SO Cheat Sheet
Memoria virtuală

Mapare fișiere și 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 fișier/memorie în spațiul de adresă al unui proces

`start`	adresa de start pentru mapare, NULL înseamna lipsa unei preferințe; 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 fișierului mapat; ignorat la mapare memorie
`offset`	offset în cadrul fișierului 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 fișierului cu maparea din memorie

`start, length`	identifică zona de memorie
`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 spațiul 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_address, old_size`	identifică vechea zonă mapată
`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, length`	identifică zona
`advice`	MADV_NORMAL, MADV_RANDOM, MADV_SEQUENTIAL, MADV_WILLNEED, MADV_DONTNEED
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_OBJERR
`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 fișierelor 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 fișier de mapat
- **lpAttributes** - atribute securitate pentru acces handle întors
- **flProtect** - tipul mapării: PAGE_READONLY, PAGE_READWRITE, PAGE_WRITECOPY
- **dwMaximumSizeHigh, dwMaximumSizeLow** - dimensiunea maximă
- **lpName** - șir 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 fișier

- **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 fișier 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 spațiul 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 protectiei 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: functionează 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 interogat
- **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 in calcul paginile cu macar un octet în [lpAddress, lpAddess + 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