

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

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

Tratarea semnalelor

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

<code>signum</code>	numărul semnalului
<code>handler</code>	una din valorile <code>SIG_IGN</code> , <code>SIG_DFL</code> sau adresa unei funcții de tratare
<code>întoarce</code>	adresa handler-ului anterior sau <code>SIG_ERR</code> în caz de eroare

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

<code>signum</code>	numărul semnalului
<code>act</code>	noua acțiune de executat
<code>oldact</code>	vechea acțiune
<code>întoarce</code>	0 succes, -1 eroare

Semnalarea proceselor

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

<code>pid</code>	procesul destinație
<code>sig</code>	semnalul trimis
<code>întoarce</code>	0 succes, -1 eroare

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

<code>pid</code>	procesul destinație
<code>signo</code>	semnalul trimis
<code>value</code>	informație suplimentară, ce însoțește semnalul, și care poate fi obținută din câmpul <code>siginfo_t->si_value</code>
<code>întoarce</code>	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

<code>mask</code>	masca temporară
<code>întoarce</code>	întotdeauna -1

Timer-e

`unsigned int alarm(unsigned int seconds)` – pregătește trimiterea unui semnal `SIGALRM`, la expirarea unei anumit interval de timp

<code>seconds</code>	numărul de secunde după care se dorește livrarea semnalului
<code>întoarce</code>	numărul de secunde rămase pâna la expirarea unei eventuale planificări anterioare

`int setitimer(int which, const struct itimerval *new_value, struct itimerval *old_value)` – pregătește trimiterea unui semnal `SIGALRM`, la expirarea unui anumit interval de timp, oferind un control mai fin, și posibilitatea reluării

<code>which</code>	timpul luat în considerare, putând lua una din valorile <code>ITIMER_REAL</code> , <code>ITIMER_VIRTUAL</code> , <code>ITIMER_PROF</code>
<code>new_value</code>	noii parametri ai timer-ului
<code>old_value</code>	vechii parametri
<code>întoarce</code>	0 succes, -1 eroare

Windows

Evenimente de la consolă

`BOOL WINAPI SetConsoleCtrlHandler(PHANDLER_ROUTINE HandlerRoutine, BOOL Add)` – înregistrează un handler de tratare a evenimentelor de consolă

`HandlerRoutine` adresa handler-ului de tratare a evenimentului. Poate fi `NULL` dacă se dorește ignorarea evenimentului `CTRL+C`

`Add` `TRUE` pentru adăugarea handler-ului, `FALSE` pentru înlăturarea lui

`întoarce` `TRUE` pentru succes

Waitable Timer Objects

`HANDLE WINAPI CreateWaitableTimer(LPSECURITY_ATTRIBUTES lpAttributes, BOOL bManualReset, LPCTSTR 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

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