Système d'Exploitation Unix
TP
Les appels système : les signaux

Tous les programmes devront être développés avec passage de leurs éventuels paramètres à la fonction `main (int argc, char *argv [])`. Les valeurs de retour des appels aux primitives devront être testées et les messages d'erreurs affichés avec `perror`. Les messages d'erreurs à destination de l'utilisateur se feront sur le fichier standard des erreurs `stderr`.

Question 1 Emisssion d'un signal `kill ()`

La primitive `int kill(pid_t pid, int sig)` permet l'envoi d'un signal à un processus ou à un groupe de processus. La valeur du paramètre `pid` définit l'ensemble des processus destinataires du signal `sig`.

a) Ecrire un programme qui envoie différents signaux au(x) processus selon les valeurs possibles des deux paramètres `pid` et `sig`.

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

int main(int argc, char *argv[]){
     int nb_sig;
     int sig;
     int pid;

     if (argc != 4) {
          fprintf (stderr, "usage %s: signal pid nb_signal\n", argv[0]);
          exit (EXIT_FAILURE);
     }
     printf ("\n>>> DEBUT <<<\n");
     sig=atoi(argv[1]);
     pid=atoi(argv[2]);
     for (nb_sig=atoi(argv[3]); nb_sig>0; nb_sig--){
          if (kill (pid,sig)<0){
               perror ("kill ");
               fprintf (stderr, "ERREUR: %d signaux %d restant pour %d\n",
                                                  nb_sig, sig, pid);
               exit (EXIT_FAILURE);
          }// if
          else printf("Signal %d envoyé à %d\n", sig, pid);
     }// for
     printf ("\n>>> FIN: %s signaux %d envoyés à %d <<<\n",
                                                  argv[3], sig, pid);

}// main
```


b) Que devient le processus destinataire ? Expliquer !
La terminaison du processus est le comportement par défaut à la réception du signal `SIGUSR1` ou `SIGUSR2`

Une valeur nulle (0) du paramètre `sig` correspond au test d'existence du processus mentionné.

c) Vérifier l'affirmation précédente.

d) Dans le cas précédent, y a-t-il émission de signal ? Non

Que devient le processus destinataire ? *il continu son exécution*

La fonction `int raise(int sig)` fait partie de l'interface standard du langage C et permet l'envoi du signal `sig` au processus courant.

e) Ecrire un programme qui s'envoie plusieurs signaux différents.

f) Décrire le résultat de la précédente exécution ? Expliquer !

Question 2 Attente d'un signal `pause ()`

La primitive `int pause(void)` permet de se mettre en attente de l'arrivée de signaux.

a) Ecrire un programme dont le processus correspondant se met en attente de l'arrivée de signaux. Emettre à ce processus les signaux `SIGUSR1` et `SIGSTOP`.

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

b) Que devient le processus après la délivrance du signal `SIGUSR1` ?

La terminaison du processus est le comportement par défaut à la réception du signal `SIGUSR1` ou `SIGUSR2`

c) Que devient le processus après la délivrance du signal `SIGSTOP` ?

Il passe à l'état « suspendu » (T)

d) Que se passe-t-il s'il reçoit à la suite le message `SIGCONT` ?

Il reprend son exécution et passe à l'état S ou R

Question 3 Le gestionnaire de signal ANSI C `signal ()`

La primitive `void (*signal(int signo, void (*func)(int)))(int);` permet d'installer un nouveau gestionnaire `func` pour le signal `signo`. Le gestionnaire peut être soit une fonction spécifique de l'utilisateur, soit l'une des constantes `SIG_IGN` ou `SIG_DFL`.

a) Ecrire un programme qui ignore la réception des signaux USR1 et USR2.

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

main(){
     signal (SIGUSR1, SIG_IGN);
     signal (SIGUSR2, SIG_IGN);
     pause ();
}
```

b) Vérifier que le processus ignore bien les signaux utilisateur.

c) Quel aurait été le résultat si les signaux n'étaient pas ignorés ?

La terminaison du processus est le comportement par défaut à la réception du signal SIGUSR1 ou SIGUSR2

d) Réécrire le programme précédent de façon à installer un gestionnaire spécifique pour les deux signaux USR1 et USR2. Sur réception de chacun de ces deux signaux, le gestionnaire doit afficher la nature (USR1 ou USR2) du signal reçu ainsi que le nombre total d'occurrences reçues de ce même signal.

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

int nb_sigusr1=0;
int nb_sigusr2=0;

void gest_sigusr1 (int sig) {
     nb_sigusr1++;
     printf ("recu %d signal %d (SIG_USR1)\n", nb_sigusr1, sig);
} //gest_sigusr1

void gest_sigusr2 (int sig) {
     nb_sigusr2++;
     printf ("recu %d signal %d (SIG_USR2)\n", nb_sigusr2, sig);
} //gest_sigusr2


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

     signal (SIGUSR1, gest_sigusr1);
     signal (SIGUSR2, gest_sigusr2);
     for (;;)
          pause ();

}// main
```


La boucle for est ici nécessaire car la fin de l'exécution du gestionnaire d'un signal capté provoque la fin d'exécution de la primitive pause.

e) Ecrire un programme qui envoie un grand nombre de signaux au programme précédent. Quel résultat obtenez-vous lorsque vous émettez plusieurs occurrences des signaux précédents ? Expliquer !

On constate que tous les signaux envoyés ne sont pas traités par leur destinataire. Ce n'est pas le nombre de signaux qui est en cause mais leur fréquence d'envoi qui est trop élevée. Certains signaux sont perdus chez le destinataire car il ne peut y avoir qu'une seule instance de signal pendant.

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

int main(int argc, char *argv[]){
int nb_sig;
int sig;
int pid;

if (argc != 4) {
     fprintf (stderr, "usage %s: signal pid nb_signal\n", argv[0]);
     exit (EXIT_FAILURE);
}
printf ("\n>>> DEBUT <<<\n");
sig=atoi(argv[1]);
pid=atoi(argv[2]);
for (nb_sig=atoi(argv[3]); nb_sig>0; nb_sig--){

     if (kill (pid,sig)<0){
          perror ("kill ");
          fprintf (stderr, "ERREUR: %d signaux %d restant pour %d\n",
                                                  nb_sig, sig, pid);
          exit (EXIT_FAILURE);
     }// if
     else printf("Signal %d envoyé à %d\n", sig, pid);
}// for
printf ("\n>>> FIN: %s signaux %d envoyés à %d <<<\n",
                                          argv[3], sig, pid);
}// main
```

Question 4 Le gestionnaire de signal POSIX `sigaction ()`

La primitive `int sigaction(int signo, const struct sigaction *act, struct sigaction *oldact);` permet de modifier l'action effectuée par un processus lors de la réception d'un signal spécifique

a) Ecrire un programme de façon à installer un gestionnaire spécifique pour les deux signaux USR1 et USR2. Sur réception de chacun de ces deux signaux, le gestionnaire doit afficher la nature (USR1 ou USR2) du signal reçu ainsi que le nombre total d'occurrences reçues de ce même signal.

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

int nb_sigusr1=0;
int nb_sigusr2=0;
struct sigaction action;

void gest_sigusr1 (int sig) {
    nb_sigusr1++;
    printf ("recu %d signal %d (SIG_USR1)\n", nb_sigusr1, sig);
} //gest_sigusr1

void gest_sigusr2 (int sig) {
    nb_sigusr2++;
    printf ("recu %d signal %d (SIG_USR2)\n", nb_sigusr2, sig);
} //gest_sigusr2

// ====================================
    int main(int argc, char *argv[]){
// ====================================
    // Installation du gestionnaire pour SIG_USR1
    action.sa_handler=gest_sigusr1;
    sigemptyset(&action.sa_mask);
    if (sigaction (SIGUSR1, &action, NULL)<0){perror("!!!pb signal");
        exit (EXIT_FAILURE);}

    // Installation du gestionnaire pour SIG_USR2
    action.sa_handler=gest_sigusr2;
    sigemptyset(&action.sa_mask);
    if (sigaction (SIGUSR2, &action, NULL)<0) {perror("!!!pb signal");
    exit (EXIT_FAILURE);}
    for (;;)
        pause ();
}// main
```


b) Envoyer un grand nombre de signaux au programme au programme précédent. Quel résultat obtenez-vous lorsque vous émettez plusieurs occurrences des signaux précédents ? Expliquer ! Comment y remédier ?

Lorsque un grand nombre de signaux est envoyé avec une fréquence élevée des signaux sont perdus: nombre signaux envoyés > nombre signaux reçus. Ceci est dû au fait qu'il ne peut y avoir qu'une seule instance d'un type de signal delivré, pendant ou bloqué à la fois. Un premier élément de solution est de procéder comme en c)

c) Ecrire un programme qui crée un processus fils. Le père itère un grand nombre de fois :

- Envoyer le signal SIGUSR1 à mon fils
- Attendre que mon fils m'envoie le signal SIGUSR1

Le fils itère le même nombre de fois que son père:

- Attendre que mon père m'envoie le signal SIGUSR1
- Envoyer le signal SIGUSR1 à mon père

En fin d'exécution, le père affiche le nombre de SIGUSR1 envoyé et le fils le nombre de SIGUSR1 reçu. Que se passe-t-il ? Expliquer et proposer une solution

Le programme se bloque dès qu'un signal est reçu alors qu'on ne l'attend pas (`pause`). Il faut utiliser `sigsuspend` qui modifie le masque et suspend le processus de manière atomique

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

#define MAX_P 10

int P[MAX_P+1];
int nb_processus;
int nb_iteration;
int destinataire;
int nb_signaux_recus;
int mon_numero;
int indent;
int i;

sigset_t signaux;
struct sigaction action;
```

```
void traiter_signal(int num_signal){
    int expediteur;

    if (mon_numero==nb_processus)
        expediteur=1;
    else expediteur=mon_numero+1;

    for (indent=mon_numero-1; indent>0; indent--) printf ("\t\t\t");
    printf("P%d: Recu signal de P%d\n", mon_numero, expediteur);
    nb_signaux_recus++;

    return;
}

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

    if (argc != 3) {
         fprintf (stderr, "usage %s: nb_processus nb_envoi_signal\n",
                                                              argv[0]);
         exit (EXIT_FAILURE);
    }

    nb_processus=atoi(argv[1]);
    if (nb_processus < 1) {
         fprintf (stderr, "usage %s: nb_processus doit etre 1\n",argv[0]);
         exit (EXIT_FAILURE);
    }
    nb_iteration=atoi(argv[2]);

 // Installation du gestionnaire pour SIG_USR1
    action.sa_handler=traiter_signal;
    sigemptyset(&action.sa_mask);
    if (sigaction (SIGUSR1, &action, NULL)<0)
              {perror("!!!pb signal"); exit (EXIT_FAILURE);}

    sigemptyset (&signaux);          // sigaux = ensemble vide
    sigaddset (&signaux, SIGUSR1);  // Ajout de SIGUSR1 dans signaux
    sigprocmask (SIG_BLOCK, &signaux, NULL); // Blocage de SIGUSR1

    sigemptyset (&signaux); // signaux = ensemble vide pour sigsuspend

    P[1]=getpid();

    for (i=2; i<=nb_processus; i++){
        P[i]=fork();
```



```
    if (P[i]==0){ /* FILS i */
          nb_signaux_recus=0;
        mon_numero=i;
        destinataire=(i-1);

        for (i=1; i<=nb_iteration; i++){
            for (indent=mon_numero-1; indent>0; indent--)
                                                  printf ("\t\t\t");
            printf("P%d: Attente du signal\n",mon_numero);

          // Remplacement du masque courant par l'ensemble "signaux"
          // et attente d'un signal
             sigemptyset (&signaux);       // sigaux = ensemble vide
             sigaddset (&signaux, SIGUSR1);// Ajout de SIGUSR1
          // Déblocage de SIGUSR1
             sigprocmask (SIG_UNBLOCK, &signaux, NULL);
          // Attente de SIGUSR1
             pause();

              sigemptyset (&signaux);        // sigaux = ensemble vide
              sigaddset (&signaux, SIGUSR1);  // Ajout de SIGUSR1
          // Blocage de SIGUSR1
             sigprocmask (SIG_BLOCK, &signaux, NULL);

             for (indent=mon_numero-1; indent>0; indent--)
                                                  printf ("\t\t\t");
             printf("P%d: Envoi du signal a P%d\n",mon_numero,
                                                      destinataire);
            if (kill (P[destinataire], SIGUSR1) < 0)
                  {perror("!!! pere pb kill"); exit (EXIT_FAILURE);}

          } // for
          for (indent=mon_numero-1; indent>0; indent--)
                                                  printf ("\t\t\t");
          printf("P%d: TERMINE - %d signaux reçus\n",mon_numero,
                                                  nb_signaux_recus);
          exit (0);
    } /* FILS i */
}// for

/* PERE : P1 */
mon_numero=1;
destinataire=nb_processus;

for (i=1;i<=nb_iteration;i++){
    printf("P%d: Envoi du signal a P%d\n",mon_numero, destinataire);
    if (kill (P[destinataire], SIGUSR1) < 0)
          {perror("!!! pere pb kill"); exit (EXIT_FAILURE);}
```

```
        printf("P%d: Attente du signal\n",mon_numero);
        sigemptyset (&signaux); // sigaux = ensemble vide
        sigaddset (&signaux, SIGUSR1);  // Ajout de SIGUSR1 dans signaux
      // Déblocage de SIGUSR1
        sigprocmask (SIG_UNBLOCK, &signaux, NULL);

        pause();//Attente de SIGUSR1
        sigemptyset (&signaux);         // sigaux = ensemble vide
        sigaddset (&signaux, SIGUSR1);  // Ajout de SIGUSR1 dans signaux
      // Blocage de SIGUSR1
        sigprocmask (SIG_BLOCK, &signaux, NULL);

    } //for
    printf("P%d: TERMINE - %d signaux reçus\n",mon_numero,
                                              nb_signaux_recus);
} /* main */
```

Question 5 Fonctions de manipulation de signaux POSIX

La norme POSIX a défini le type de données `sigset_t` pour contenir un ensemble de signaux et les cinq fonctions suivantes pour manipuler les ensembles de signaux :

```
int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigaddset(sigset_t *set, int signo);
int sigdelset(sigset_t *set, int signo);
```

Toutes retournent 0 si OK, -1 sinon

```
int sigismember(const sigset_t *set, int signo);
```

Retourne 1 si vrai, 0 sinon

a) Ecrire un programme qui gère un ensemble de signaux `SIG_SET`, initialement vide, et qui vous propose le menu interactif suivant :

- Ajout d'un signal donné à l'ensemble `SIG_SET`,
- Suppression d'un signal donné de l'ensemble `SIG_SET`,
- Test si un signal donné appartient à l'ensemble `SIG_SET`,
- Affichage des signaux appartenant à l'ensemble `SIG_SET`.

Question 6 Masquage de signaux POSIX :`sigprocmask(),sigpending()`

La fonction `sigprocmask()` permet de modifier la liste des signaux actuellement bloqués. La fonction `sigpending()` retourne l'ensemble des signaux bloqués pour la délivrance et qui sont pendants pour le processus appelant.

a) Ecrire un programme qui :

- bloque le signal `SIGQINT` (Conrol C), `SIGUSR1`,

- s'endort n secondes (le temps de lui envoyer un pu plusieurs exemplaires des signaux `SIGQINT` et/ou `SIGUSR1`),
- affiche les signaux pendants,
- s'endort encore n secondes
- débloque tous les signaux.
- affiche `FIN DU PROCESSUS`.

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


int main(int argc, char *argv[]){
     sigset_t signaux_bloques,
         signaux_pendants;
     int i, p;

    printf ("%d: BLOCAGE DE SIGUSR1 (%d) et de SIGINT (%d)\n",
              getpid(),SIGUSR1,SIGINT);
    sigemptyset (&signaux_bloques);      // sigaux = ensemble vide
    sigaddset (&signaux_bloques, SIGUSR1); // Ajout de SIGUSR1
    sigaddset (&signaux_bloques, SIGINT); // Ajout de SIGINT
    sigprocmask (SIG_BLOCK, &signaux_bloques, NULL); // Blocage
    sleep (30);
    p=sigpending (&signaux_pendants);

    printf ("%d: SIGNAUX PENDANTS\n", getpid());
    for (i=1;i<NSIG;i++){
          if (sigismember(&signaux_pendants, i))
              printf("\tsignal n°%d\n", i);
    }
    sleep (15);
    printf ("%d: DEBLOCAGE DE SIGUSR1 (%d) et de SIGINT (%d)\n",
                getpid(),SIGUSR1,SIGINT);
    sigprocmask (SIG_UNBLOCK, &signaux_bloques, NULL);// Déblocage
    printf ("%d: FIN DU PROCESSUS\n", getpid());

} /* main */
```

b) Quel comportement obtenez-vous de l'exécution du programme ?

Le programme se termine prématurément une fois les signaux débloqués. `SIGINT` est délivré avant `SIGUSR1`.

Question 7 La primitive : `alarm()`

L'appel à la primitive `unsigned int alarm(unsigned int sec) ;` correspond à une requête au système d'envoyer au processus appelant le signal `SIGALARM` dans `sec` secondes.

a) Ecrire un programme qui lit cinq données depuis son entrée standard et qu'il sauvegarde dans les cinq entrées d'un vecteur. Pour chaque donnée, il doit faire une requête à l'utilisateur en affichant un message lui demandant de l'introduire. Si la donnée n'est pas introduite par l'utilisateur au bout d'un temps donné (2 secondes, par exemple) le programme réitère sa requête jusqu'à ce qu'elle le soit. A chaque nouvelle requête, pour une même donnée, le temps d'attente est augmenté d'une seconde. A la fin du programme, pour chaque donnée, afficher le nombre de tentatives de requêtes qu'il a fallu ainsi que le temps écoulé avant son introduction.

```
#include <unistd.h>
#include <signal.h>
#include <stdio.h>
#include <signal.h>
#include <signal.h>

#define  NOMBRE_DONNEES 5
#define  DELAI_INITIAL  2

struct lecture {
    int donnee;
    int tentatives;
    int temps_ecoule;
} tableau[NOMBRE_DONNEES];

int delai_courant;
int indice_donnee_courante;




//---------------------------------------------------------
  void interception_signal_sigalarm(int no_signal) {
//---------------------------------------------------------
  tableau[indice_donnee_courante].tentatives++;
  tableau[indice_donnee_courante].temps_ecoule += delai_courant;

  printf("\nIntroduire la donnee d'indice %d : ",
                                      indice_donnee_courante);
  alarm(++delai_courant);
}
```



```
int main(){

    signal(SIGALRM, interception_signal_sigalarm);

    for (indice_donnee_courante = 0;
            indice_donnee_courante < NOMBRE_DONNEES;
                        indice_donnee_courante++) {
        tableau[indice_donnee_courante].tentatives = 1;
        tableau[indice_donnee_courante].temps_ecoule = 0;

        delai_courant = DELAI_INITIAL;

        printf("\nIntroduire la donnee d'indice %d : ",
                                            indice_donnee_courante);
        alarm(delai_courant);

        scanf("%d", &tableau[indice_donnee_courante].donnee);
        tableau[indice_donnee_courante].temps_ecoule += (delai_courant
                                                        - alarm(0));
    } //for

    printf("\nRang\tValeur\tTentatives\tTemps\n\n");

    for (indice_donnee_courante = 0;
            indice_donnee_courante < NOMBRE_DONNEES;
                        indice_donnee_courante++) {

     printf("%d\t%d\t%d\t\t%d\n", indice_donnee_courante,
            tableau[indice_donnee_courante].donnee,
            tableau[indice_donnee_courante].tentatives,
            tableau[indice_donnee_courante].temps_ecoule);

    } // for
} // main
```

Système d'Exploitation Unix
TP
Les appels système : les signaux

Tous les programmes devront être développés avec passage de leurs éventuels paramètres à la fonction `main (int argc, char *argv [])`. Les valeurs de retour des appels aux primitives devront être testées et les messages d'erreurs affichés avec `perror`. Les messages d'erreurs à destination de l'utilisateur se feront sur le fichier standard des erreurs `stderr`.

Question 1 Emisssion d'un signal `kill ()`

La primitive `int kill(pid_t pid, int sig)` permet l'envoi d'un signal à un processus ou à un groupe de processus. La valeur du paramètre `pid` définit l'ensemble des processus destinataires du signal `sig`.

a) Ecrire un programme qui envoie différents signaux au(x) processus selon les valeurs possibles des deux paramètres `pid` et `sig`.

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

int main(int argc, char *argv[]){
     int nb_sig;
     int sig;
     int pid;

     if (argc != 4) {
          fprintf (stderr, "usage %s: signal pid nb_signal\n", argv[0]);
          exit (EXIT_FAILURE);
     }
     printf ("\n>>> DEBUT <<<\n");
     sig=atoi(argv[1]);
     pid=atoi(argv[2]);
     for (nb_sig=atoi(argv[3]); nb_sig>0; nb_sig--){
          if (kill (pid,sig)<0){
               perror ("kill ");
               fprintf (stderr, "ERREUR: %d signaux %d restant pour %d\n",
                                                  nb_sig, sig, pid);
               exit (EXIT_FAILURE);
          }// if
          else printf("Signal %d envoyé à %d\n", sig, pid);
     }// for
     printf ("\n>>> FIN: %s signaux %d envoyés à %d <<<\n",
                                                  argv[3], sig, pid);

}// main
```


b) Que devient le processus destinataire ? Expliquer !
La terminaison du processus est le comportement par défaut à la réception du signal `SIGUSR1` ou `SIGUSR2`

Une valeur nulle (0) du paramètre `sig` correspond au test d'existence du processus mentionné.

c) Vérifier l'affirmation précédente.

d) Dans le cas précédent, y a-t-il émission de signal ? Non

Que devient le processus destinataire ? *il continu son exécution*

La fonction `int raise(int sig)` fait partie de l'interface standard du langage C et permet l'envoi du signal `sig` au processus courant.

e) Ecrire un programme qui s'envoie plusieurs signaux différents.

f) Décrire le résultat de la précédente exécution ? Expliquer !

Question 2 Attente d'un signal `pause ()`

La primitive `int pause(void)` permet de se mettre en attente de l'arrivée de signaux.

a) Ecrire un programme dont le processus correspondant se met en attente de l'arrivée de signaux. Emettre à ce processus les signaux `SIGUSR1` et `SIGSTOP`.

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

b) Que devient le processus après la délivrance du signal `SIGUSR1` ?

La terminaison du processus est le comportement par défaut à la réception du signal `SIGUSR1` ou `SIGUSR2`

c) Que devient le processus après la délivrance du signal `SIGSTOP` ?

Il passe à l'état « suspendu » (T)

d) Que se passe-t-il s'il reçoit à la suite le message `SIGCONT` ?

Il reprend son exécution et passe à l'état S ou R

Question 3 Le gestionnaire de signal ANSI C `signal ()`

La primitive `void (*signal(int signo, void (*func)(int)))(int);` permet d'installer un nouveau gestionnaire `func` pour le signal `signo`. Le gestionnaire peut être soit une fonction spécifique de l'utilisateur, soit l'une des constantes `SIG_IGN` ou `SIG_DFL`.

a) Ecrire un programme qui ignore la réception des signaux USR1 et USR2.

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

main(){
     signal (SIGUSR1, SIG_IGN);
     signal (SIGUSR2, SIG_IGN);
     pause ();
}
```

b) Vérifier que le processus ignore bien les signaux utilisateur.

c) Quel aurait été le résultat si les signaux n'étaient pas ignorés ?

La terminaison du processus est le comportement par défaut à la réception du signal SIGUSR1 ou SIGUSR2

d) Réécrire le programme précédent de façon à installer un gestionnaire spécifique pour les deux signaux USR1 et USR2. Sur réception de chacun de ces deux signaux, le gestionnaire doit afficher la nature (USR1 ou USR2) du signal reçu ainsi que le nombre total d'occurrences reçues de ce même signal.

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

int nb_sigusr1=0;
int nb_sigusr2=0;

void gest_sigusr1 (int sig) {
     nb_sigusr1++;
     printf ("recu %d signal %d (SIG_USR1)\n", nb_sigusr1, sig);
} //gest_sigusr1

void gest_sigusr2 (int sig) {
     nb_sigusr2++;
     printf ("recu %d signal %d (SIG_USR2)\n", nb_sigusr2, sig);
} //gest_sigusr2


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

     signal (SIGUSR1, gest_sigusr1);
     signal (SIGUSR2, gest_sigusr2);
     for (;;)
          pause ();

}// main
```


La boucle for est ici nécessaire car la fin de l'exécution du gestionnaire d'un signal capté provoque la fin d'exécution de la primitive pause.

e) Ecrire un programme qui envoie un grand nombre de signaux au programme précédent. Quel résultat obtenez-vous lorsque vous émettez plusieurs occurrences des signaux précédents ? Expliquer !

On constate que tous les signaux envoyés ne sont pas traités par leur destinataire. Ce n'est pas le nombre de signaux qui est en cause mais leur fréquence d'envoi qui est trop élevée. Certains signaux sont perdus chez le destinataire car il ne peut y avoir qu'une seule instance de signal pendant.

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

int main(int argc, char *argv[]){
int nb_sig;
int sig;
int pid;

if (argc != 4) {
     fprintf (stderr, "usage %s: signal pid nb_signal\n", argv[0]);
     exit (EXIT_FAILURE);
}
printf ("\n>>> DEBUT <<<\n");
sig=atoi(argv[1]);
pid=atoi(argv[2]);
for (nb_sig=atoi(argv[3]); nb_sig>0; nb_sig--){

     if (kill (pid,sig)<0){
          perror ("kill ");
          fprintf (stderr, "ERREUR: %d signaux %d restant pour %d\n",
                                                  nb_sig, sig, pid);
          exit (EXIT_FAILURE);
     }// if
     else printf("Signal %d envoyé à %d\n", sig, pid);
}// for
printf ("\n>>> FIN: %s signaux %d envoyés à %d <<<\n",
                                          argv[3], sig, pid);
}// main
```

Question 4 Le gestionnaire de signal POSIX `sigaction ()`

La primitive `int sigaction(int signo, const struct sigaction *act, struct sigaction *oldact);` permet de modifier l'action effectuée par un processus lors de la réception d'un signal spécifique

a) Ecrire un programme de façon à installer un gestionnaire spécifique pour les deux signaux USR1 et USR2. Sur réception de chacun de ces deux signaux, le gestionnaire doit afficher la nature (USR1 ou USR2) du signal reçu ainsi que le nombre total d'occurrences reçues de ce même signal.

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

int nb_sigusr1=0;
int nb_sigusr2=0;
struct sigaction action;

void gest_sigusr1 (int sig) {
    nb_sigusr1++;
    printf ("recu %d signal %d (SIG_USR1)\n", nb_sigusr1, sig);
} //gest_sigusr1

void gest_sigusr2 (int sig) {
    nb_sigusr2++;
    printf ("recu %d signal %d (SIG_USR2)\n", nb_sigusr2, sig);
} //gest_sigusr2

// ====================================
    int main(int argc, char *argv[]){
// ====================================
    // Installation du gestionnaire pour SIG_USR1
    action.sa_handler=gest_sigusr1;
    sigemptyset(&action.sa_mask);
    if (sigaction (SIGUSR1, &action, NULL)<0){perror("!!!pb signal");
        exit (EXIT_FAILURE);}

    // Installation du gestionnaire pour SIG_USR2
    action.sa_handler=gest_sigusr2;
    sigemptyset(&action.sa_mask);
    if (sigaction (SIGUSR2, &action, NULL)<0) {perror("!!!pb signal");
    exit (EXIT_FAILURE);}
    for (;;)
        pause ();
}// main
```


b) Envoyer un grand nombre de signaux au programme au programme précédent. Quel résultat obtenez-vous lorsque vous émettez plusieurs occurrences des signaux précédents ? Expliquer ! Comment y remédier ?

Lorsque un grand nombre de signaux est envoyé avec une fréquence élevée des signaux sont perdus: nombre signaux envoyés > nombre signaux reçus. Ceci est dû au fait qu'il ne peut y avoir qu'une seule instance d'un type de signal delivré, pendant ou bloqué à la fois. Un premier élément de solution est de procéder comme en c)

c) Ecrire un programme qui crée un processus fils. Le père itère un grand nombre de fois :

- Envoyer le signal SIGUSR1 à mon fils
- Attendre que mon fils m'envoie le signal SIGUSR1

Le fils itère le même nombre de fois que son père:

- Attendre que mon père m'envoie le signal SIGUSR1
- Envoyer le signal SIGUSR1 à mon père

En fin d'exécution, le père affiche le nombre de SIGUSR1 envoyé et le fils le nombre de SIGUSR1 reçu. Que se passe-t-il ? Expliquer et proposer une solution

Le programme se bloque dès qu'un signal est reçu alors qu'on ne l'attend pas (`pause`). Il faut utiliser `sigsuspend` qui modifie le masque et suspend le processus de manière atomique

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

#define MAX_P 10

int P[MAX_P+1];
int nb_processus;
int nb_iteration;
int destinataire;
int nb_signaux_recus;
int mon_numero;
int indent;
int i;

sigset_t signaux;
struct sigaction action;
```

```
void traiter_signal(int num_signal){
    int expediteur;

    if (mon_numero==nb_processus)
        expediteur=1;
    else expediteur=mon_numero+1;

    for (indent=mon_numero-1; indent>0; indent--) printf ("\t\t\t");
    printf("P%d: Recu signal de P%d\n", mon_numero, expediteur);
    nb_signaux_recus++;

    return;
}

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

    if (argc != 3) {
         fprintf (stderr, "usage %s: nb_processus nb_envoi_signal\n",
                                                              argv[0]);
         exit (EXIT_FAILURE);
    }

    nb_processus=atoi(argv[1]);
    if (nb_processus < 1) {
         fprintf (stderr, "usage %s: nb_processus doit etre 1\n",argv[0]);
         exit (EXIT_FAILURE);
    }
    nb_iteration=atoi(argv[2]);

 // Installation du gestionnaire pour SIG_USR1
    action.sa_handler=traiter_signal;
    sigemptyset(&action.sa_mask);
    if (sigaction (SIGUSR1, &action, NULL)<0)
              {perror("!!!pb signal"); exit (EXIT_FAILURE);}

    sigemptyset (&signaux);          // sigaux = ensemble vide
    sigaddset (&signaux, SIGUSR1);  // Ajout de SIGUSR1 dans signaux
    sigprocmask (SIG_BLOCK, &signaux, NULL); // Blocage de SIGUSR1

    sigemptyset (&signaux); // signaux = ensemble vide pour sigsuspend

    P[1]=getpid();

    for (i=2; i<=nb_processus; i++){
        P[i]=fork();
```



```
    if (P[i]==0){ /* FILS i */
          nb_signaux_recus=0;
        mon_numero=i;
        destinataire=(i-1);

        for (i=1; i<=nb_iteration; i++){
            for (indent=mon_numero-1; indent>0; indent--)
                                                  printf ("\t\t\t");
            printf("P%d: Attente du signal\n",mon_numero);

          // Remplacement du masque courant par l'ensemble "signaux"
          // et attente d'un signal
             sigemptyset (&signaux);       // sigaux = ensemble vide
             sigaddset (&signaux, SIGUSR1);// Ajout de SIGUSR1
          // Déblocage de SIGUSR1
             sigprocmask (SIG_UNBLOCK, &signaux, NULL);
          // Attente de SIGUSR1
             pause();

              sigemptyset (&signaux);        // sigaux = ensemble vide
              sigaddset (&signaux, SIGUSR1);  // Ajout de SIGUSR1
          // Blocage de SIGUSR1
             sigprocmask (SIG_BLOCK, &signaux, NULL);

             for (indent=mon_numero-1; indent>0; indent--)
                                                  printf ("\t\t\t");
             printf("P%d: Envoi du signal a P%d\n",mon_numero,
                                                      destinataire);
            if (kill (P[destinataire], SIGUSR1) < 0)
                  {perror("!!! pere pb kill"); exit (EXIT_FAILURE);}

          } // for
          for (indent=mon_numero-1; indent>0; indent--)
                                                  printf ("\t\t\t");
          printf("P%d: TERMINE - %d signaux reçus\n",mon_numero,
                                                  nb_signaux_recus);
          exit (0);
    } /* FILS i */
}// for

/* PERE : P1 */
mon_numero=1;
destinataire=nb_processus;

for (i=1;i<=nb_iteration;i++){
    printf("P%d: Envoi du signal a P%d\n",mon_numero, destinataire);
    if (kill (P[destinataire], SIGUSR1) < 0)
          {perror("!!! pere pb kill"); exit (EXIT_FAILURE);}
```

```
        printf("P%d: Attente du signal\n",mon_numero);
        sigemptyset (&signaux); // sigaux = ensemble vide
        sigaddset (&signaux, SIGUSR1);  // Ajout de SIGUSR1 dans signaux
      // Déblocage de SIGUSR1
        sigprocmask (SIG_UNBLOCK, &signaux, NULL);

        pause();//Attente de SIGUSR1
        sigemptyset (&signaux);         // sigaux = ensemble vide
        sigaddset (&signaux, SIGUSR1);  // Ajout de SIGUSR1 dans signaux
      // Blocage de SIGUSR1
        sigprocmask (SIG_BLOCK, &signaux, NULL);

    } //for
    printf("P%d: TERMINE - %d signaux reçus\n",mon_numero,
                                              nb_signaux_recus);
} /* main */
```

Question 5 Fonctions de manipulation de signaux POSIX

La norme POSIX a défini le type de données `sigset_t` pour contenir un ensemble de signaux et les cinq fonctions suivantes pour manipuler les ensembles de signaux :

```
int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigaddset(sigset_t *set, int signo);
int sigdelset(sigset_t *set, int signo);
```

Toutes retournent 0 si OK, -1 sinon

```
int sigismember(const sigset_t *set, int signo);
```

Retourne 1 si vrai, 0 sinon

a) Ecrire un programme qui gère un ensemble de signaux `SIG_SET`, initialement vide, et qui vous propose le menu interactif suivant :

- Ajout d'un signal donné à l'ensemble `SIG_SET`,
- Suppression d'un signal donné de l'ensemble `SIG_SET`,
- Test si un signal donné appartient à l'ensemble `SIG_SET`,
- Affichage des signaux appartenant à l'ensemble `SIG_SET`.

Question 6 Masquage de signaux POSIX :`sigprocmask(),sigpending()`

La fonction `sigprocmask()` permet de modifier la liste des signaux actuellement bloqués. La fonction `sigpending()` retourne l'ensemble des signaux bloqués pour la délivrance et qui sont pendants pour le processus appelant.

a) Ecrire un programme qui :

- bloque le signal `SIGQINT` (Conrol C), `SIGUSR1`,

- s'endort n secondes (le temps de lui envoyer un pu plusieurs exemplaires des signaux `SIGQINT` et/ou `SIGUSR1`),
- affiche les signaux pendants,
- s'endort encore n secondes
- débloque tous les signaux.
- affiche `FIN DU PROCESSUS`.

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


int main(int argc, char *argv[]){
     sigset_t signaux_bloques,
         signaux_pendants;
     int i, p;

    printf ("%d: BLOCAGE DE SIGUSR1 (%d) et de SIGINT (%d)\n",
              getpid(),SIGUSR1,SIGINT);
    sigemptyset (&signaux_bloques);      // sigaux = ensemble vide
    sigaddset (&signaux_bloques, SIGUSR1); // Ajout de SIGUSR1
    sigaddset (&signaux_bloques, SIGINT); // Ajout de SIGINT
    sigprocmask (SIG_BLOCK, &signaux_bloques, NULL); // Blocage
    sleep (30);
    p=sigpending (&signaux_pendants);

    printf ("%d: SIGNAUX PENDANTS\n", getpid());
    for (i=1;i<NSIG;i++){
          if (sigismember(&signaux_pendants, i))
              printf("\tsignal n°%d\n", i);
    }
    sleep (15);
    printf ("%d: DEBLOCAGE DE SIGUSR1 (%d) et de SIGINT (%d)\n",
                getpid(),SIGUSR1,SIGINT);
    sigprocmask (SIG_UNBLOCK, &signaux_bloques, NULL);// Déblocage
    printf ("%d: FIN DU PROCESSUS\n", getpid());

} /* main */
```

b) Quel comportement obtenez-vous de l'exécution du programme ?

Le programme se termine prématurément une fois les signaux débloqués. `SIGINT` est délivré avant `SIGUSR1`.

Question 7 La primitive : `alarm()`

L'appel à la primitive `unsigned int alarm(unsigned int sec) ;` correspond à une requête au système d'envoyer au processus appelant le signal `SIGALARM` dans `sec` secondes.

a) Ecrire un programme qui lit cinq données depuis son entrée standard et qu'il sauvegarde dans les cinq entrées d'un vecteur. Pour chaque donnée, il doit faire une requête à l'utilisateur en affichant un message lui demandant de l'introduire. Si la donnée n'est pas introduite par l'utilisateur au bout d'un temps donné (2 secondes, par exemple) le programme réitère sa requête jusqu'à ce qu'elle le soit. A chaque nouvelle requête, pour une même donnée, le temps d'attente est augmenté d'une seconde. A la fin du programme, pour chaque donnée, afficher le nombre de tentatives de requêtes qu'il a fallu ainsi que le temps écoulé avant son introduction.

```
#include <unistd.h>
#include <signal.h>
#include <stdio.h>
#include <signal.h>
#include <signal.h>

#define  NOMBRE_DONNEES 5
#define  DELAI_INITIAL  2

struct lecture {
    int donnee;
    int tentatives;
    int temps_ecoule;
} tableau[NOMBRE_DONNEES];

int delai_courant;
int indice_donnee_courante;




//---------------------------------------------------------
  void interception_signal_sigalarm(int no_signal) {
//---------------------------------------------------------
  tableau[indice_donnee_courante].tentatives++;
  tableau[indice_donnee_courante].temps_ecoule += delai_courant;

  printf("\nIntroduire la donnee d'indice %d : ",
                                      indice_donnee_courante);
  alarm(++delai_courant);
}
```



```
int main(){

    signal(SIGALRM, interception_signal_sigalarm);

    for (indice_donnee_courante = 0;
            indice_donnee_courante < NOMBRE_DONNEES;
                        indice_donnee_courante++) {
        tableau[indice_donnee_courante].tentatives = 1;
        tableau[indice_donnee_courante].temps_ecoule = 0;

        delai_courant = DELAI_INITIAL;

        printf("\nIntroduire la donnee d'indice %d : ",
                                            indice_donnee_courante);
        alarm(delai_courant);

        scanf("%d", &tableau[indice_donnee_courante].donnee);
        tableau[indice_donnee_courante].temps_ecoule += (delai_courant
                                                        - alarm(0));
    } //for

    printf("\nRang\tValeur\tTentatives\tTemps\n\n");

    for (indice_donnee_courante = 0;
            indice_donnee_courante < NOMBRE_DONNEES;
                        indice_donnee_courante++) {

     printf("%d\t%d\t%d\t\t%d\n", indice_donnee_courante,
            tableau[indice_donnee_courante].donnee,
            tableau[indice_donnee_courante].tentatives,
            tableau[indice_donnee_courante].temps_ecoule);

    } // for
} // main
```