SO Cheat Sheet

Memoria virtuala

Mapare fisiere si memorie POSIX

In 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 fișier/memorie în spațiul de adresă al unui proces

`start`	adresa de start pt mapare, NULL inseamna lipsa unei preferinte; trebuie sa fie multiplu de dimensiunea unei pagini
`length`	lungimea maparii
`prot`	tipul de acces la zona: PROT_READ, PROT_WRITE, PROT_EXEC, PROT_NONE
`flags`	tipul de mapare: cel putin una din MAP_PRIVATE/MAP_SHARED, MAP_FIXED, MAP_LOCKED; MAP_ANONYMOUS pt mapare memorie
`fd`	descriptorul fisierului mapat; ignorat la mapare memorie
`offset`	offset in cadrul fisierului mapat; ignorat la mapare memorie
întoarce	adresa maparii, MAP_FAILED in caz de eroare

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

`start, length`	identifica zona de memorie
`flags`	MS_SYNC, MS_ASYNC, MS_INVALIDATE
întoarce	0 in caz de succes, -1 in caz de eroare

`int munmap(void *start, size_t length)` – demapeaza o zona din spațiul de adresă al procesului ; poate identifica zone apartinand unor mapari diferite, unele deja demapate

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

`old_address, old_size`	identifica vechea zona mapata
`new_size`	noua dimensiune
`flags`	MREMAP_MAYMOVEE
întoarce	pointer spre noua zona, MAP_FAILED in caz de eroare

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

`addr`	adresa maparii, multiplu de dimensiunea unei pagini
`len`	lungimea zonei considerate; se va aplica pt toate paginile cu cel putin un byte in intervalul [addr, addr + len -1]
`prot`	PROT_READ, PROT_WRITE, PROT_EXEC, PROT_NONE
întoarce	0 in caz de succes, -1 in caz de eroare

`int madvise(void *start, size_t length, int advice)` – indicatii despre cum va fi folosita o zona mapata

`addr, length`	identifica zona
`advice`	MADV_NORMAL, MADV_RANDOM, MADV_SEQUENTIAL, MADV_WILLNEED, MADV_DONTNEED
întoarce	0 in caz de succes, -1 in caz de eroare

Blocarea paginarii POSIX

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

`int mlockall(int flags)` – blocheaza 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

Exceptii accesare memorie POSIX

Funcția de tip sigaction va primi ca parametru o structură siginfo_t, avand setate:

`si_signo`	SIGSEGV, SIGBUS
`si_code`	caz SIGSEGV: SEGV_MAPPER, SEGV_ACCERR; caz SIGBUS: BUS_ADRALN, BUS_ADRERR, BUS_OBJERR
`si_addr`	adresa care a generat exceptia

ElectricFence

Folosit pentru depanare buffer overrun. Pentru a preveni si buffer underrun, definiti variabila de mediu EF_PROTECT_BELOW.

Programul trebuie linkat cu efence: -lefence.

Maparea fisierelor Win32

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

- **hFile** - handle fisier de mapat
- **lpAttributes** - atribute securitate pentru acces handle intors
- **flProtect** - tipul maparii: PAGE_READONLY, PAGE_READWRITE, PAGE_WRITECOPY
- **dwMaximumSizeHigh, dwMaximumSizeLow** - dimensiunea maxima
- **lpName** - sir identificare, optional
- **întoarce** - handle către obiect FileMapping

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

- **hFileMappingObject** - obiect de tip FileMapping
- **dwDesiredAccess** - FILE_MAP_READ, FILE_MAP_WRITE, FILE_MAP_COPY
- **dwFileOffsetHigh, dwFileOffsetLow** - offset
- **dwNumberOfBytesToMap** - numar octeti de mapat
- **întoarce** - pointer la zona mapata

`BOOL UnmapViewOfFile( LPCVOID lpBaseAddress )` – demapare fisier mapat in memorie

- **lpBaseAddresst** - adresa inceput zona
- **întoarce** - TRUE pentru succes, FALSE altfel

Mapare memorie Win32

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

`LPVOID VirtualAllocEx( HANDLE hProcess, LPVOID lpAddress, SIZE_T dwSize, DWORD flAllocationType, DWORD flProtect )` – aloca memorie in spatiul altui proces

- **hProcess** - handle proces pentru care se aplica functia
- **fAllocationType** - MEM_RESERVE, MEM_COMMIT, MEM_RESET
- **lpAddress** - adresa unde incepe alocarea; multiplu de 4KB pentru alocare și 64KB pentru rezervare; NULL - nicio preferinta
- **dwSize** - dimensiunea zonei
- **flProtect** - modul de acces pentru zona alocata: 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 alocata

`BOOL VirtualFree( LPVOID lpAddress, SIZE_T dwSize, DWORD dwFreeType )` – elibereaza zona de memorie in spatiul procesului curent

`BOOL VirtualFreeEx( HANDLE hProcess, LPVOID lpAddress, SIZE_T dwSize, DWORD dwFreeType )` – elibereaza zona de memorie pentru alt proces

- **hProcess** -handle proces pentru care se aplica functia
- **lpAddress, dwSize** - identifica zona
- **dwFreeType** - tipul operatiei: MEM_DECOMMIT, MEM_RELEASE
- **întoarce** - TRUE - succes, FALSE - eroare

`BOOL VirtualProtect( LPVOID lpAddress, SIZE_T dwSize, DWORD flNewProtect, PDWORD lpflOldProtect )` – schimbarea protectiei unei zone de memorie mapate in spatiul procesului curent

`BOOL VirtualProtectEx( HANDLE hProcess, LPVOID lpAddress, SIZE_T dwSize, DWORD flNewProtect, PDWORD lpflOldProtect )` – schimbarea protectiei unei zone de memorie mapate in alt proces

- **Process** - handle proces pentru care se aplica functia
- **lpAddress, dwSize** - identifica zona
- **flNewProtect** - noi drepturi
- **lpflOldProtect** - salvare drepturi vechi
- **întoarce** - TRUE - succes, FALSE - eroare

Observatie: functioneaza 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 aplica functia
- **lpAddress** - adresa din cadrul zonei de interogat
- **lpBuffer** - buffer ce retine informatii despre zona
- **dwLength** - numar octeti scrisi in buffer
- **întoarce** - 0 - nicio informatie furnizata, diferit de 0 - altfel

Blocarea paginarii 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 aplica functia
- **lpAddress, dwSize**sunt luate in calcul paginile cu macar un octet in [lpAddress, lpAddess + dwSize]
- **întoarce**TRUE - succes, FALSE - altfel

`BOOL VirtualUnlock( LPVOID lpAddress, SIZE_T dwSize )` – repornirea paginarii pentru procesul curent

`BOOL VirtualUnlockEx( HANDLE hProcess, LPVOID lpAddress, SIZE_T dwSize )` – repornirea paginarii pentru alt proces

Exceptii Win32

`PVOID AddVectoredExceptionHandler( ULONG FirstHandler, PVECTORED_EXCEPTION_HANDLER VectoredHandler )` – adauga o functie de tratare exceptii

- **firstHandler** - adaugare la inceput sau la final lista de tratat exceptii
- **Vectored Handler** - functia de tratat exceptii

`ULONG RemoveVectoredExceptionHandler( PVOID VectoredHandlerHandle )` – elimina o functie de tratat exceptii

`LONG WINAPI VectoredHandler( PEXCEPTION_POINTERS ExceptionInfo )` – semnatura functiei de tratare a exceptiilor

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 exceptia