SO Cheat Sheet
Semnale

Linux

Trebuie inclus header-ul `signal.h`

Descrierea semnalelor

`char *strsignal(int sig)` – întoarce descrierea textuală a unui semnal

`void psignal(int sig, const char *s)` – afişează descrierea textuală a unui semnal, alături de mesajul dat ca parametru

Măşti de semnale

`int sigemptyset(sigset_t *set)` – elimină toate semnalele din mască

`int sigfillset(sigset_t *set)` – adaugă toate semnalele la mască

`int sigaddset(sigset_t *set, int signo)` – adaugă semnalul precizat la mască

`int sigdelset(sigset_t *set, int signo)` – elimină semnalul precizat din mască

`int sigismember(sigset_t *set, int signo)` – verifică daca semnalul precizat aparţine măştii

Blocarea semnalelor

`int sigprocmask(int how, const sigset_t *set, sigset_t *oldset)` – obţine sau modifică masca de semnale a firului apelant

`how`	unul dintre `SIG_BLOCK`, `SIG_UNBLOCK`, `SIG_SETMASK`
`set`	masca ce conţine noile semnale blocate/deblocate
`oldset`	vechea mască de semnale
întoarce	0 succes, -1 eroare

Tratarea semnalelor

`sighandler_t signal(int signum, sighandler_t handler)` – stabileşte acţiunea efectuată la primirea unui semnal

`signum`	numărul semnalului
`handler`	una din valorile `SIG_IGN`, `SIG_DFL` sau adresa unei funcţii de tratare
întoarce	adresa handler-ului anterior sau `SIG_ERR` în caz de eroare

`int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact)` – stabileşte acţiunea efectuată la primirea unui semnal

`signum`	numărul semnalului
`act`	noua acţiune de executat
`oldact`	vechea acţiune
întoarce	0 succes, -1 eroare

Semnalarea proceselor

`int kill(pid_t pid, int sig)` – trimite un semnal unui proces, fără a garanta recepţia

`pid`	procesul destinaţie
`sig`	semnalul trimis
întoarce	0 succes, -1 eroare

`int sigqueue(pid_t pid, int signo, const union sigval value)` – trimite un semnal unui proces, garantând recepţia

`pid`	procesul destinaţie
`signo`	semnalul trimis
`value`	informaţie suplimentară, ce însoţeşte semnalul, şi care poate fi obţinută din câmpul `siginfo_t->si_value`
întoarce	0 succes, -1 eroare

Aşteptarea semnalelor

`int sigsuspend(const sigset_t *mask)` – înlocuieşte, temporar, masca de semnale, şi se blochează în aşteptarea unui semnal neblocat

`mask`	masca temporară
întoarce	întotdeauna -1

Timer-e

`int timer_create(clockid_t clockid, struct sigevent *evp, timer_t *timerid)` – crearea unui timer

`clockid`	specifică tipul ceasului: CLOCK_REALTIME, CLOCK_MONOTONIC, CLOCK_PROCESS_CPUTIME_ID, CLOCK_THREAD_CPUTIME_ID
`evp`	specifică modul de notificare la expirarea timer-ului
`timerid`	întoarce identificatorul timer-ului
întoarce	0 succes, -1 eroare

`int timer_settime(timer_t timerid, int flags, const struct itimerspec *new_value, struct itimerspec * old_value)` – armarea unui timer

`timerid`	identificator timer
`flags`	poate fi 0 sau TIMER_ABSTIME
`new_value`	noii parametrii ai timer-ului
`old_value`	vechii parametrii
întoarce	0 succes, -1 eroare

`int timer_delete(timer_t timerid)` – ștergerea unui timer

`timerid`	identificator timer
întoarce	0 succes, -1 eroare

Windows

Waitable Timer Objects

`HANDLE WINAPI CreateWaitableTimer(LPSECURITY_ATTRIBUTES lpAttributes, BOOL bManualReset, LPCTSTR lpTimerName)` – creează sau deschide un timer

- **lpAttributes** - permite moştenirea handle-ului timer-ului în procesele copil
- **bManualReset** - dacă este `TRUE`, timer-ul rămâne în starea *signaled* până ce se apelează `SetWaitableTimer`
- **lpTimerName** - numele timer-ului
- **întoarce** - handle-ul timer-ului, `NULL` în caz de eroare

`BOOL WINAPI SetWaitableTimer(HANDLE hTimer, const LARGE_INTEGER *pDueTime, LONG lPeriod, PTIMERAPCROUTINE pfnRoutine, LPVOID lpRoutineArg, BOOL fResume)` – creează sau deschide un timer

- **hTimer** - handle-ul timer-ului
- **pDueTime** - primul interval, după care expiră timer-ul, în multipli de 100 ns
- **lPeriod** - perioada timer-ului, în milisecunde
- **pfnRoutine** - adresa funcţiei executate la expirare (opţional)
- **lpRoutineArg** - parametrul funcţiei de tratare (opţional)
- **fResume** - dacă este `TRUE`, şi timer-ul intră în starea *signaled*, sistemul aflat în starea de conservare a energiei îşi reia activitatea
- **întoarce** - `TRUE` pentru succes

`BOOL WINAPI CancelWaitableTimer(HANDLE hTimer)` – dezactivează un timer

- **hTimer** - handle-ul timer-ului

`DWORD WINAPI WaitForSingleObject(HANDLE hHandle, DWORD dwMilliseconds)` – permite aşteptarea expirării unui timer