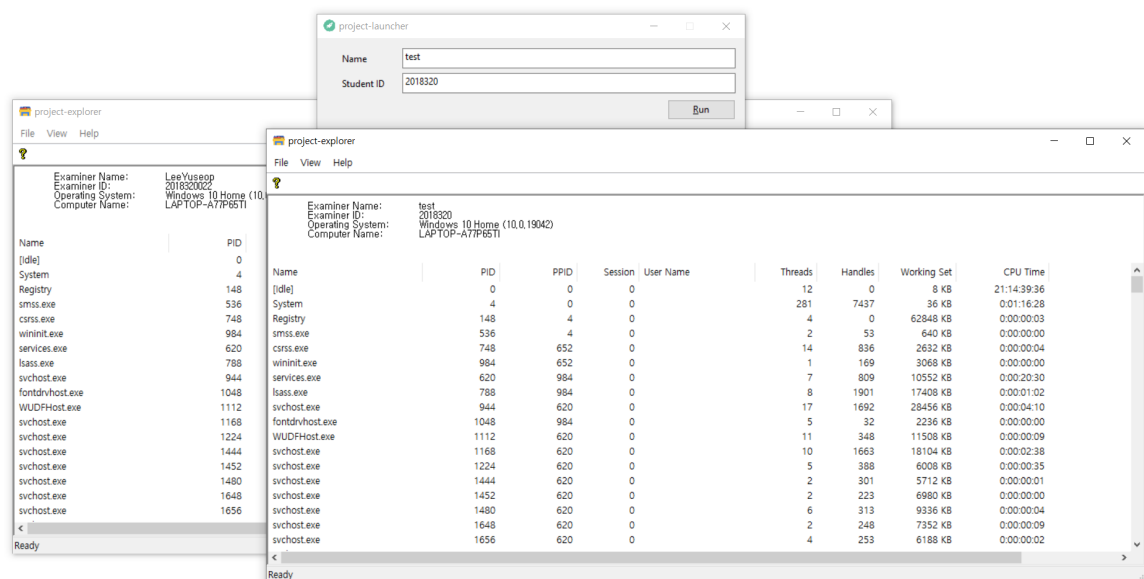
useGetProductInfoAPI 함수와 useiKUSER_SHARED_DATAstruct 함수, useGetComputerNameAPI 함수는 이전 과제에서 직접 만든 함수를 가져와 출력 형식만 바꿨으므로 따로 설명은 하지 않겠다.

S&P Explorer 실행 화면



project-explorer를 실행시킨 결과이다. project-explorer를 단독 실행 시키면 공유 메모리에 이름과 학번이 저장되어 있지 않기 때문에 Examiner Name과 Examiner ID가 빈칸으로 되어 있는 것을 확인할 수 있다. 앞선 Launcher 실행 화면과 비교해보면 project-explorer를 project-launcher를 통해 실행하면 이름과 학번이 적힌다는 것을 알 수 있다.

추가 실행 화면



launcher에서 이름과 학번을 쓰고 Run을 한 뒤에도 계속하여 이름과 학번을 달리하여 Run이 되는 것을 볼 수 있다.

project-explorer

File View Help

Examiner Name: test
Examiner ID: 2018320
Operating System: Windows 10 Home (10.0.19042)
Computer Name: LAPTOP-A77P65TI

Name	PID	PPID	Session	User Name	Threads	Handles	Working Set	CPU Time
[Idle]	0	0	0		12	0	8 KB	21:15:00:44
System	4							0:01:16:30
Registry	148							0:00:00:03
smss.exe	536							0:00:00:00
csrss.exe	748							0:00:00:04
wininit.exe	984							0:00:00:00
services.exe	620							0:00:20:31
lsass.exe	788							0:00:01:02
svchost.exe	944							0:00:04:10
fontdrvhost.exe	1048	984	0		5	32	2236 KB	0:00:00:00
WUDFHost.exe	1112	620	0		11	348	11508 KB	0:00:00:09
svchost.exe	1168	620	0		10	1679	18204 KB	0:00:02:38
svchost.exe	1224	620	0		4	389	5996 KB	0:00:00:35
svchost.exe	1444	620	0		2	301	5712 KB	0:00:00:01
svchost.exe	1452	620	0		2	223	6980 KB	0:00:00:00
svchost.exe	1480	620	0		7	313	9352 KB	0:00:00:04
svchost.exe	1648	620	0		2	248	7352 KB	0:00:00:09
svchost.exe	1656	620	0		4	253	6188 KB	0:00:00:02

Ready

project-launcher

Name: test

Student ID: 2018320

terminated timestamp: 22:41:00

Run

Run을 두번 한 후, 하나를 닫아도 다른 explorer는 잘 돌아가며, launcher에 종료한 explorer의 종료 timestamp가 찍히는 것을 확인할 수 있다.