Système d'exploitation 2

SIGNAUX

Signal message réduit à un booléen qui peut être envoyé à un / plusieurs processus.
↳ Notifier des évent. <signal.h>
↳ Simplifier la communication entre les processus.

5 actions

ABORT	arrêt du processus
DUMP	
IGNORE	infos sans arrêt
STOP	passe le processus en endormi
CONTINUE	passage d'endormi à run.

3 Réponses à un signal

- exécuter l'info par défaut
- ignorer certains signaux ⚠ SIGKILL
- intercepter et invoquer une fonction gestionnaire_signal

2 Phases de transmission d'un signal

émission maj du descripteur du processus pour enregistrer le nouveau signal émis
réception le noyau force le processus destinataire à réagir en donnant une réponse.

Signal pendant signal envoyé en attente de réception
↳ Si deux signaux pendants du même id arrive, le second sera détruit et jamais pris en compte
↳ Seul les processus en cours d'exec peuvent recevoir des signaux.

Structure d'un signal

_NSIG nombre de signaux du système
↳ Linux = 64

⚠ 0 associé à aucun système

Dans le descripteur de processus

signal	stocke les signaux envoyés par le processus
blocked	stocke les signaux bloqués
sigpending	flag si ⩾ 1 signaux en attente
gsig	pointeur vers signal_struct permet de décrire comment chaque signal doit être géré.

La structure sigaction permet de définir comment un programme doit réagir à un signal.

sa_handler	action à réaliser ↳ pointeur vers la f^{ct} de gestion ↳ SIG_DFL action par défaut ↳ SIG_IGN ignorer
sa_flags	ensemble de flag qui précise la gestion du signal
sa_mask	ensemble des signaux à masquer pdt l'exécution du gestionnaire de signal.

On peut afficher la liste des signaux sys_siglist[]

Envoyer un signal

```
int kill(pid_t pid, int n_sig);
```

pid_t pid : processus auquel je veut envoyer le signal

int n_sig : macros correspondant au signal à envoyé SIGUSR4, SIGINT, ⋮

>0	process classique
= 0	tous les process du ⩵ groupe que l'appelant
-1	tous les process sauf init et l'appelant.

Envoyer un signal interne
int kill(getpid(), n_sig);
int raise(int n_sig);
} instructions équivalentes

Réception d'un signal
signal ⇒ Contrôler les signaux facilement
sigaction ⇒ Contrôle précis du gestionnaire
void (* signal (int n_sig, void (*gestionnaire)(int)))(int)
n° du signal — nom du gestionnaire de signal

Exemple : signal (SIGINT, handler);
le nom du gestionnaire

⚠ renvoie SIG_ERR en cas d'échec
⚠ SIG_KILL & SIG_STOP ne sont pas contrôlable

int sigaction (int n_sig
const struct sigaction * nouv
const struct sigaction * ancien);

Il est possible de gérer des ensembles de signaux
Initialisation d'un ensemble de signaux
int sigemptyset (sigset_t* ens);
↳ 0 ok
↳ -1 erreur
pointeur vers l'ensemble de signaux.

Remplir un ensemble
int sigfillset (sigset_t* ens);

Ajout d'un signal dans l'ensemble
int sigaddset (sigset_t* ens, int n_sig);

Suppression d'un signal dans l'ensemble
int sigdelset (sigset_t* ens, int n_sig);

Un ensemble est vide ?
int sigisemptyset (const sigset_t* ens);

Blocage de signal
int sigprocmask (int mth, // opération
const sigset_t * ens, // ens concerné
sigset_t * svg); // ancien ens de signaux

→ mth
↳ SIG_BLOCK bloquer les signaux
↳ SIG_UNBLOCK débloquer les signaux
↳ SIG_SETMASK remplacer le masque par ens

Récupérer l'ensemble des signaux pendants
int sigpending (sigset_t* ens);
Où seront stocké les signaux en attente

Attente d'un signal
int pause (void);

Les alarmes
SIGALARM est utilisé pour effectuer des temporisations
unsigned int alarm (const unsigned int seconds);
- second = 0 déprog. de l'alarme
- prog. d'une alarme
  ↳ suppression de l'ancienne

⚠ On ne peut programmer une seule alarme