운영체제 과제 (HW2)

4기 취약점분석트랙 이현정

- **4GB이상의 파일을 메모리에 매핑하려고 할 때, 오류가 발생하는 원인?**

윈도우가 시스템에 장착된 메모리를 사용하기 위해선 정확한 주소가 있어야만 원하는 위치에 제대로 접근(access)할 수 있으므로 모든 메모리 공간에 대한 주소 정보가 필요합니다. 그래서 시스템은 처음에 시작될 때 시스템에 장착된 모든 메모리 공간의 주소 정보를 담은 Physical Address Map(PAM, 물리 메모리 주소 정보)을 만듭니다. 시스템은 이 PAM정보를 윈도우에게 넘겨주고, 윈도우는 넘겨받은 PAM을 토대로 메모리를 인식하고 접근합니다. 즉 윈도우는 PAM에 할당된 메모리만을 사용할 수 있고 만약 PAM에 포함되지 못한 메모리 공간이 있다면, 해당 메모리 공간은 윈도우에서는 인식할 수도 없고 사용할 수도 없습니다.

32비트 CPU는 32비트 주소 지정 방식을 사용하여 할당할 수 있는 전체 메모리 공간의 개수도 32비트, 즉 2^32 * bytes = 4GB가 되었습니다. 따라서 PAM에 할당할 수 있는 전체 메모리 주소는 4GB의 크기를 가지게 되었습니다. 이러한 한계로 인해 32비트 시스템에서 인식하고 사용할 수 있는 전체 메모리의 용량은 4GB가 되었습니다.

따라서 4GB이상의 파일은 한 번에 메모리에 매핑할 수 없습니다.

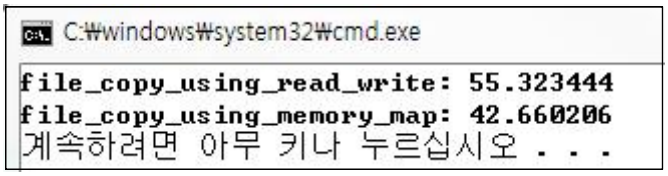
- **해결 코드**

```
// file copy using mmio
DWORD bufSize = 1024;
PUCHAR buf = (PUCHAR)malloc(bufSize);

LARGE_INTEGER offset;
offset.QuadPart = (LONGLONG)0;

StopWatch sw2;
sw2.Start();

FileIoHelper FIOHelper;
FIOHelper.FIOpenForRead(L"D:₩₩test.txt");
FIOHelper.FIOCreateFile(L"D:₩₩test3.txt", Size);

while (offset.QuadPart < Size.QuadPart) {
    if ((Size.QuadPart - offset.QuadPart) >(LONGLONG)bufSize) {
        bufSize = 1024;
    }
    else {
        bufSize = (DWORD)(Size.QuadPart - offset.QuadPart);
    }
    FIOHelper.FIOReadFromFile(offset, bufSize, buf);    // read src file
    FIOHelper.FIOWriteToFile(offset, bufSize, buf);     // write to dst file

    offset.QuadPart += (LONGLONG)bufSize;
}
```

멘토님이 주신 FileIoHelperClass 객체를 생성하고 FIOpenForRead()함수를 이용해 원본 파일을 읽었습니다. 그리고 FIOCreateFile() 함수로 4GB크기의 test3.txt 파일을 생성해 줍니다. 파일을 생성만 해 준 것이지 아직 파일에 내용은 없으므로 이제 Memory Mapped IO 방식으로 파일에 내용을 복사해줍니다.

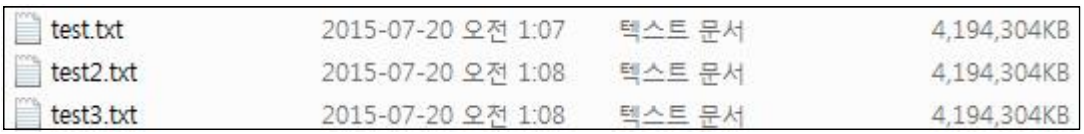
1024MB씩 메모리에 맵핑되고 복사를 하게 됩니다. 다 맵핑&복사가 되고 남은 사이즈가 1024MB보다 작다면 남은 바이트만큼을 메모리에 맵핑&복사시킵니다.

- **결과 화면**



위 화면을 통해 ReadFile(), WriteFile() API로 용량이 4GB인 파일을 복사하는 데 약 55.3초가 걸린 것을 알 수 있고, Memory Mapped IO 방식으로 파일을 복사하는 데 약 42.7초가 걸린 것을 알 수 있습니다.

C드라이브에 용량이 없어서 외장하드로 파일을 복사했는데, 만약 C드라이브에 용량이 있었다면 위 시간보다 더 빨리 파일 복사가 수행됐을 것 같습니다.



test.txt, test2.txt, test3.txt가 생성된 것을 볼 수 있습니다.

test.txt는 create_very_big_file() 함수를 이용하여 만든 복사할 대상이 되는 파일입니다.

test2.txt는 test.txt 파일을 ReadFile(), WriteFile() API를 이용하여 복사한 파일입니다.

test3.txt는 test.txt 파일을 Memory Mapped IO 방식을 이용하여 복사한 파일입니다.