Mapování souborů do paměti ve Windows

Cvičení VIII

Při popisu mapovaní souborů do paměti v operačním systému Microsoft Windows se můžeme opřít o znalosti, které jsme získali v případě unixových OS, protože principy zůstavají stejné a jsou zde jen technické, ale přesto zajímavé, rozdíly. Na tyto rozdíly se v tomto cvičení zaměříme především.

1 Práce se soubory

Mapování souboru do paměti v případě OS Windows je prováděno ve třech krocích.

1. Nejdříve otevřeme soubor pomocí funkce `CreateFile`.[1]
2. Následně pro tento soubor vytvoříme objekt mapování do paměti pomocí funkce `CreateFileMapping`.[2] Tato funkce má jako své argumenty, jednak handle na soubor, otevřený v předchozím kroce, odkaz na bezpečnostní atributy,[3] příznaky ochrany paměti pro namapovanou oblast,[4] maximální velikost namapované oblasti[5] a pojmenování objektu.[6]
3. K samotnému namapování souboru do paměti dojde zavoláním funkce `MapViewOfFile`,[7] které předáme handle na objekt mapování vytvořený v předchozím kroce, příznaky s jakými má být provedeno mapování jednotlivých stránek, začátek oblasti v souboru (offset), která má být namapována do paměti,[8] a velikost oblasti, která má být namapována.

1.1 Čtení dat ze souboru

Následující kód ukazuje mapování obsahu existujícího souboru do paměti.

[1] `https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-createfilea`

[2] `https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-createfilemappinga`

[3] Může být NULL

[4] Např. pouze pro čtení zápis, povolení zápisu, použití copy-on-write. Tyto příznaky musí být kompatibilní s příznaky, které byly použity při otevření souboru.

[5] Tato hodnota je uvedena jako dvě 32bitová slova. Pokud máme existující soubor a uvedeme 0, uvažuje se velikost celého souboru.

[6] To může být NULL, praktické použití uvidíme v dalších příkladech.

[7] `https://learn.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-mapviewoffile`

[8] Tento offset je předán jako dvě 32bitová slova a musí být násobkem velikosti stránky.

```c
#include <windows.h>
#include <tchar.h>
int _tmain() {
    TCHAR* inputFile = _T("foo.txt");
    DWORD length = 20;
    HANDLE hFile = CreateFile(inputFile, GENERIC_READ, FILE_SHARE_READ, NULL,
                              OPEN_EXISTING, 0, NULL);
    if (hFile == INVALID_HANDLE_VALUE) {
        _tprintf(_T("Unable to open: %s\n"), inputFile);
        return 1;
    }
    HANDLE hMapping = CreateFileMapping(hFile, NULL, PAGE_READONLY, 0, 0, NULL);
    if (hMapping == NULL) {
        _tprintf(_T("Unable to create file mapping\n"));
        _tprintf(_T("last error: %i"), GetLastError());
        return 1;
    }
    LPVOID data = MapViewOfFile(hMapping, FILE_MAP_READ, 0, 0, length);
    if (data == NULL) {
        _tprintf(_T("Mapping failed\n"));
        return 1;
    }
    // prace s daty
    char* text = (char*)data;
    //text[0] = 'X';
    _tprintf(_T("%lx\n"), data);
    for (int i = 0; i < length; i++) {
        _tprintf(_T("%c"), text[i]);
    }
    // konec prace s daty
    UnmapViewOfFile(data);
    CloseHandle(hMapping);
    CloseHandle(hFile);
    return 0;
}
```

Tento kód přesně kopíruje výše popsaný postup a namapuje do paměti obsah souboru pouze pro čtení. Jednotlivé operace mohou z mnoha důvodů selhat. Pozor, funkce `CreateFile` při svém selhání vrací hodnotu `INVALID_HANDLE_VALUE`, funkce `CreateFileMapping` a `MapViewOfFile` vrací `NULL`. Častým důvod selhání jednotlivých funkcí je nekompatibilní nastavení oprávnění/příznaků pro přístup k paměti a souborům. Při hledání příčiny selhání nám může být nápomocna funkce `GetLastError()`, která vrací kód chyby, a seznam jednotlivých chybových kodů.[9]

[9] `https://learn.microsoft.com/en-us/windows/win32/debug/system-error-codes`

Pokud bychom chtěli mít možnost upravovat data v paměti, aniž by došlo ke změně souboru, jinými slovy použít techniku copy-on-write, použili bychom na řádku 18 příznak `FILE_MAP_COPY`.

```
LPVOID data = MapViewOfFile(hMapping, FILE_MAP_COPY, 0, 0, length);
```

Po skončení práce s pamětí je nutné oblast odmapovat pomocí funkce `UnmapViewOfFile` a uzavřít všechny otevřené handly.

Úkol č. 1: Vyzkoušejte, že do dané oblasti paměti nelze zapsat. Kód upravte tak, aby do dané oblasti mohlo být zapisováno. Ověřte, že se změny projevily jen v datech, se kterými pracuje aktuální proces, a nejsou promítnuty do vstupního souboru.

1.2 Zápis a vytvoření souboru

Při práci se souborem v režimu pro čtení i zápis pracujeme analogicky jako v případě čtení dat ze souboru. Tj. otevřeme vhodným způsobem soubor:

```
HANDLE hFile = CreateFile(inputFile, GENERIC_READ | GENERIC_WRITE,
                          FILE_SHARE_READ, NULL, CREATE_ALWAYS, 0, NULL);
```

Vytvoříme objekt mapování s právem zápisu:

```
HANDLE hMapping = CreateFileMapping(hFile, NULL, PAGE_READWRITE, 0, length, NULL);
```

A provedeme namapování do paměti, opět s příznakem povolujícím zápis do dané oblasti paměti.

```
LPVOID data = MapViewOfFile(hMapping, FILE_MAP_WRITE, 0, 0, length);
```

Následně můžeme s pamětí pracovat pomocí libovolných vhodných operací, např.:

```
memcpy(data, "Hello world", 11);
```

Úkol č. 2: Uvažujme vstupní soubor, který se bude skládat z řádků, kde na každém řádku bude posloupnost čísel v desítkové soustavě, které jsou odděleny mezerami. Napište program, který tento vstupní soubor převede do formátu, který bude obsahovat znaky `'.'` a `'X'`, kde znak `'X'` bude uveden na pozici, která odpovídá číselné hodnotě ze vstupu.

Příklad vstupu:

```
0
0 1
0 2
0 3
0 1 2 3 4
```

Příklad výstupu:

```
X....
XX...
X.X..
X..X.
XXXXX
```

Program napište tak, aby pro čtení i zápis dat používal mapování souborů do paměti.

Bonusový úkol: Napište druhou variantu programu, která bude používat běžné funkce pro práci se soubory.

Využijte kód z minulého cvičení.

2 Sdílená paměť

V případě operačního systému Windows můžeme mapování souborů do paměti využít k vytvoření sdílené paměti a komunikaci mezi procesy, jako jsme to viděli v minulém cvičení. V přiložených zdrojových kódech je demonstrována komunikace dvou procesů, kde jeden předává data do sdílené schránky a druhý tato data čte.

2.1 Sdílená paměť jako soubor namapovaný do paměti

Při vytváření sdílené paměti mezi těmito procesy nejdříve jeden z nich (v našem případě zapisující proces) vytvoří objekt mapování do paměti a inicializuje tuto oblast. Tento objekt mapování odpovídá mapování souboru do paměti v režimu pro čtení a zápis, avšak je tomuto objektu (posledním argumentem) přiřazeno pojmenování.

```
TCHAR* mboxName = _T("mbox");
hMapping = CreateFileMapping(hFile, NULL, PAGE_READWRITE, 0, sizeof(struct mbox), mboxName);
```

Takto vytvořený objekt může získat druhý proces pomocí funkce `OpenFileMapping`.[10]

```
hMapping = OpenFileMapping(FILE_MAP_ALL_ACCESS, FALSE, mboxName);
```

A následně provést namapování pomocí `MapViewOfFile`.

Analogickým způsobem můžeme sdílet synchronizační prostředky mezi procesy. Jeden proces (v našem případě opět zapisující proces) vytvoří zámek, který bude synchronizovat přístup do sdílené paměti.

```
TCHAR* mboxLockName = _T("mboxlock");
hLock = CreateMutex(NULL, FALSE, mboxLockName);
```

A druhý proces k tomuto zámku získá přístup pomocí funkce `OpenMutex`[11].

[10] `https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-openfilemappinga`
[11] `https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-openmutexw`

```
hLock = OpenMutex(SYNCHRONIZE, FALSE, mboxLockName);
```

Poznámka: Zde vidíme, jak se ze standardních synchronizačních nástrojů, které jsme dosud používali pouze pro vlákna, mohou stát nástroje pro synchronizaci procesů.

2.2 Anonymní sdílená paměť

V předchozím případě jsme pro vytvoření sdílené paměti použili soubor. To nemusí být vždy žádoucí. Pokud chceme namapovat paměť, která není svázana s žádným běžným souborem,[12] můžeme funkci `CreateFileMapping` předat místo handle otevřeného souboru hodnotu `INVALID_HANDLE_VALUE`. V takovém případě dojde k namapování stránek ze stránkovacího souboru.[13]

Tuto paměť můžeme sdílet mezi procesy, protože k její identifikaci slouží pojmenování objektu, které uvedeme u funkce `CreateFileMapping`. Ukázkový příklad můžeme upravit následovně.

```
TCHAR* mboxName = _T("mbox");
hMapping = CreateFileMapping(INVALID_HANDLE_VALUE, NULL, PAGE_READWRITE, 0,
                             sizeof(struct mbox), mboxName);
```

Úkol č. 3 Vytvořte třetí typ procesu, který bude kontinuálně sledovat obsah sdílené schránky, a bude zobrazovat informaci, v jakém stavu se schránka nachází (případně obsah zprávy). Tento typ procesu by neměl obsah schránky měnit.

3 Ochrana paměti

Podobně jako v unixových operačních systémech můžeme využít vlastností mapování souboru do paměti k tomu, abychom vytvořili oblasti paměti, které mají specifické vlastnosti z pohledu ochrany přístupu, např. jsou jen pro čtení, umožňují spouštět kód apod.

Následující kód ukazuje, jak vytvořit oblast paměti, kde je uložený spustitelný kód. Konkrétně se jedná o funkci, která vrací hodnotu svého argumentu zvýšenou o jedna. Tato funkce v assembleru a strojovém kódu vypadá následovně:

```
0: 89 c8                    mov    eax,ecx
2: ff c0                    inc    eax
4: c3                       ret
```

```
typedef int (*intfun)(int);
char data[] = { 0x89, 0xc8, 0xff, 0xc0, 0xc3 };
int _tmain(){
    int length = sizeof(data);
    HANDLE hMapping = CreateFileMapping(INVALID_HANDLE_VALUE, NULL, PAGE_EXECUTE_READWRITE,
                                        0, length, NULL);
```

[12] Tj. ekvivalent mapování anonymní paměti v unixech.
[13] `pagefile.sys`

```
    if (hMapping == NULL) /* ... */
    LPVOID exData = MapViewOfFile(hMapping, FILE_MAP_ALL_ACCESS | FILE_MAP_EXECUTE,
                                  0, 0, length);
    memcpy(exData, data, length);
    intfun f = (intfun)exData;
    _tprintf(_T("%i\n"), f(10));
    CloseHandle(hMapping);
}
```