Tips to remember

Un peu de cours

Signaux UNIX

Les différents états d'un signal

- **Envoyé** d'un process émetteur à un recepteur
- **Pendant**, n'a pas été traité par le recepteur : il peut être bloqué par ce dernier
  - /!\ Si un autre signal du même type arrive, alors il sera perdu !
- **Délivré** est pris en compte par le processus destinataire.

Remarque : un processus en mode maître ne peut pas être interrrompu par un signal

La prise en compte d'un signal, ou handler

Lorsqu'il n'est pas personnalisé, le handler suit un traitement prédéfini (routine) par défaut dans le systeme.
Dans le PCB du processus, un structure permet de gérer les signaux (mémorisation de ceux en attente, masquage, pointeurs vers les handlers…)

Les 5 traitements par défaut

- terminaison du process
- terminaison du process avec image mémoire (fichier core)
- ignoré
- suspension du process
- continuation

Les principaux

Nom du signal	**Evenement associé**	**Traitement par défaut**
SIGALRM	Fin d'une fonction alarm (temporisation)	Terminaison du process

Nom du signal	**Evenement associé**	**Traitement par défaut**
SIGCHLD	Terminaison d'un fils	Signal ignoré
SIGINT	Interruption "intr" demandée par le terminal (*type* `Ctrl+c`)	Terminaison du process
SIGQUIT	Interruption "quit" demandée par le terminal	Terminaison du process avec fichier core
SIGKILL	Signal de terminaison classique	Terminaison du process
SIGSEGV	Violation d'un segment mémoire	Terminaison du process avec fichier core

Let's practice

La structure sigaction

```
struct sigaction{
    void (* sa_handler)(); //pointeur sur handler ou SIG_DFL ou SIG_IGN (ces deux d
    sigset_t sa_mask; //liste signaux supplémentaires à bloquer éventuellement
    int sa_flags; /indicateurs optionnels
}
```

Seul l'attribut `sa_handler` est obligatoire.

La fonction sigaction()

```
#include <signal.h>
int sigaction(int sig, struct sigaction * new_handler, struct sigaction * old_handl
```

Tips

- On va déclarer nos pid et sigaction en tant que variables globales (en dehors du main) pour pouvoir les utiliser dans les handlers
- Après l'utilisation de la fonction alarm, penser à mettre une boucle infinie pour éviter que le programme ne se termine avant le délai imposé
- On défini le sa_handler avant l'appel à la fonction sigaction

- Les sigaction se déclarent dans le corps du fils ou du père
- Le old_handler n'importe pas : il peut être égal à null si c'est la première fois qu'on déclare notre signal.
- On utilise fflush(stdout) après chaque printf() pour épurer notre sortie standart

À plus d'un fils

Lorsqu'on veut faire communiquer au moins 3 processus (qui se créent en cascade), il est intelligent de penser à déclarer un tableau permettant de retrouver les pid des autres : ils pourront ainsi s'envoyer des signaux et communiquer.

```
int lespid[3];


int main(int argc, char const * argv[]) {
        lespid[0] = getpid();
        int n = fork();
        if (n == 0){
                lespid[1] = getpid();
                int m = fork();
                if (m == 0){
                        lespid[2] = getpid();
                        fonction_fils2();
                } else {
                        lespid[2] = m;
                        fonction_fils1();
                }
        }
        else {
                lespid[1] = n;
                fonction_pere();
        }
}
```