

# SO Cheat Sheet

## Memoria virtuală

### Mapare fişiere şi memorie POSIX

În POSIX trebuie incluse header-ele **sys/types.h**, **sys/mman.h**.

`void *mmap(void *start, size_t length, int prot, int flags, int fd, off_t offset)` – mapare fişier/memorie în spaţiul de adresă al unui proces

**start**            adresa de start pentru mapare, NULL înseamna lipsa unei preferinţe; dacă e diferită de NULL, e considerată doar un hint, maparea fiind creată la o adresă apropiată multiplu de dimensiunea unei pagini

**length**        lungimea mapării

**prot**            tipul de acces la zona de memorie: PROT\_READ, PROT\_WRITE, PROT\_EXEC, PROT\_NONE

**flags**            tipul de mapare: cel puțin una din MAP\_PRIVATE/MAP\_SHARED, MAP\_FIXED, MAP\_LOCKED; MAP\_ANONYMOUS pentru mapare memorie

**fd**              descriptorul fişierului mapat; ignorat la mapare memorie

**offset**          offset în cadrul fişierului mapat; ignorat la mapare memorie

*întoarce*        adresa mapării, MAP\_FAILED (care reprezintă (void \*) -1) în caz de eroare

`int msync(void *start, size_t length, int flags)` – declanșează în mod explicit sincronizarea fişierului cu maparea din memorie

**start,**            identifică zona de memorie

**length**

**flags**            MS\_SYNC, MS\_ASYNC, MS\_INVALIDATE

*întoarce*        0 în caz de succes, -1 în caz de eroare

`int munmap(void *start, size_t length)` – demapează o zonă din spaţiul de adresă al procesului ; poate identifica zone aparținând unor mapari diferite, unele deja demapate

`void *mremap(void *old_address, size_t old_size, size_t new_size, unsigned long flags)` – redimensionare zonă mapată

**old\_**            identifică vechea zonă mapată

**address,**

**old\_size**

**new\_size**        noua dimensiune

**flags**            MREMAP\_MAYMOVEE

*întoarce*        pointer spre noua zonă în caz de succes, MAP\_FAILED în caz de eroare

`int mprotect(const void *addr, size_t len, int prot)` – schimbă drepturile de acces ale unei mapări

**addr**            adresa mapării, multiplu de dimensiunea unei pagini

**len**             lungimea zonei considerate; se va aplica pentru toate paginile cu cel puțin un byte în intervalul [addr, addr + len -1]

**prot**            PROT\_READ,        PROT\_WRITE,        PROT\_EXEC, PROT\_NONE

*întoarce*        0 în caz de succes, -1 în caz de eroare

`int madvise(void *start, size_t length, int advice)` – indicații despre cum va fi folosită o zonă mapată

**addr,**            identifică zona

**length**

**advice**          MADV\_NORMAL,    MADV\_RANDOM,    MADV\_SEQUENTIAL,    MADV\_WILLNEED,    MADV\_DONT-NEED

*întoarce*        0 în caz de succes, -1 în caz de eroare

### Blocarea paginării POSIX

`int mlock(const void *addr, size_t len)` – blochează paginarea paginilor incluse în intervalul [addr, addr + len - 1]

`int mlockall(int flags)` – blochează paginarea tuturor paginilor procesului

**flags**            MCL\_CURRENT, MCL\_FUTURE

`int munlock(const void *addr, size_t len)` – reporni paginarea tuturor paginilor din intervalul [addr, addr + len - 1]

`int munlockall(void)` – reporni paginarea tuturor paginilor procesului

### Excepții accesare memorie POSIX

Funcția de tip sigaction va primi ca parametru o structură siginfo\_t, având setate:

**si\_signo**        SIGSEGV, SIGBUS

**si\_code**        caz SIGSEGV: SEGV\_MAPPER, SEGV\_ACCERR; caz SIGBUS: BUS\_ADRALN, BUS\_ADRERR, BUS\_OB-JERR

**si\_addr**        adresa care a generat excepția

### ElectricFence

Folosit pentru depanare buffer overrun. Pentru a preveni și buffer underrun, definiți variabila de mediu EF\_PROTECT\_BELOW.

Programul trebuie linkat cu efence: -lefence.

### Maparea fişierelor Win32

`HANDLE CreateFileMapping( HANDLE hFile, LPSECURITY_ATTRIBUTES lpAttributes, DWORD flProtect, DWORD dwMaximumSizeHigh, DWORD dwMaximumSizeLow, LPCTSTR lpName )` – crează un obiect FileMapping, pentru a fi mapat

- **hFile** - handle fişier de mapat
- **lpAttributes** - attribute securitate pentru acces handle întors
- **flProtect** - tipul mapării: PAGE\_READONLY, PAGE\_READWRITE, PAGE\_WRITECOPY
- **dwMaximumSizeHigh, dwMaximumSizeLow** - dimensiunea maximă
- **lpName** - şir identificare, opțional
- **întoarce** - în caz de succes întoarce un handle către un obiect FileMapping, în caz de eșec întoarce NULL

`LPVOID MapViewOfFile( HANDLE hFileMappingObject, DWORD dwDesiredAccess, DWORD dwFileOffsetHigh, DWORD dwFileOffsetLow, SIZE_T dwNumberOfBytesToMap )` – creează mapare fişier

- **hFileMappingObject** - obiect de tip FileMapping
- **dwDesiredAccess** - FILE\_MAP\_READ, FILE\_MAP\_WRITE, FILE\_MAP\_COPY
- **dwFileOffsetHigh, dwFileOffsetLow** - offset
- **dwNumberOfBytesToMap** - număr octeți de mapat
- **întoarce** - în caz de succes întoarce un pointer la zona mapată, în caz de eșec întoarce NULL

`BOOL UnmapViewOfFile( LPCVOID lpBaseAddress )` – demapare fişier mapat în memorie

- **lpBaseAddresst** - adresa început zonă
- **întoarce** - TRUE pentru succes, FALSE altfel

### Mapare memorie Win32

`LPVOID VirtualAlloc( LPVOID lpAddress, SIZE_T dwSize, DWORD flAllocationType, DWORD flProtect )` – alocă memorie în spatiul procesului curent

`LPVOID VirtualAllocEx( HANDLE hProcess, LPVOID lpAddress, SIZE_T dwSize, DWORD flAllocationType, DWORD flProtect )` – alocă memorie în spaţiul altui proces

- **hProcess** - handle proces pentru care se aplică funcția
- **fAllocationType** - MEM\_RESERVE, MEM\_COMMIT, MEM\_RESET
- **lpAddress** - adresa unde începe alocarea; multiplu de 4KB pentru alocare și 64KB pentru rezervare; NULL - nicio preferință
- **dwSize** - dimensiunea zonei
- **flProtect** - modul de acces pentru zona alocată: PAGE\_EXECUTE, PAGE\_EXECUTE\_READ, PAGE\_EXECUTE\_READWRITE, PAGE\_EXECUTE\_WRITECOPY, PAGE\_READONLY, PAGE\_READWRITE, PAGE\_WRITECOPY, PAGE\_NOACCESS, PAGE\_GUARD, PAGE\_NOCACHE
- **întoarce** - pointer la zona alocată

BOOL VirtualFree( LPVOID lpAddress, SIZE\_T dwSize, DWORD dwFreeType ) – eliberează zona de memorie din spațiul procesului curent

BOOL VirtualFreeEx( HANDLE hProcess, LPVOID lpAddress, SIZE\_T dwSize, DWORD dwFreeType ) – eliberează o zonă de memorie pentru alt proces

- **hProcess** -handle proces pentru care se aplică funcția
- **lpAddress, dwSize** - identifică zona
- **dwFreeType** - tipul operației: MEM\_DECOMMIT, MEM\_RELEASE
- **întoarce** - TRUE - succes, FALSE - eroare

BOOL VirtualProtect( LPVOID lpAddress, SIZE\_T dwSize, DWORD flNewProtect, PDWORD lpflOldProtect ) – schimbarea protecției unei zone de memorie mapate în spațiul procesului curent

BOOL VirtualProtectEx( HANDLE hProcess, LPVOID lpAddress, SIZE\_T dwSize, DWORD flNewProtect, PDWORD lpflOldProtect ) – schimbarea protecției unei zone de memorie mapate în alt proces

- **Process** - handle proces pentru care se aplică funcția
- **lpAddress, dwSize** - identifică zona
- **flNewProtect** - noi drepturi
- **lpflOldProtect** - salvare drepturi vechi
- **întoarce** - TRUE - succes, FALSE - eroare

Observație: funcționează doar pentru pagini din aceeasi regiune rezervată alocată cu apelul VirtualAlloc sau VirtualAllocEx folosind MEM\_RESERVE.

## Interogarea zonelor mapate Win32

DWORD VirtualQuery( LPCVOID lpAddress, PMEMORY\_BASIC\_INFORMATION lpBuffer, SIZE\_T dwLength ) – interogarea zonelor mapate din procesul curent  
DWORD VirtualQueryEx( HANDLE hProcess, LPCVOID lpAddress, PMEMORY\_BASIC\_INFORMATION lpBuffer, SIZE\_T dwLength ) – interogarea zonelor mapate din alt proces

- **hProcess** - handle proces pentru care se aplică funcția
- **lpAddress** - adresa din cadrul zonei de interogare
- **lpBuffer** - buffer ce reține informații despre zonă
- **dwLength** - număr octeți scriși în buffer
- **întoarce** - 0 - nicio informație furnizată, diferit de 0 - altfel

## Blocarea paginării Win32

BOOL VirtualLock( LPVOID lpAddress, SIZE\_T dwSize ) – blocare paginare proces curent

BOOL VirtualLockEx( HANDLE hProcess, LPVOID lpAddress, SIZE\_T dwSize ) – blocare paginare alt proces

- **hProcess**handle proces pentru care se aplică funcția
- **lpAddress, dwSize** sunt luate în calcul paginile cu macar un octet în [lpAddress, lpAddress + dwSize]
- **întoarce** TRUE - succes, FALSE - altfel

BOOL VirtualUnlock( LPVOID lpAddress, SIZE\_T dwSize ) – repornirea paginării pentru procesul curent

BOOL VirtualUnlockEx( HANDLE hProcess, LPVOID lpAddress, SIZE\_T dwSize ) – repornirea paginării pentru alt proces

## Excepții Win32

PVOID AddVectoredExceptionHandler( ULONG FirstHandler, PVECTORED\_EXCEPTION\_HANDLER VectoredHandler ) – adaugă o funcție de tratare excepții

- **firstHandler** - adaugare la început sau la final lista de tratat excepții
- **Vectored Handler** - funcția de tratat excepții

ULONG RemoveVectoredExceptionHandler( PVOID VectoredHandlerHandle ) – elimină o funcție de tratat excepții

LONG WINAPI VectoredHandler( PEXCEPTION\_POINTERS ExceptionInfo ) – semnatura funcției de tratare a excepțiilor

struct \_EXCEPTION\_POINTERS: PEXCEPTION\_RECORD  
ExceptionRecord; PCONTEXT ContextRecord;

struct \_EXCEPTION\_RECORD: DWORD ExceptionCode;  
DWORD ExceptionFlags; struct \_EXCEPTION\_RECORD\*  
ExceptionRecord; PVOID ExceptionAddress; DWORD  
NumberParameters; ULONG\_PTR  
ExceptionInformation[EXCEPTION\_MAXIMUM\_PARAMETERS];

- **ExceptionCode** va fi setat la EXCEPTION\_ACCESS\_VIOLATION sau EXCEPTION\_DATATYPE\_MISALIGNMENT
- **Exception Address** va fi setat la adresa instrucțiunii care a cauzat excepția
- **Number Parameters** va fi setat la 2
- **Exception Information** prima intrare e 0 pt citire, 1 pentru scriere; a doua intrare e adresa ce a generat excepția