SO Cheat Sheet

Comunicare între procese

Semafoare POSIX

Trebuie inclus headerul **semaphore.h**

`sem_t* sem_open(const char *name, int oflag [, mode_t mode, unsigned int value])` – creează / deschide un semafor existent

`name`	identifică semaforul
`oflag`	O_CREAT sau/și O_EXCL
`mode`	specifică permisiunile noului semafor
`value`	valoarea inițială
întoarce	adresa semaforului

În caz de eroare, întoarce SEM_FAILED și setează errno (EACCESS, EEXIST, EINVAL, EMFILE, ENAMETOOLONG, ENFILE, ENOENT, ENOMEM)

`int sem_wait(sem_t *sem)` – decrementează semaforul, blocându-se dacă semaforul avea valoarea 0

`int sem_trywait(sem_t *sem)` – decrementează semaforul neblocant

În caz de eroare, atât sem_wait() cât și sem_trywait() întorc -1 și setează errno la EINTR, EINVAL(sau EAGAIN pentru sem_trywait())

`int sem_post(sem_t *sem)` – incrementează semaforul

`int sem_getvalue(sem_t *sem, int *sval)` – află valoarea semaforului

`int sem_close(sem_t *sem)` – închide semaforul

În caz de eroare, atât sem_post() și sem_getvalue(), cât și sem_close() întorc -1 și setează errno la EINVAL

`int sem_unlink(const char *name)` – șterge semaforul

În caz de eroare, întoarce -1 și setează errno (EACCESS, ENAMETOOLONG, ENOENT)

Cozi de mesaje POSIX

Trebuie inclus headerul **mqueue.h**

`mqd_t mq_open(const char *name, int oflag [, mode_t mode, struct mq_attr *attr])` – creează / deschide o coadă de mesaje existent

`name`	identifică coada de mesaje
`oflag`	opțiunea trebuie să conțină exact una din următoarele O_RDONLY, O_WRONLY, O_RDWR și oricâte din O_NONBLOCK, O_CREAT, O_EXCL
`mode`	specifică permisiunile
`attr`	NULL - atributele implicite
întoarce	descriptorul cozii de mesaje

`mqd_t mq_send(mqd_t mqdes, const char *buffer, size_t length, unsigned priority)` – adaugă mesajul la coadă

`mqdes`	descriptorul cozii de mesaje
`buffer`	mesajul
`length`	lungimea mesajului
`priority`	o valoare nenegativă ce specifică prioritatea mesajului
întoarce	0

`ssize_t mq_receive(mqd_t mqdes, char *buffer, size_t length, unsigned *priority)` – scoate cel mai vechi mesaj cu cea mai mare prioritate din coadă

`mqdes`	descriptorul cozii de mesaje
`buffer`	mesajul
`length`	lungimea mesajului
`priority`	o valoare nenegativă ce specifică prioritatea mesajului
întoarce	numărul de octeți ai mesajului

În caz de eroare, atât mq_send(), cât și mq_receive() întorc -1 și setează errno (EAGAIN, EBADF, EMSGSIZE, EINTR, EINVAL, ETIMEDOUT)

`mqd_t mq_close(mqd_t mqdes)` – închide descriptorul cozii de mesaje

`mqdes`	descriptorul cozii de mesaje
întoarce	0

În caz de eroare, întoarce -1 și setează errno la EBADF

`mqd_t mq_unlink(const char *name)` – șterge coada de mesaje

`name`	identifică coada de mesaje
întoarce	0

În caz de eroare, întoarce -1 și setează errno (EACCES, ENAMETOOLONG, ENOENT)

Memorie partajată POSIX

Trebuie incluse headerele **sys/types.h**, **sys/mman.h**, **fcntl.h**

`int shm_open(const char *name, int flags, mode_t mode)` – creează / deschide o zonă de memorie.

`name`	identifică zona de memorie
`flags`	opțiunea trebuie să conțină exact una dintre O_RDONLY sau O_RDWR și oricâte din O_CREAT, O_EXCL, O_TRUNC
`mode`	specifică permisiunile
întoarce	un descriptor

În caz de eroare, întoarce -1 și setează errno (EACCES, EEXIST, EINVAL, EMFILE, ENAMETOOLONG, ENFILE, ENOENT)

`int ftruncate(int fd, off_t length)` – dimensionează zona de memorie

`fd`	descriptor ce specifică zona
`length`	dimensiune
întoarce	0

În caz de eroare, întoarce -1 și setează errno (EBADF sau EINVAL)

`void *mmap(void *address, size_t length, int protection, int flags, int fd, off_t offset)` – mapează zona de memorie partajată în spaţiul de memorie al procesulu

`address`	preferinţă asupra adresei unde se va face maparea
`length`	dimensiunea
`protection`	specifică permisiunile - trebuie să fie în concordanţă cu modul de deschidere a fd PROT_EXEC, PROT_READ, PROT_WRITE, PROT_NONE
`flag`	opţiuni asupra tipului - de exemplu MAP_FIXED, MAP_SHARED, MAP_PRIVATE
`fd`	descriptorul
`offset`	poziţia de început
întoarce	adresa la care s-a realizat maparea

În caz de eroare întoarce MAP_FAILED.

`int munmap(void *address, size_t length)` – realizează demaparea

`address`	adresa de început
`length`	dimensiunea
întoarce	0 (în caz de eroare întoarce -1 si errno setat)

`int shm_unlink(const char *name)` – şterge zona de memorie partajată

`name`	identifică coada de mesaje
întoarce	0

Mutex Win32

`HANDLE CreateMutex( LPSECURITY_ATTRIBUTES lpMutexAttr, BOOL bInitialOwner, LPCTSTR lpName )` – crează un mutex

- **lpMutexAttr** - pointer la o structură de tip SECURITY_ATTRIBUTES, dacă e NULL handle-ul nu poate fi moștenit
- **bInitialOwner** - TRUE - proprietarul iniţial este cel care l-a creat
- **lpName** - numele mutexului
- **întoarce** - handle către mutex

`HANDLE OpenMutex( DWORD dwDesiredAccess, BOOL bInheritHandle, LPCTSTR lpName )` – deschide un mutex

- **dwDesireAccess** - dreturile de acces
- **bInheritHandle** - TRUE - handle-ul poate fi moștenit de procesele copil
- **lpName** - numele mutexului
- **întoarce** - handle către mutex

`BOOL ReleaseMutex( HANDLE hMutex )` – cedează posesia mutexului

- **hMutex** - handle către mutex
- **întoarce** - o valoare diferită de 0

Funcţii de asteptare

`DWORD SignalObjectAndWait( HANDLE hObjToSignal, HANDLE hObjToWaitOn, DWORD dwMilliseconds, BOOL bAlertable )`

- **hObjToSignal** - handle către obiect de semnalizat
- **hObjToWaitOn** - handle către obiect de aşteaptat
- **dwMilliseconds** - interval de timeout (0 până la INFINITE)
- **bAlertable** - TRUE - alertează
- **întoarce** - WAIT_FAILED, WAIT_IO_COMPLETION, WAIT_ABANDONED, WAIT_OBJECT_0, WAIT_TIMEOUT

`DWORD WaitForSingleObject( HANDLE hHANDLE, WORD dwMillisec )`

`DWORD WaitForSingleObjectEx( HANDLE hHandle, DWORD dwMilliseconds, BOOL bAlertable )`

- **hHandle** - handle către obiect
- **dwMilliseconds** - interval de timeout (0 până la INFINITE)
- **bAlertable** - TRUE - alertează
- **întoarce** - WAIT_FAILED, WAIT_IO_COMPLETION, WAIT_ABANDONED, WAIT_OBJECT_0, WAIT_TIMEOUT

`DWORD WaitForMultipleObjects( DWORD nCount, const HANDLE* lpHandles, BOOL bWaitAll, DWORD dwMilliseconds )`

`DWORD WaitForMultipleObjectsEx( DWORD nCount, const HANDLE* lpHandles, BOOL bWaitAll, DWORD dwMilliseconds, BOOL bAlertable )`

- **nCount** - numărul de handle-uri
- **lpHandles** - un vector cu handle către obiecte
- **bWaitAll** - TRUE - se întoarce dacă starea tuturor obiectelor e signal se întoarce când starea vreunui obiect e signal
- **dwMilliseconds** - interval de timeout (0 până la INFINITE)
- **bAlertable** - TRUE - alertează
- **întoarce** - WAIT_FAILED, WAIT_IO_COMPLETION, WAIT_ABANDONED, WAIT_OBJECT_0, WAIT_TIMEOUT

Semafoare Win32

`HANDLE CreateSemaphore( LPSECURITY_ATTRIBUTES lpSemAttr, LONG lInitialCount, LONG lMaximumCount, LPCTSTR lpNAME )` – crează / deschide un semafor

- **lpSemAttr** - pointer la o structură de tip SECURITY_ATTRIBUTES (NULL handle-ul nu poate fi moştenit)
- **lInitialCount** - valoarea iniţială a semaforului
- **lMaximumCount** - valoarea maximă a semaforului
- **lpNAME** - numele semaforului
- **întoarce** - handle către semafor

`HANDLE OpenSemaphore( DWORD dwDesiredAccess, BOOL bInheritHandle, LPCTSTR lpNAME )` – deschide un semafor existent

- **dwDesireAccess** - dreturile de acces
- **bInheritHandle** - TRUE - handle-ul poate fi moştenit
- **lpNAME** - numele semaforului
- **întoarce** - handle către semafor

`BOOL ReleaseSemaphore( HANDLE hSemaphore, LONG lReleaseCount, LPLONG lpPreviousCount )` – incrementează semaforul

- **hSemaphore** - handle către semafor
- **lReleaseCount** - cu cat se incrementează valoarea semaforului
- **lpPreviousCount** - valoarea precedentă a semaforului
- **întoarce** - o valoare diferită de 0

Cozi de mesaje Win32

`HANDLE CreateMailslot( LPCTSTR lpName, DWORD nMaxMessageSize, DWORD lReadTimeout, LPSECURITY_ATTRIBUTES lpSecAttr )` – creează o coadă de mesaje

- **lpName** - numele cozii de mesaje
- **nMaxMessSize** - dimensiunea maximă a unui mesaj - 0 pentru nelimitat
- **lReadTimeout** - timpul, în milisecunde, în care o operaţie de citire aşteaptă ca un mesaj să fie scris înainte de timeout MAILSLOT_WAIT_FOREVER
- **lpSecAttr** - NULL - handle nu poate fi mostenit
- **întoarce** - handle către coada de mesaje

`BOOL GetMailslotInfo( HANDLE hMailslot, LPDWORD lpMaxMessageSize, LPDWORD lpNextSize, LPDWORD lpMessageCount, LPDWORD lpReadTimeout )` – întoarce informaţii despre coada de mesaje

- **hMailslot** - handle către coada de mesaje
- **lpMaxMessSize** - dimensiunea maximă a unui mesaj
- **lpNextSize** - dimensiunea următorului mesaj sau MAILSLOT_NO_MESSAGE
- **lpMessageCount** - numărul de mesaje in asteptare să fie citite
- **lpReadTimeout** - timpul, în milisecunde, în care o operaţie de citire aşteaptă ca un mesaj să fie scris înainte de timeout
- **întoarce** - o valoare diferită de 0

Ultimii 3 parametrii pot fi NULL.

`BOOL SetMailslotInfo( HANDLE hMailslot, DWORD lReadTimeout )`

- **hMailslot** - handle către coada de mesaje
- **lReadTimeout** - timpul, în milisecunde, în care o operaţie de citire asteaptă ca un mesaj să fie scris înainte de timeout
- **întoarce** - o valoare diferită de 0

Memorie partajată Win32

`HANDLE CreateFileMapping( HANDLE hFile, LPSECURITY_ATTRIBUTES lpAttributes, DWORD flProtect, DWORD dwMaxSizeHigh, DWORD dwMaxSizeLow, LPCTSTR lpName )` – creează un obiect de tipul File Mapping

- **hFile** - handle către fişierul din care se va crea obiectul FileMapping
- **lpAttributes** - dacă e NULL handle-ul nu poate fi moştenit
- **flProtect** - specifică opţiunea de protecţie PAGE_EXECUTE_READ, PAGE_READONLY, PAGE_EXECUTE_READWRITE, PAGE_READWRITE, PAGE_EXECUTE_WRITECOPY, PAGE_WRITECOPY
- **dwMaxSizeHigh** - partea semnificativă din dimensiunea maximă
- **dwMaxSizeLow** - partea nesemnificativă din dimensiunea maximă
- **lpName** - numele obiectului de tip FileMapping
- **întoarce** - handle către obiectul de tip FileMapping

`LPVOID MapViewOfFile( HANDLE hFileMapObject, DWORD dwDesiredAccess, DWORD dwFileOffsetH, DWORD dwFileOffsetL, SIZE_T dwNrOfBytesToMap )` – mapează o zonă de memorie

- **hFileMapObject** - handle către un obiect de tip FileMapping
- **dwDesireAccess** - tipul accesului ce determină protecţia paginilor : FILE_MAP_ALL_ACCESS, FILE_MAP_EXECUTE, FILE_MAP_COPY, FILE_MAP_READ, FILE_MAP_WRITE
- **dwFileOffsetH** - partea semnificativă ce marchează inceputul
- **dwFileOffsetL** - partea nesemnificativă din ce marchează inceputul
- **dwNrBytesToMap** - numărul de octeţi mapaţi
- **întoarce** - adresa de inceput a zonei mapată

`HANDLE OpenFileMapping( DWORD dwDesiredAccess, BOOL bInheritHandle, LPCTSTR lpName )` – accesează un obiect de tipul FileMapping

- **dwDesireAccess** - drepturile de acces
- **bInheritHandle** - TRUE - handle poate fi mostenit
- **lpName** - numele obiectului de tip FileMapping
- **întoarce** - handle deschis către obiectul de tip FileMapping

`BOOL UnmapViewOfFile( LPCVOID lpBaseAddress )` – demapează o zonă de memorie

- **lpBaseAddress** - pointer către adresa de bază a zonei mapate
- **întoarce** - o valoare diferită de 0