Outils de communication entre processus
Les signaux

Signaux : définition

- Le signal est une interruption logicielle.
  - Il informe les processus de l'occurrence d'événements asynchrones.
  - Il ne transporte pas d'informations.
  - Le processus qui le prend en compte exécute un traitement.

- 64 signaux identifiés par un numéro et un nom (SIGX)
  - 1 à 31 : signaux classiques
  - 32 à 63 : signaux temps réel

Signaux Classiques

- L'événement associé à un signal
  - extérieur au processus (frappe caractère, terminaison d'un autre processus)
  - intérieur au processus correspondant à une erreur (erreur arithmétique ou violation mémoire) : levée d'une trappe.

Liste des signaux selon le standard POSIX

- signaux relatifs à la fin de processus :
  - SIGCHLD (17) : mort du fils
  - SIGKILL (9) : signal de terminaison
- signaux relatifs à des erreurs
  - SIGILL (4) : instruction illégale
  - SIGFPE (8) : erreur arithmétique
  - SIGSEGV (11) : violation mémoire
  - SIGPIPE (13) : écriture dans un tube sans lecteur
- signaux relatifs aux temporisations
  - SIGALRM (14): fin de temporisation (fonction alarm)

Liste des signaux selon le standard POSIX

- signaux relatifs aux interactions avec le terminal (extinction, frappe touche DELETE et BREAK)
  - SIGHUP, SIGINT (crtl C), SIGQUIT (crtl \)

- signaux relatifs à la mise au point de programmes

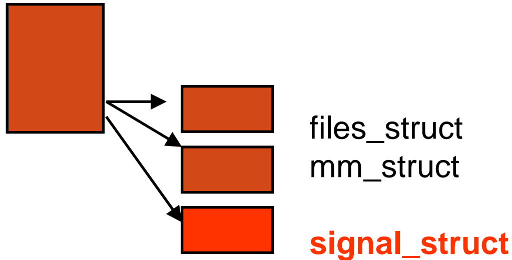
- Deux signaux disponibles pour les utilisateurs (SIGUSR1, SIGUSR2)

Signaux : aspects du traitement

- Comment le noyau envoie-t-il un signal à un processus ?
- Comment le noyau traite-t-il un signal ?
- Comment un processus contrôle-t-il ses réactions vis-à-vis des signaux ?

Signaux : structures de données du processus

PCB



Typedef struct { unsigned long sig[2]; } sigset_t;

Signaux classiques 11100000001111111111111111111000
Signaux temps réel 01010011010010101001010101000101

```
Struct  signal_struct { ……
                struct k_sigaction action [64];
                …….}
```

TASK_STRUCT

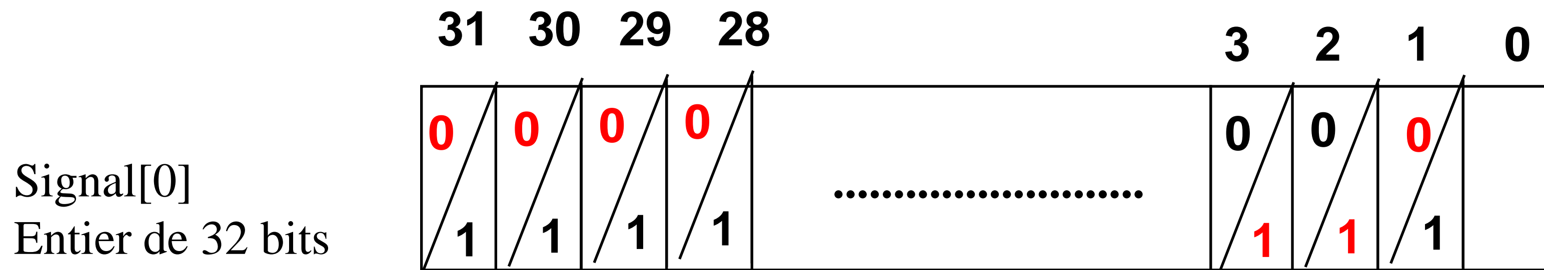
volatile long state; *- - état du processus*

sigset_t signal; -- signaux reçus
sigset_t blocked; -- signaux bloqués (« masqués »)

struct task_struct *next_task, *prev_task; *-- chainage PCB*
struct task_struct *next_run, *prev_run; *-- chainage PCB Prêt*
int pid; *-- pid du processus*
struct task_struct *p_opptr, *p_pptr, *p_cptr,; *-- pointeurs PCB père originel, père actuel, fils*
long need_resched; *-- ordonnancement requis ou pas*
long utime, stime, cutime, cstime;
-- temps en mode user, noyau, temps des fils en mode user, noyau
unsigned long policy; *-- politique ordonnancement SCHED_RR, SCHED-FIFO, SCHED_OTHER*
struct thread_struct tss; *-- valeurs des registres du processeur*
struct mm_struct *mm; *-- contexte mémoire*
*struct files_struct *files ; -- table fichiers ouverts*
*struct signal_struct *sig ; -- table de gestion des signaux ; actions à réaliser lors de la prise en compte des signaux par le processus*

1. Comment le noyau envoie-t-il un signal à un processus ?

- Positionnement d'un bit à 1 dans le champ signal de la table des processus, correspondant au type de signal reçu.
  - signal pendant (reçu)
  - vecteur de bits : pas de mémorisation du nombre de signaux d'un type reçu.



2. Comment le noyau traite-t-il un signal ?

- Le noyau traite la réception des signaux quand le processus quitte le mode noyau pour retourner au mode utilisateur
  - signal délivré
  - le bit correspondant est remis à 0

Utilisateur

Noyau

☞ **Un processus ne s'exécute jamais en mode utilisateur avant de traiter les signaux**

Ordonnancement
Election

Examen des signaux reçus par le processus élu
Traitement
(do_signal())

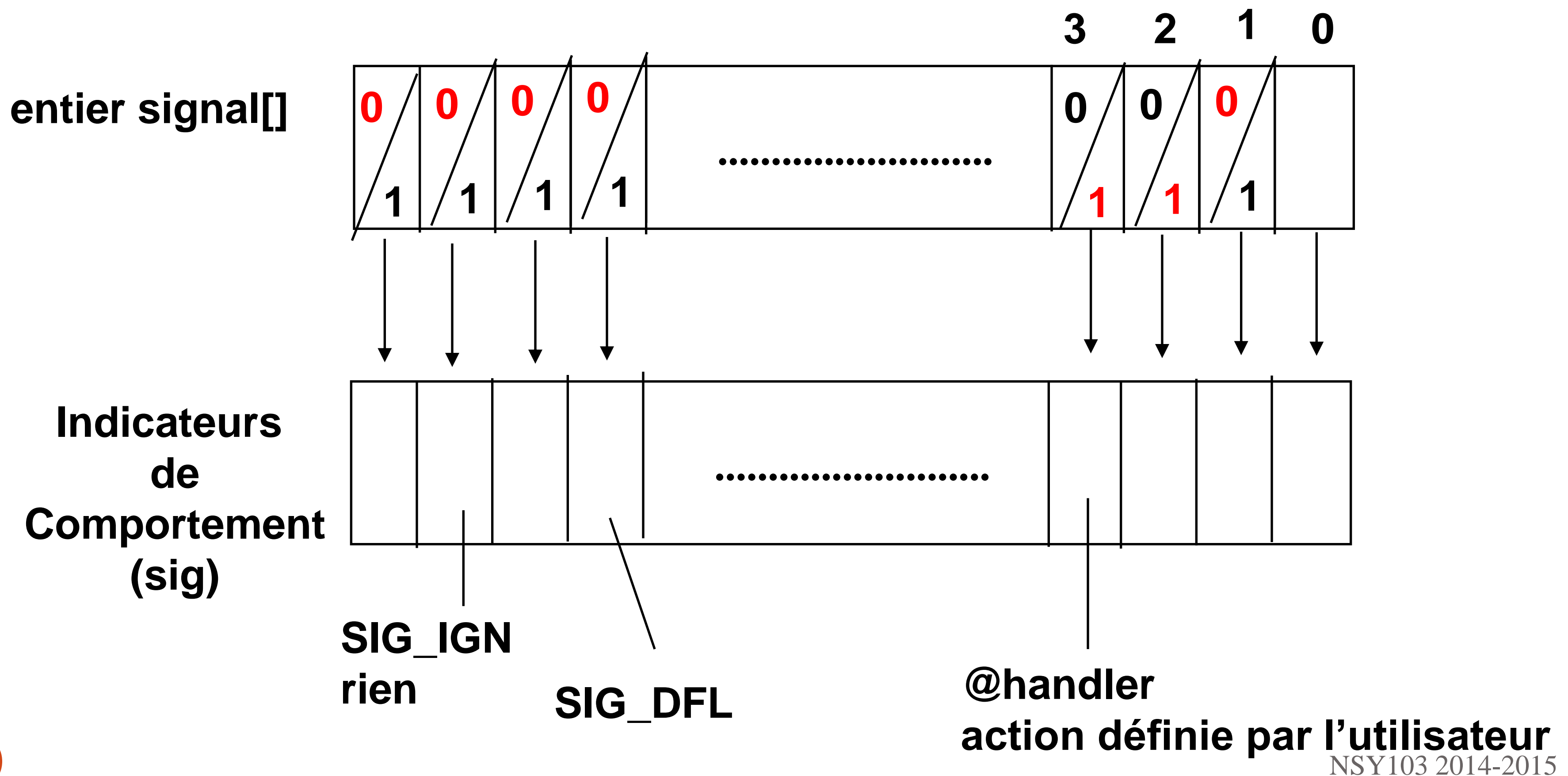
3. Comment un processus contrôle-t-il ses réactions vis-à-vis des signaux ?

- Traitement des signaux

  - A tout signal est associé un traitement par défaut (ignorer, terminer le processus avec ou sans core, stopper le processus);
  - Tout processus peut installer pour chaque type de signal (hormis SIGKILL), un nouveau handler
    - handler SIG_IGN : pour ignorer le signal (sauf mort du fils pour Linux)
    - handler fonction utilisateur pour capter le signal
      - fonction signal et sigaction

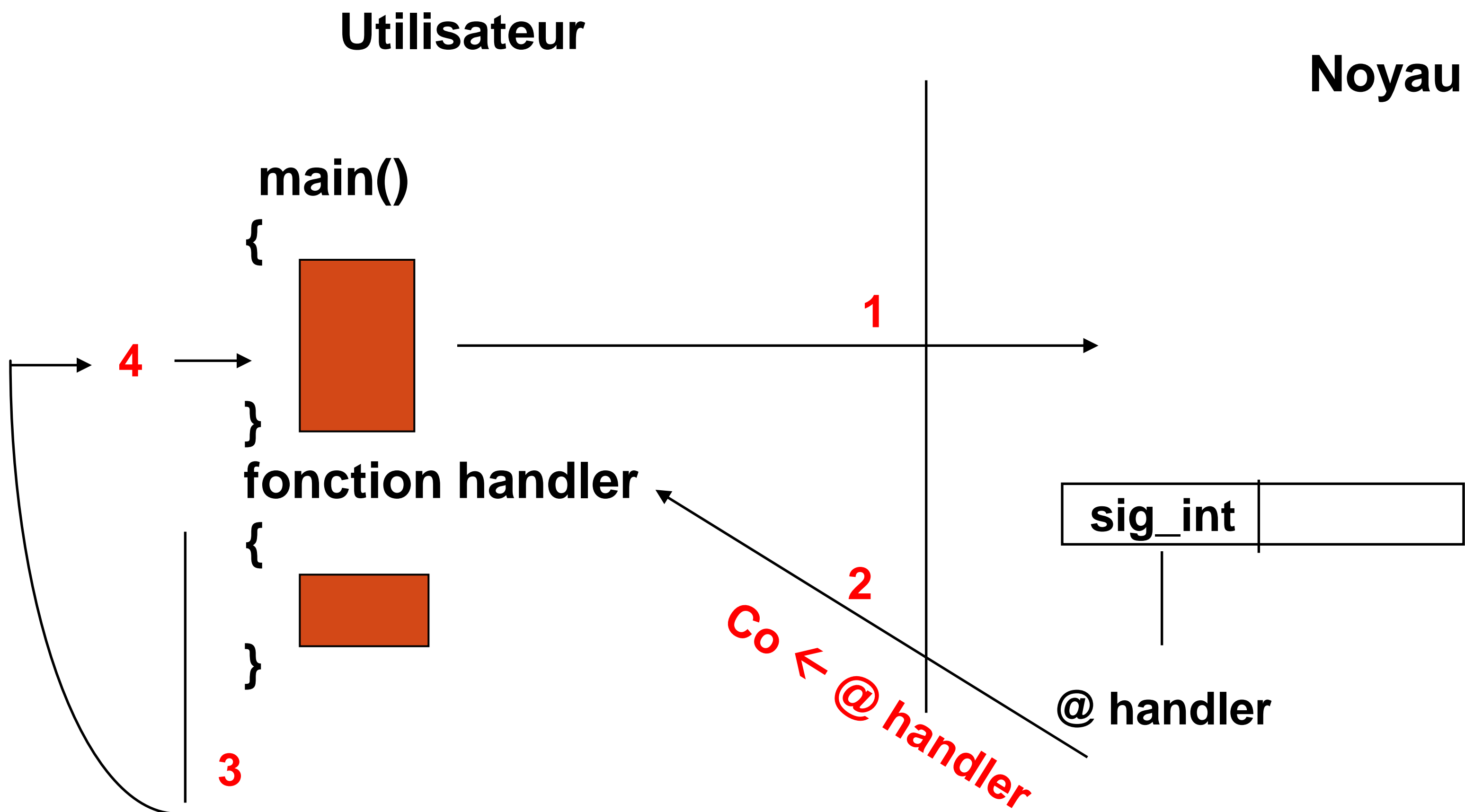
3. Comment un processus contrôle-t-il ses réactions vis-à-vis des signaux ? (fonction du noyau do_signal())

- Traitement des signaux



3. Exécution d'un handler de signal défini par l'utilisateur

- Traitement des signaux



Les fonctions liées aux signaux

- Envoyer un signal à un processus
  - int kill (pid_t pid, int sig) kill (12563, SIGKILL)
  - kill -n°signal pid kill -9 12563

- Associer un handler à un signal
  - signal(int sig, fonction) signal(SIGINT, p_hand)
  - sigaction(int sig, struct sigaction action, NULL)

- Armer une temporisation
  - int alarm (int seconds) alarm(10)
  - au bout de seconds unités de temps, le signal SIGALRM est envoyé au processus

- Attendre un signal
  - int pause();

Primitive sigaction()

Associer un handler à un signal (POSIX)

```
int sigaction (int sig, const struct sigaction *p_action,
                        struct sigaction *p_action_anc)

struct sigaction {void (*sa_handler)(); /* SIG_DFL ou SIG_IGN
                                           ou pointeur sur handler */
                  sigset_t sa_mask; /* masque sur signaux */
                  int sa_flags; }  /* options… à mettre à NULL */
```

```
struct sigaction action;
action.sa_handler = p_hand;
sigaction (SIG_INT, &action, NULL);
```

-

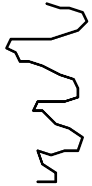
Signaux : synthèse

Utilisateur | **Noyau**

Processus 1

{

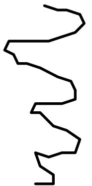
signal(sig1, handler)



}

handler()

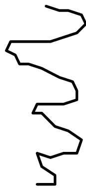
{



}

Processus 2

{



kill (proc1, sig1)

}

processus1

sig1

handler

Signaux : synthèse

Utilisateur | **Noyau**

Processus 1

```
{
signal(sig1, handler)
}

handler()
{
}
```

Processus 2

```
{
kill (proc1, sig1)
}
```

processus1

sig1

Signal pendant

handler

Signaux : synthèse

Utilisateur

Noyau

Processus 1

{

signal(sig1, handler)

}

handler()

{

}

Processus 2

{

kill (proc1, sig1)

It horloge

}

processus1

sig1

0

Signal délivré

handler

ordonnancement
élection processus 1
examen, traitement signaux

Signaux et interruptions

- Signaux
- Le processus P2 envoie un signal au processus P1 (signal pendant chez P1)
- Plus tard, le processus P1 est élu. Il quitte le mode noyau. Il exécute le handler du signal en mode utilisateur (signal délivré à P1).

- Interruptions
- Le dispositif matériel X envoie une interruption lors de l'exécution du processus P1.
- Immédiatement, le processus P1 est dérouté en mode noyau pour exécuter le handler « routine » de l'interruption.

/* programme où un processus crée un fichier sur lequel il travaille. En cas de réception du signal SIGINT, le processus ferme le fichier sur lequel il travaille avant de se terminer */

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

int desc;

main()
{
int i;
 extern onintr();    /* Handler */

desc = open("toto", O_RDWR, 0);
signal (SIGINT, onintr);
i = 0;
while (i < 5)
{            write (desc, "abcdefghijgl", 12);
             i = i + 1;  }
printf("fin process normal\n");
close (desc);
```

```
onintr ()
{
            printf("Handler onintr\n");
            close(desc);
            exit ();
}
```

/* programme où un processus crée un fils puis se met en attente de la fin de son fils. Le fils exécute un code qui boucle. Au bout de 10 secondes, le fils n'étant pas achevé, le père tue son fils */

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <signal.h>
 pid_t pid;

main()
{
extern onalarm();    /* Handler */

pid = fork();
if (pid == -1)
                printf ("erreur creation de processus");
else
                if (pid == 0)
                {
                 printf ("valeur du fork, %d", pid);
                 printf (" je suis le fils, mon pid est %d\n", getpid());
                 for(;;)
                                printf("je boucle !!!! \n");
                 exit();  }
                else  {
                 printf ("valeur du fork, %d", pid);
                 signal (SIGALRM, onalarm);
                 alarm(5);
                 wait ();  }
}
```

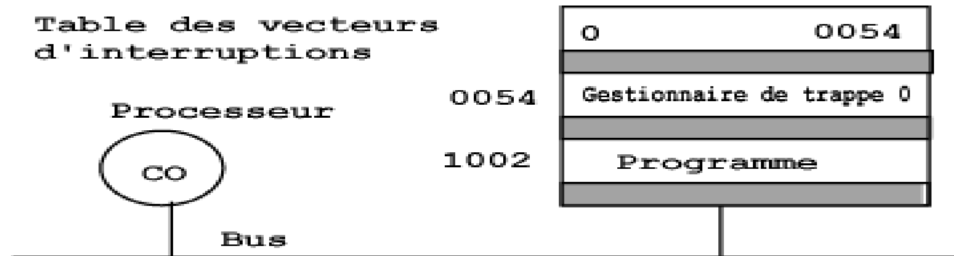
```
onalarm ()
{
                printf("Handler onalarm\n");
                kill (pid, SIGKILL);
                exit ();
}
```

Les signaux pour gérer les exceptions

signaux relatifs à des erreurs

- SIGILL (4) : instruction illégale
- SIGFPE (8) : erreur arithmétique
- SIGSEGV (11) : violation mémoire
- SIGPIPE (13) : écriture dans un tube sans lecteur

NOTIONS DE BASE : traitement d'une exception



Mode utilsateur

```
main()
{
int i, j, fd;
char tab[10];

i = 3;

fd = open ("fichier",
                O_RDWR);

if (fd == -1)
   {
    perror("Problème
   ouverture fichier\n");
    exit(1);
   }
else
    {
    read (fd, tab, 10);
    printf("caractères
          lus: %s \n", tab);
    i = i /j;
    close(fd);
    }
}
```

Co <- 0054
PSW <- Superviseur

Mode superviseur

Gestionnaire d'exception 0

0054 Sauvegarde des registres généraux

do_divide_error()
Fonction de traitement
de l'exception
Envoi d'un signal
au processus fautif

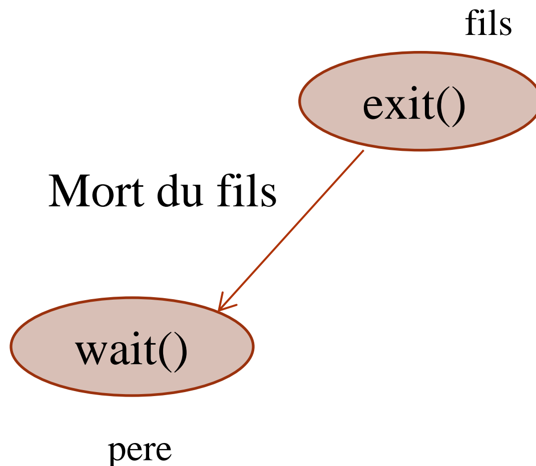
Restitution des registres généraux
ret_from_exception
Exécution du traitement
associé au signal émis
Par défaut arrêt de
l'exécution du processus

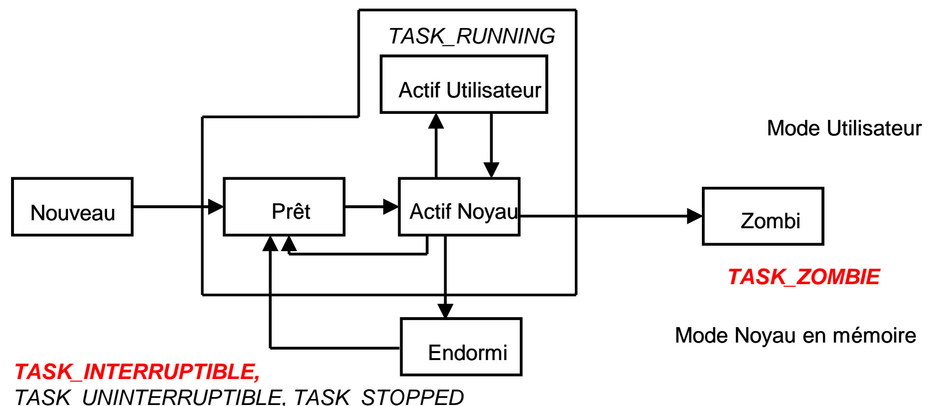
Signal mort du fils
Synchronisation père et fils

Synchronisation père –fils : par défaut

- Lorsque le fils se termine, il envoie un signal « Mort du fils » à son père. Le père récupère cette mort par la primitive wait().

- Réception du Signal « mort du fils » :
  - Par défaut, réveille le processus s'il est en endormi en mode interruptible

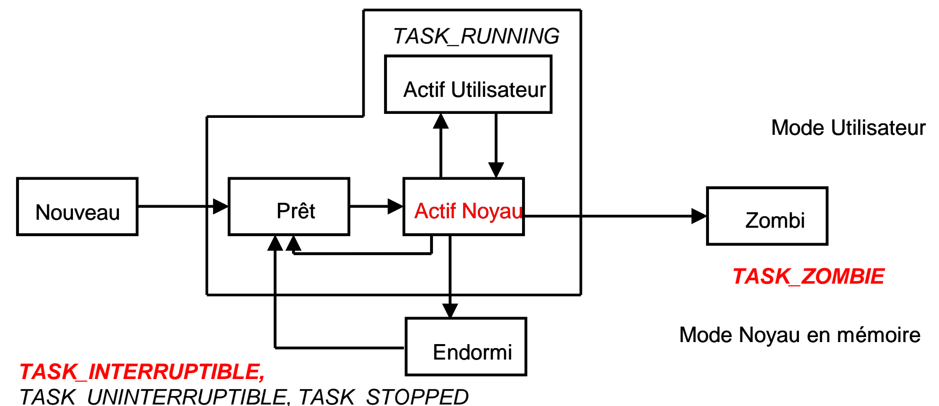




Synchronisation père –fils : par défaut

- Réception du Signal « mort du fils » :
  - Par défaut, réveille le processus s'il est en endormi en mode interruptible

```
algorithme wait
entrée: adresse d'une variable pour ranger l'état du
        processus qui fait l'exit.
sortie: ID et code d'exit du fils
{
    if (le processus en attente n'a pas de processus fils)
        return (erreur);
    for(;;) /* boucle jusqu'à un return */
    {
        if (le processus en attente a des fils zombies)
        {
            accéder à un fils zombie quelconque;
            rajouter l'utilisation de l'UC par le fils au parent;
            libérer l'élément de la table des processus
                                      correspondant au fils;
            return (ID et code d'exit du fils);
        }
        if (le processus n'a pas de fils)   % au moins un %
            return (erreur);
        s'endormir en une priorité interruptible en attente de
                          l'événement: exit du processus fils;
    }
}
```



Par défaut, le processus démantèle un processus fils zombi

Synchronisation père –fils : traitement user

- Réception du Signal « mort du fils » :
  - Si un traitement utilisateur est associé au signal, le noyau déclenche l'exécution de ce traitement

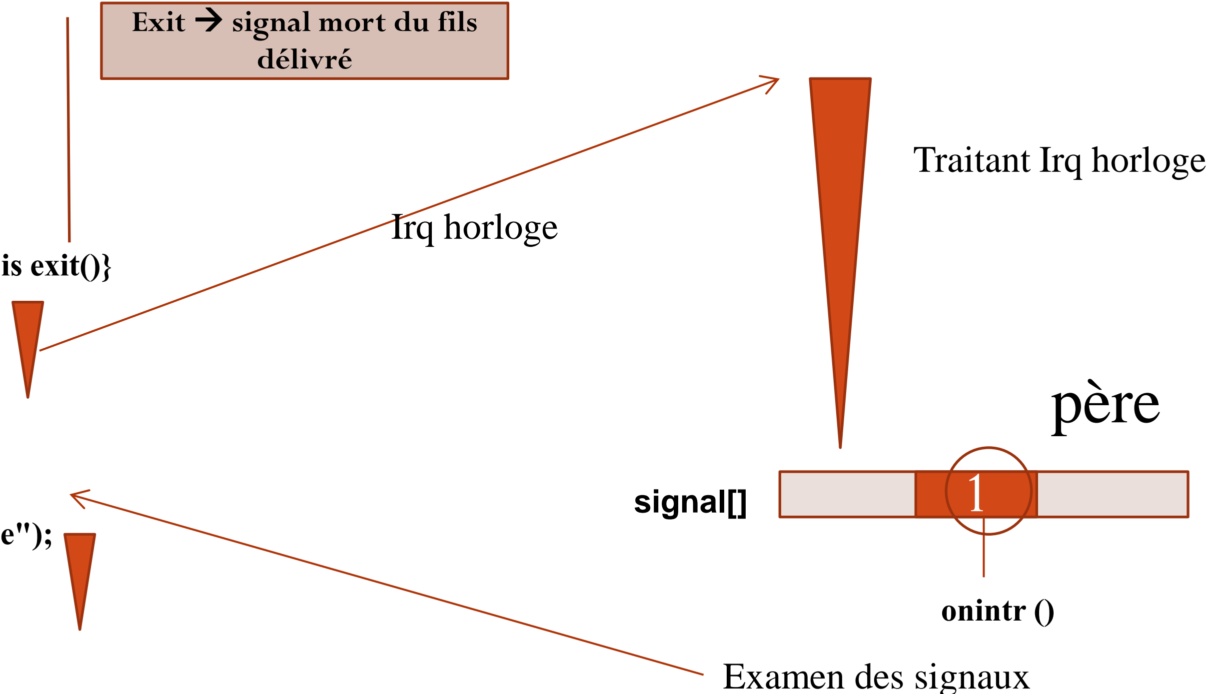
```
main()
{
int i; pid_t ret;
 extern onintr();    /* Handler */

signal (SIGCHLD, onintr);
ret = fork();
if (ret==0) { le fils fait des choses puis exit()}
else
{    le père execute des choses    }
}

onintr ()

{ printf("mort du fils pris en compte");

wait() }
```



Synchronisation père –fils : traitement « ignorer »

- Réception du Signal « mort du fils » :
  - Si un traitement « ignore » est associé au signal, le noyau recherche tous les fils zombies du processus

```
main()
{
int i; pid_t ret;

signal (SIGCHLD, SIG_IGN);
Socke= socket(… TCP);
for(;;){
socks = accept(socke, …..)
ret = fork();
if (ret==0) { close (socke);
 le fils traite la requete sur socks puis exit()}
else
   close (socks);
}
}
```

Exit → signal mort du fils délivré

Irq horloge

Traitant Irq horloge

père

signal[]

1

SIG_IGN

Examen des signaux
Détruire tous les fils zombies existants