

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)` – afișează descrierea textuală a unui semnal, alături de mesajul dat ca parametru

Măști 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 dacă semnalul precizat aparține măștii

Blocarea semnalelor

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

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

Tratarea semnalelor

`sighandler_t signal(int signum, sighandler_t handler)` – stabilește acțiunea efectuată la primirea unui semnal

`signum` numărul semnalului
`handler` una din valorile `SIG_IGN`, `SIG_DFL` sau adresa unei funcții 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)` – stabilește acțiunea efectuată la primirea unui semnal

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

Semnalarea proceselor

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

`pid` procesul destinație
`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 recepția

`pid` procesul destinație
`signo` semnalul trimis
`value` informație suplimentară, ce însoțește semnalul, și care poate fi obținută din câmpul `siginfo_t->si_value`
`întoarce` 0 succes, -1 eroare

Așteptarea semnalelor

`int sigsuspend(const sigset_t *mask)` – înlocuiește, temporar, masca de semnale, și se blochează în așteptarea 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 parametri ai timer-ului
`old_value` vechii parametri
`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, LPCWSTR lpTimerName)` – creează sau deschide un timer

- `lpAttributes` - permite moștenirea 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 funcției executate la expirare (opțional)
- `lpRoutineArg` - parametrul funcției de tratare (opțional)
- `fResume` - dacă este `TRUE`, și timer-ul intră în starea *signaled*, sistemul aflat în starea de conservare a energiei își 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 așteptarea expirării unui timer