Cours 3
Signaux

Signaux

- **Mécanisme de communication de base**
  - Un signal est une information transmise à un programme durant son exécution.
    - A chaque signal est associée une valeur entière positive non nulle et strictement inférieure à **NSIG (**constante non POSIX)
    - C'est par ce mécanisme que le système communique avec les processus utilisateurs :
      - en cas d'erreur (violation mémoire, erreur d'E/S),
      - à la demande de l'utilisateur lui-même via le clavier (caractères d'interruption ctrl-C, ctrl-Z...),
      - lors d'une déconnection de la ligne/terminal, etc.
    - Possibilité d'envoi d'un signal entre processus.
    - Traitement par défaut.

Signaux: Les principaux signaux POSIX

Nom	*Événement*	*comportement*
Terminaison		
SIGINT	ctrl-C	terminaison
SIGQUIT	<QUIT> ctrl-\	terminaison + core
SIGKILL	Tuer un processus	terminaison
SIGTERM	Signal de terminaison	terminaison
SIGCHLD	Terminaison ou arrêt d'un processus fils	ignoré
SIGABRT	Terminaison anormale	terminaison + core
SIGHUP	Déconnexion terminal	terminaison

Signaux: Les principaux signaux POSIX

Nom	*Événement*	*comportement*
Suspension/reprise		
SIGSTOP	Suspension de l'exécution	suspension
SIGTSTP	Suspension de l'exécution (ctrl-Z)	suspension
SIGCONT	Continuation du processus arrêté	reprise
Fautes		
SIGFPE	erreur arithmétique	terminaison + core
SIGBUS	erreur sur le bus	terminaison + core
SIGILL	instruction illégale	terminaison + core
SIGSEGV	violation protection mémoire	terminaison + core
SIGPIPE	Erreur écriture sur un tube sans lecteur	terminaison

Signaux: Les principaux signaux POSIX

Nom	*Événement*	*comportement*
Autres		
SIGALRM	Fin de temporisation	terminaison
SIGUSR1	Réservé à l'utilisateur	terminaison
SIGUSR2	Réservé à l'utilisateur	terminaison
SIGTRAP	Trace/breakpoint trap	terminaison + core
SIGIO	E/S asynchrone	terminaison

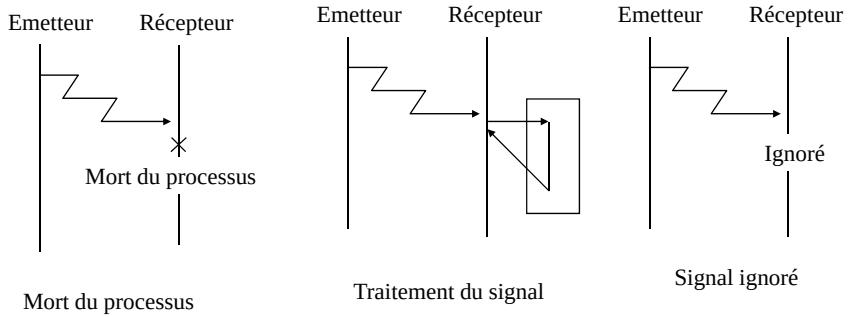
SIGNAUX

- **A chaque signal est associé une valeur**
  - **"/usr/include/signal.h"**
  - **Liste des signaux:**
    ```
    $ kill -l
    ```
  - **Utiliser plutôt le nom de la constante au lieu du numéro**
    - **Exemple: SIGKILL (=9), SIGINT (=2), etc.**
      - **kill –KILL <num. proc>; kill –INT <num. proc>**
  - Envoyer un signal revient à envoyer ce numéro à un processus. Tout processus a la possibilité d'émettre à destination d'un autre processus un signal, à condition que ses numéros de propriétaires (UID) lui en donnent le droit vis-à-vis de ceux du processus récepteur.

Signaux - Terminologie

- **Signal pendant**
  - Signal qui a été envoyé à un processus mais qui n'a pas encore été pris en compte.
    - Cet envoi est mémorisé dans le BCP du processus.
    - Si un exemplaire d'un signal arrive à un processus alors qu'il en existe un exemplaire pendant, le signal est **perdu.**
- **Délivrance**
  - Un signal est délivré à un processus lorsque le processus le prend en compte et réalise l'action qui lui est associée.
    - La délivrance a lieu lorsque le processus **passe de l'état actif noyau à l'état actif utilisateur** : retour appel système, retour interruption matérielle, élection par l'ordonnanceur**.**
- **Signal masqué ou bloqué**
  - La délivrance du signal est ajournée

4. SIGNAUX: Conséquence de la délivrance d'un signal



- **Ne pas confondre avec les interruptions**
  - Matérielles : int. horloge, int. Disque, etc.

SIGNAUX: Délivrance d'un signal

- **Comportement par défaut**
  - Terminaison du processus
  - Terminaison du processus avec production d'un fichier de nom *core*
  - Signal ignoré
  - Suspension du processus (*stopped* ou *suspended*)
  - Continuation du processus

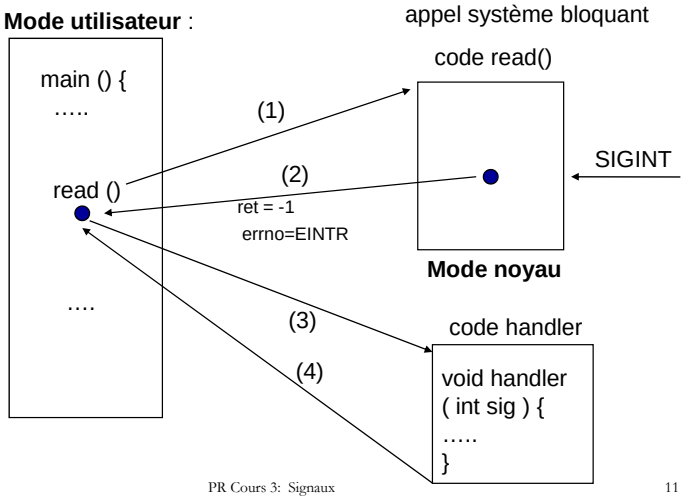
- **Installation d'un nouveau handler (sigaction) ***
  - **SIG_IGN** (ignorer le signal)
  - **Fonction** définie par l'utilisateur
  - **SIG_DFL** (restituer le comportement par défaut)

  * Applicable à tous les signaux sauf **SIGKILL**, **SIGSTOP**

Signaux : Délivrance d'un signal –appel système priorité interruptible

- **L'arrivée d'un signal à un processus endormi à un niveau de priorité interruptible le réveille**
  - Processus passe à l'état prêt
  - Le signal sera délivré lors de l'élection du processus
    - Fonction *handler* associée sera exécutée
  - Exemples d'appels système interruptibles:
    - *pause*,
    - *sigsuspend*,
    - *Wait/waitpid*
    - *read*, *write*,
    - etc.

Délivrance d'un signal – appel système priorité interruptible



Signaux : L'envoi des signaux (kill)

- **Appel système**
  - **int kill (pid_t pid, int signal)**
    - Par défaut la réception d'un signal provoque la terminaison

  pid:
  pid: processus d'identité *pid*
  0 : tous les processus dans le même groupe
  -1 : non défini par POSIX. Tous les processus du système
  < -1 : tous les processus du groupe |*pid*|

  signal:
  valeur entre 0 et NSIG (0 = test d'existence)

- **Commande**
  - $ **kill -l** liste des signaux
  - $ **kill -sig pid** envoi d'un signal

Exemple – kill

```
#include <sys/types.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

int main (int arg, char** argv) {

    printf ("debut application \n");

/* envoyer un SIGINT à soi-même */
  kill(getpid ( ), SIGINT);
  printf ("fin application \n");

    return EXIT_SUCCESS;
}
```


Signaux : Masquage signaux

- **Signaux bloqués ou masqués**
  - Leur délivrance est différée
  - Même s'ils se trouvent pendants il ne sont pas délivrés
  - Fonction pour masquer et démasquer des signaux
  - Pendant l'exécution du handler associé à un signal, celui-ci est bloqué (norme POSIX)
    - Possibilité de le débloquer dans le handler associé
  - Un processus fils:
    - n'hérite pas des signaux pendants
    - hérite du masque de signaux et du handler
    - *fork()* suivi par un *exec() :* réinitialisation dans le fils avec les handlers par défaut.

Signaux : Manipulation des ensembles de signaux

- **Fonctions qui ne changent pas les signaux eux-mêmes mais permettent de manipuler des variables "ensembles de signaux".**
  - int sigemptyset(sigset_t *set);
  - int sigfillset(sigset_t *set);
  - int sigaddset(sigset_t *set, int sig);
  - int sigdelset(sigset_t *set, int sig);
  - int sigismember(sigset_t *set, int sig);

    (retourne !=0 si signal présent)

Signaux : Manipulation des ensembles de signaux

sigset_t sig; /*ensemble de tous les signaux = Π*/

- sigemptyset(&sig);
  - sig = Ø
- sigfillset(&sig);
  - sig=Π
- sigemptyset(&sig); sigaddset(&sig, SIGINT); sigaddset(&sig, SIGQUIT);
  - sig={SIGINT, SIGQUIT}
- sigfillset(&sig); sigdelset(&sig, SIGINT); sigdelset(&sig, SIGQUIT);
  - sig= Π /{SIGINT, SIGQUIT}

Signaux : Masquage des signaux

- **Blocage des signaux:**
  - Un processus peut installer un masque de signaux à l'exclusion de SIGKILL et SIGSTOP
  - Le traitement des signaux est retardé
    - signal pendant.
  - Un processus fils hérite le masque de signaux mais non pas les signaux pendants
  - Liste des signaux pendants bloqués:
    - **int sigpending (sigset_t *set);**

Signaux : Masquage des signaux

- **Appel à la fonction sigprocmask**
- **int sigprocmask(int how, const sigset_t *set, sigset_t *old);**

how : SIG_BLOCK : bloquer en plus les signaux positionnés dans set
SIG_UNBLOCK : démasquer
SIG_SETMASK : bloquer uniquement les signaux dans set

set : masque de signaux

old: valeur du masque antérieur, si non NULL

- Le nouveau masque est formé par *set*, ou composé par *set* et le masque antérieur

Signaux : Exemple – masquage signaux

sigprocmask-ex.c

```
#define _XOPEN_SOURCE 700

#include <signal.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv) {

 sigset_t sig_proc;
 printf ("Debut application \n");

sigemptyset(&sig_proc);
sigaddset (&sig_proc, SIGINT);
sigprocmask(SIG_BLOCK,&sig_proc, NULL);

  sleep (10);
  printf("apres sleep \n");
  sigprocmask(SIG_UNBLOCK,&sig_proc, NULL);
  printf("fin programme \n");

  return EXIT_SUCCESS;
}
```

> **sigprocmask-ex**
Debut application
>**ctrl-C**
après sleep

SIGINT

SIGINT pendant

délivré

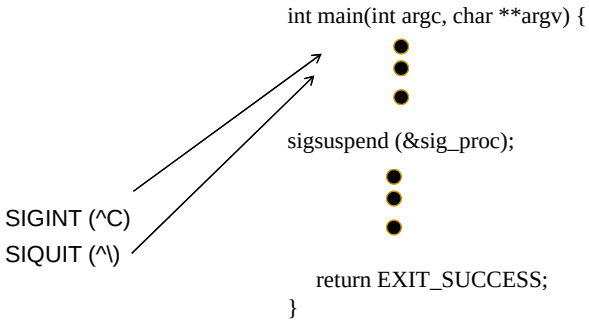
Exemple – signaux pendants

```
int main (int argc, char* argv []) {

        sigset_t  sig_set;     /* liste des signaux bloqués */

        sigemptyset (&sig_set);
        sigaddset (&sig_set,SIGINT);

        /*masquer le signal SIGINT */
        sigprocmask (SIG_SETMASK, &sig_set, NULL);

        kill (getpid (), SIGINT);

        /* obtenir la liste des signaux pendants */
         sigpending (&sig_set);

        if (sigismember ( &sig_set,SIGINT) )
          printf("SIGINT pendant \n");

        return EXIT_SUCCESS;
}
```

Signaux : Attente d'un SIGNAL

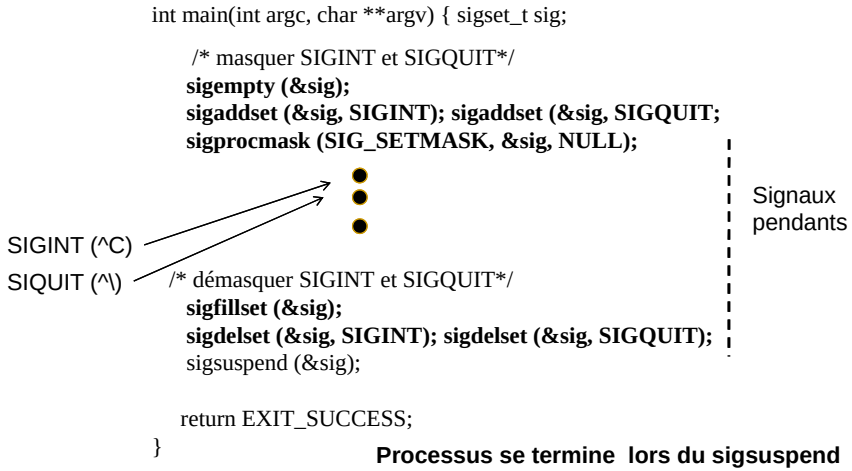
- **Processus passe à l'état « stoppé ». Il est réveillé par l'arrivée d'un signal non masqué**
  - **int pause (void)**
    - Ne permet ni d'attendre l'arrivée d'un signal de type donné, ni de savoir quel signal a réveillé le processus.
  - **int sigsuspend (cons sigset_t *p_ens)**
    - Installation du masque des signaux pointé par *p_ens*. Le masque d'origine est réinstallé au retour de la fonction.

Signaux : sigsuspend



Processus se termine avant le sigsupend

Signaux : Exemple – sigprocmask + sigsuspend



Processus se termine lors du sigsuspend

Signaux : Changement du traitement par défaut

```
struct sigaction { void (*sa_handler)  () ; /*fonction */
                   sigset_t sa_mask;  /* masque des signaux */
                   int sa_flags; /* options */
```

- **Le comportement que doit avoir un processus lors de la délivrance d'un signal est décrit par la structure sigaction**
  - *sa_handler* :
    - fonction à exécuter, SIG_DFL (traitement par défaut), ou SIG_IGN (ignoré le signal)
  - *sa_mask :* correspond à une liste de signaux qui seront ajoutés à la liste de signaux qui se trouvent bloqués lors de l'exécution du *handler*.
    - *sa_mask U {sig}*:
    - Le signal en cours de délivrance est **automatiquement** masqué par le handler
  - *sa_flags*: différentes options

Signaux : struct sigaction (cont.)

- **Quelques options pour *sa_flags***
  - *SA_NOCLDSTOP* : Le signal SIGCHLD n'est pas envoyé à un processus lorsqu'un de ses fils est stoppé.
  - *SA_RESETHAND* : Rétablir l'action à son comportement par défaut une fois que le gestionnaire a été appelé
  - SA_RESTART: Un appel système interrompu par un signal capté est repris au lieu de renvoyer -1.
  - *SA_NOCLDWAIT*: Si le signal est SIGCHLD, le processus fils qui se termine ne devient pas ZOMBIE
  - etc.
- **La plupart des options ne sont pas dans la norme POSIX**

Signaux : Changement du traitement par défaut

- **int sigaction (int sig, struct sigaction *act, struct sigaction *anc);**
  - **Permet l'installation d'un handler *act* pour le signal *sig***
    - *act* et *anc* pointent vers une structure du type *struct sigaction*
    - La délivrance du signal *sig*, entraînera l'exécution de la fonction pointée par *act->sa_handler*, si non NULL
    - *anc:* si non NULL, pointe vers l'ancienne structure sigaction

Signaux : Exemple changement traitement par défaut (sigaction)

```
#define _XOPEN_SOURCE 700

#include <signal.h>
#include <stdio.h>
#include <stdlib.h>

void sig_hand(int sig){
  printf ("signal reçu %d \n",sig);
}
```

sigaction-ex.c

```
> sigaction-ex
signal reçu 2
fin programme
```

```
int main(int argc, char **argv) {

  sigset_t sig_proc;
  struct sigaction action;

    sigemptyset(&sig_proc);
    action.sa_mask=sig_proc;
    action.sa_flags=0;
    action.sa_handler = sig_hand;

    sigaction(SIGINT, &action,NULL);

    kill (getpid(), SIGINT);
    printf("fin programme \n");

  return EXIT_SUCCESS;
```


Signaux : Exemple – sigaction + sigsuspend

```
void sig_hand(int sig){
  printf ("signal reçu %d \n",sig);
}
int main(int argc, char **argv) {
  sigset_t sig_proc;
  struct sigaction action;
    sigemptyset(&sig);

      /* changer le traitement */
      action.sa_mask=sig_proc;
      action.sa_flags=0;
      action.sa_handler = sig_hand;

 sigaction(SIGINT, &action,NULL);
 sigaction(SIGQUIT, &action,NULL);
```

```
    sigsuspend (&sig_proc);

    return EXIT_SUCCESS;
}
```

Processus peut se bloquer pour toujours !!

Signaux : Exemple – sigaction + sigsuspend

```
void sig_hand(int sig){
  printf ("signal reçu %d \n",sig);
}
int main(int argc, char **argv) {
  sigset_t sig_proc;
  struct sigaction action;
    sigemptyset(&sig);

      /* changer le traitement */
      action.sa_mask=sig_proc;
      action.sa_flags=0;
      action.sa_handler = sig_hand;

sigaction(SIGINT, &action,NULL);
sigaction(SIGQUIT, &action,NULL);
```

```
/* masquer tous les signaux*/
     sigfillset (&sig);
     sigprocmask (SIG_SETMASK, &sig_proc,
NULL);
                ⋮

/* attendre le signal SIGINT et SIGQUIT */
     sigdelset (&sig, SIGINT);
     sigdelset (&sig, SIGQUIT);
     sigsuspend (&sig_proc);

     return EXIT_SUCCESS;
}
```


Attente d'un signal (exemple1)

```
pid_t pid_fils;
void sig_hand(int sig){
  printf ("signal reçu %d \n",sig);
}

int main(int argc, char **argv) {
  sigset_t sig_proc;
  struct sigaction action;
     sigemptyset(&sig_proc);

     /* changer le traitement */
     action.sa_mask=sig_proc;
     action.sa_flags=0;
     action.sa_handler = sig_hand;

     sigaction(SIGUSR1, &action,0);
```

Fils se bloque pour toujours ?

```
if ( (pid_fils= fork ()) == 0) {
    sleep (1);
    printf("fils: après sleep \n");
    pause ();
    printf ("reprise fils \n");
 }
 else{
   kill (pid_fils, SIGUSR1);
   wait(NULL);
    printf ("fin pere \n");
 }
    return EXIT_SUCCESS;
}
```

sigurs1-ex1.C

OUI
> **sigusr1-ex1**
signal reçu 10
fils: après sleep

Attente d'un signal (exemple2)

```
pid_t pid_fils;
void sig_hand(int sig){
  printf ("signal reçu %d \n",sig);
}

int main(int argc, char **argv) {
  sigset_t sig_proc;
  struct sigaction action;
     sigemptyset(&sig_proc);
     /* changer le traitement */
     action.sa_mask=sig_proc;
     action.sa_flags=0;
     action.sa_handler = sig_hand;
     sigaction(SIGUSR1, &action,0);

     if ( (pid_fils= fork ()) == 0) {
          sleep (1);
         printf("fils: après sleep \n");
```

Fils se bloque pour toujours ?

```
  sigaddset (&sig_proc, SIGUSR1);
  sigprocmask (SIG_SETMASK,
            &sig_proc, NULL);
  sigfillset (&sig_proc);
  sigdelset (&sig_proc, SIGUSR1);
  sigsuspend (&sig_proc);
```

```
    printf ("reprise fils \n");
  }
  else{
    kill (pid_fils, SIGUSR1);
    wait(NULL);
     printf ("fin pere \n");
  }
    return EXIT_SUCCESS;
}
```

sigurs1-ex2.C

OUI
> **sigusr1-ex2**
signal reçu 10
fils: après sleep

Attente d'un signal (exemple3)

```
pid_t pid_fils;
void sig_hand(int sig){
  printf ("signal reçu %d \n",sig);
}
int main(int argc, char **argv) {
  sigset_t sig_proc;
  struct sigaction action;

   sigemptyset(&sig_proc);
    action.sa_mask=sig_proc;
    action.sa_flags=0;
    action.sa_handler = sig_hand;
   sigaction(SIGUSR1, &action,0);
```

```
  sigaddset (&sig_proc, SIGUSR1);
  sigprocmask (SIG_SETMASK,
            &sig_proc, NULL);
```

Fils se bloque pour toujours ?

```
if ( (pid_fils= fork ()) == 0) {
    sleep (1);
    printf("fils: après sleep \n");
    sigfillset (&sig_proc);
    sigdelset (&sig_proc, SIGUSR1);
    sigsuspend (&sig_proc);
    printf ("reprise fils \n");
 }
 else{
   kill (pid_fils, SIGUSR1);
   wait(NULL);
    printf ("fin pere \n");
 }
    return EXIT_SUCCESS;
}
```

sigurs1-ex3.C

NON
> **sigusr1-ex3**
fils: après sleep
signal reçu 10
reprise fils
fin pere

Perte de signaux pendants

- **Pas de mémorisation du nombre de signaux pendants**

sig_contUSR1.c

```
int cont; pid_t pid_fils;

void sig_hand(int sig){
 if (sig == SIGUSR1)
   cont++;
 else {
   printf ("nombre SIGUSR1 recu: %d \n", cont);
   exit (0);
 }
}

int main(int argc, char **argv) {
 sigset_t sig_proc; int i;
 struct sigaction action;

  sigemptyset(&sig_proc);
    action.sa_mask=sig_proc;
    action.sa_flags=0;
    action.sa_handler = sig_hand;
    sigaction(SIGUSR1, &action,0);
    sigaction(SIGINT, &action,0);

if ( (pid_fils= fork ()) == 0) {
  while (1)
    pause ();
}
else{
  for (i=0; i<20; i++)
      kill (pid_fils, SIGUSR1);
  kill (pid_fils, SIGINT);
  wait (NULL);
 return EXIT_SUCCESS;
 }
}
```

```
>sig_contUSR1
  nombre SIGUSR1 reçu:4
```


Signaux : SIGCHLD

- **Signal envoyé automatiquement à un processus lorsque l'un de ses fils se termine ou lorsque l'un de ses fils passe à l'état stoppé (réception du signal SIGSTOP ou SIGTSTP).**
- **Le comportement par défaut est d'ignorer le signal**
- **En captant ce signal, un processus peut prendre en compte le "moment" où la terminaison de son fils s'est produite.**
- **Elimination du fils zombie**
  - wait() , waitpid ()

Signaux : SIGCHLD - Exemple

```
void sig_hand(int sig){
      printf ("signal reçu %d \n",sig);
 if (sig == SIGCHLD)
    wait (NULL)
}

    int main(int argc, char **argv) {
     sigset_t sig_proc;
     struct sigaction action;

        sigemptyset(&sig_proc);
```

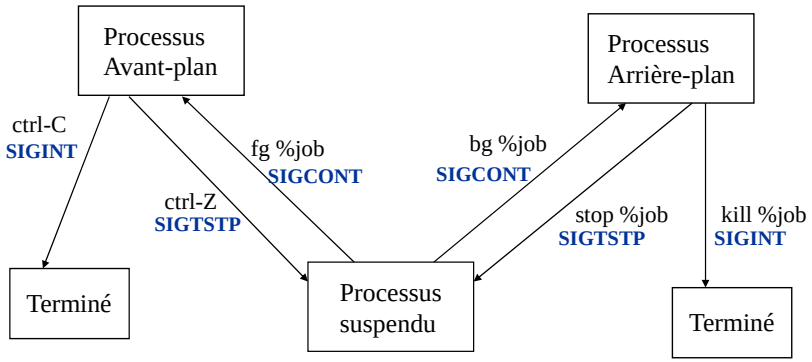
```
/* changer le traitement */
     action.sa_mask=sig_proc;
     action.sa_flags=0;
     action.sa_handler = sig_hand;
  sigaction(SIGCHLD, &action,NULL);


if (fork() != 0)
      sleep (1);

   return EXIT_SUCCESS;
}
```


Signaux : SIGSTOP/SIGTSTP, SIGCONT et SIGCHLD

- **Processus s'arrête (état bloqué) en recevant un signal SIGSTOP ou SIGTSTP**
- **Processus père est prévenu par le signal SIGCHLD de l'arrêt d'un de ses fils**
  - Comportement par défaut : ignorance du signal
  - Relancer le processus fils en lui envoyant le signal SIGCONT
  - Observation :
    - en fait le processus père reçoit un SIGCHLD a chaque fois qu'un de ses fils change de status (exemple, processus fils redémarre en recevant un SIGCONT)

Gestion de Jobs

- **Gestion par des signaux**



> **jobs** /* commande pour obtenir la liste des jobs */

Signaux : SIGSTOP/SIGCONT, SIGCHLD Exemple

```
#define _XOPEN_SOURCE 700
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>


pid_t pid_fils;
void sig_hand(int sig){
  printf ("signal reçu %d \n",sig);
   kill (pid_fils, SIGCONT);
}

int main(int argc, char **argv) {
   sigset_t sig_proc;
   struct sigaction action;
   sigemptyset(&sig_proc);

   /* changer le traitement */
   action.sa_mask=sig_proc;
   action.sa_flags=0;
   action.sa_handler = sig_hand;
   sigaction(SIGCHLD, &action,0);

    if ( (pid_fils= fork ()) == 0) {
        kill (getpid(), SIGSTOP);
        printf ("reprise fils \n");
     }
   else{
      wait (NULL);
      printf ("fin pere \n");
   }
    return EXIT_SUCCESS;
}
```

sig_STOP.c

```
> sig_stop
  signal reçu 20
  reprise fils
  fin père
```


Signaux : Utilisation des temporisateurs (*alarm et setitimer*)

- **But : Interrompre le processus au terme d'un délai**
  - Processus arme un temporisateur (timer). Lorsque le délai fixé arrive à son terme, le processus reçoit un signal.
  - Un seul temporisateur par processus
  - Utilisation des fonctions *alarm* ou *setitimer*
    - ***alarm*** : temps réel mais la résolution est en secondes.
      - Signal reçu : SIGALRM
    - ***setitimer*** : permet de définir de temporisateurs types avec une résolution plus fine que la seconde.
      - Signal reçu : SIGALRM, SIGTVALRM ou SIGPROF
  - Terminaison du processus est le traitement par défaut du signal reçu

Signaux : *alarm( )* - SIGALRM

- **alarm(int sec);**
  - Durée exprimée en secondes
    - Temps-réel (wall-clock time) dont la résolution est à la seconde
  - Un SIGALRM est généré à son terme
  - Un seul temporisateur par processus
    - Une nouvelle demande annule la précédente.
    - Un appel avec la valeur 0 annule la demande en cours.

Signaux *alarm( )* - SIGALRM (Exemple)

```
void sig_hand(int sig){
   printf ("signal reçu %d \n",sig);
   alarm (1);
 }

 int main(int argc, char **argv) {

  sigset_t sig_proc;
  struct sigaction action;
  sigemptyset(&sig_proc);
```

```
action.sa_mask=sig_proc;
action.sa_flags=0;
action.sa_handler = sig_hand;
sigaction(SIGALRM, &action,0);
 alarm (1);
 while (1)
     pause ();
 return EXIT_SUCCESS;
}
```

sig_ALRM.c

```
>sig_ALRM
  signal reçu 20
  signal reçu 20
  signal reçu 20
…..
```


Signaux: *setitimer ( )* SIGALRM, SIGVTALRM, SIGPROF

- **Primitive *setitimer* permet trois type d'alarmes**

  #include <sys/time.h>

  int setitimer (int type, struct itimerval * new, struct itimerval *old);

TYPE	TEMPORISATION	SIGNAL
ITIMER_REAL	Temps réel	SIGALRM
ITIMER_VIRTUAL	Temps en mode utilisateur	SIGVTALRM
ITIMER_PROF	Temps CPU total	SIGPROF

Fonctions non POSIX : raise

- **int raise (int sig);**
  - Envoie le signal *sig* donné au processus courant.
  - Equivaut à *kill (getpid(), sig)*

```
void sig_hand(int sig){
  printf ("signal reçu %d \n",sig);
}

int main(int argc, char **argv) {
  sigset_t sig_proc;
  struct sigaction action;

     sigemptyset(&sig_proc);
```

```
/* changer le traitement */
     action.sa_mask=sig_proc;
     action.sa_flags=0;
     action.sa_handler = sig_hand;
  sigaction(SIGINT, &action,NULL);

  raise (SIGINT);
  return EXIT_SUCESSS
```


Fonctions non POSIX: *signal*

- **typedef void (*sighandler_t)(int);**

  sighandler_t signal(int *sig*, sighandler_t *handler*);
  - installe un nouveau gestionnaire pour le signal numéro *sig*. Le gestionnaire de signal est *handler* qui peut être soit une fonction , soit une des constantes **SIG_IGN** ou **SIG_DFL**.
  - **VALEUR RENVOYÉE**
    - valeur précédente du gestionnaire de signaux, ou **SIG_ERR** en cas d'erreur.

Fonctions non POSIX : *signal*

- **Limitations:**
  - Impossible d'avoir une masque de signaux pendant l'exécution du *handler*
  - Le traitement par défaut est réinstallé après l'exécution du handler

```
void sig_hand(int sig){
  printf ("singal reçu %d \n",sig);
}

int main(int argc, char **argv) {
   signal(SIGINT, &sig_hand);
   kill (getpid(), SIGINT);
   return EXIT_SUCCESS;
}
```


Limites des Signaux

- **Quelques limitations de signaux:**
  - Aucune mémorisation du nombre de signaux reçus
  - Aucune mémorisation de la date de réception d'un signal
    - les signaux seront traités par ordre de numéro.
  - Aucun moyen de connaître le PID du processus émetteur du signal.