Université de Strasbourg

S31 Sujet N°2

Comprendre les Systèmes

gancarski@unistra.fr

IUT Robert Schuman Université de Strasbourg

Signaux

Objectif : Comprendre la synchronisation de processus père/fils

Notions : Appels système liés aux signaux

---

1 IPC - Signaux

Les signaux sont un moyen simple d'établir une communication entre processus.

→ Un processus émetteur envoie un signal vers un processus récepteur. Le processus récepteur est interrompu dans son travail et enclenche un action (handler) associée à ce signal puis il reprend son exécution là où il avait été interrompu. Il peut aussi décider de l'ignorer. Dans tous les cas, il ne sait pas qui lui a envoyé ce signal.

Il en existe 31 signaux (voir `man 2 signal`) dont :
- SIGINT = 2 program interrupt, émis lors d'un Ctrl-C,
- SIGTERM = 15 demande de fin de processus,
- SIGKILL = 9 arrêt immédiat de processus,
- SIGFPE = 8 erreur arithmétique,
- SIGALRM = 14 fin de délai, voir appel système `alarm()`.
- SIGUSR1 = 10 et SIGUSR2 = 12, réservés pour l'utilisateur (Attention : 30 et 31 sur MacOs)

Par exemple, lorsque que l'utilisateur tape Ctrl-C, un gestionnaire de signal (processus système) envoie le signal SIGINT au processus en cours d'exécution dans le terminal. Le traitement de ce signal va dépendre du processus :
- Si c'est le shell, il va l'ignorer
- Si c'est un processus en cours d'exécution, il va (en général) réagir au signal SIGINT en s'interrompant et en se fermant proprement

D'où plusieurs questions :
- Comment un signal peut-il être envoyé ?
- Comment un processus peut-il "choisir" l'opération (handler) à faire lors de la réception ?
- Comment des signaux qui peuvent arriver quasi-simultanément ou être répétés sont-ils traités ?
- Quelles informations un signal peut-il transmettre au récepteur ?
- ...

2 Un exemple concret

Une communication entre deux processus se compose de trois "phases " :

1. Ecriture du code du futur processus "récepteur"
2. Lancement du processus "récepteur"
3. Envoi du signal par un processus "émetteur"

2.1 Ecriture et lancement du processus "récepteur"

Pour qu'un processus puisse intercepter un signal en C, il faut assigner au signal la fonction qui sera exécutée lorsque le signal est reçu par le processus. En général on utilise un mécanisme POSIX en trois étapes :

1. Ecriture de la fonction qui sera appelée lors de la réception du signal
2. Création d'un "formulaire" qui permet de définir l'association entre le signal choisi et cette fonction. Attention, il s'agit bien uniquement d'un formulaire!!!
3. Appel système pour associer concrètement, via ce formulaire, le signal à la fonction : dans un table d'association des signaux, le processus stocke dans la case correspondant au numéro du signal, l'adresse du code de la fonction associée

2.1.1 Ecriture de la fonction

La fonction associée à un signal, doit avoir le profil : `void <nom de la fonction>(int S)` où `S` contiendra le numéro du signal. Par exemple :

```
void Frecu{ printf(" Signal reçu %d \n", S);
```

Pourquoi d'après vous ?

2.1.2 Le "formulaire"

La création d'un "formulaire" qui va servir à demander l'association entre un signal et une fonction se fait en déclarant une structure du type `struct sigaction` définie (dans `signal.h`) par :

```
struct sigaction {
    void     (*sa_handler) (int);  // pointeur vers la fonction à exécuter
    sigset_t   sa_mask;            // signaux à bloquer pendant l'exécution du gestionnaire
    int        sa_flags;           // attributs modifiant le comportement du signal
}
```

Exemple :

```
main() {
  struct sigaction new_action;
  new_action.sa_handler = frecu;    // Les autres champs sont laissés par défaut
}
```

Le processus peut aussi décider d'ignorer le signal : `SIG_IGN`

2.1.3 Association

Elle se fait par l'appel système `sigaction` :

```
int sigaction(int sig,  const struct sigaction *new_action; struct sigaction *old_action)
```

Exemple :

```
main() {
  struct sigaction new_action, old_action;
  new_action.sa_handler = frecu;
  sigaction(SIGUSR1, &new_action, &old_action);
  ...
}
```

2.2 Le processus "récepteur"

Après compilation, le processus récepteur va être lancé classiquement. Deux actions sont alors possibles après avoir associé un signal et une fonction :

1. Le processus continue et sera interrompu lors de la réception du signal : il n'y a rien à faire de spécial
2. Le processus se met en attente d'un signal : `pause`

1.1.1 - Compléter le code ci-dessous de façon à ce que la fonction `frecu1` soit assignée aux signaux `SIGUSR1` et `frecu2` à `SIGUSR2`.

```
void frecu1(int sigrecu) {  printf(" La fonction freçu signal reçu %d\n", sigrecu); }
void frecu2(int sigrecu) {  printf(" La fonction freçu signal reçu %d\n", sigrecu); }

int main(int argc, char *argv[]){
  struct sigaction new_action;

  new_action.sa_handler = ... ;
  sigaction (..., &..., NULL);

// Mise en attente du signal
  ...
  }
```

1.1.2 - Comment pourrait-on faire pour que sur réception de `SIGUSR2`, le processus exécute `frecu1` et `frecu2`

1.1.3 - Comment modifier le programme pour que le processus sache qui lui a envoyé le message.

2.3 Envoi d'un signal

Il existe deux possibilités pour envoyer un signal à un processus de pid XXX :

— Envoi directement en ligne de commande (depuis un shell)

```
$> kill –SIGUSR1 XXX
```

— Envoi d'un signal depuis un processus C –> Fonction `int kill(pid_t pid, int sig);`.

```
main() { ...

  kill(XXX,SIGUSR1);
  ...
}
```

2.3.1 - Attention, dans le premier cas, il s'agit d'une commande Linux alors que dans le deuxième cas, c'est un appel système en C. Comment peut-on les différencier lorsque fait un `man` ?

2.3.2 - Comment peut-on envoyer un signal à un processus dont on ne connait pas le pid ?

2.3.3 - Modifier le code de `S` de façon à ce qu'il sache si `P1` a bien reçu le signal et s'il l'a traité ou non

2.3.4 - Modifier le code de `S` de façon à ce `S` envoie son pid (via le signal) afin que `P1` sache qui a émis le signal reçu

3 Mise en pratique

3.1 Premiers pas

Les "réponses" aux questions 1.1.1, 3.1.1 à 3.1.3 seront à implanter en TP.

Soit `P1` le processus associé de la question 1.1.1.

3.1.1 - Comment faire pour envoyer à P1 un des deux signaux depuis le clavier.

3.1.2 - Idem mais on suppose que c'est un autre processus `S` qui envoie une fois un des deux signaux. Il se peut que P1 se met en attente définitivement : Pourquoi ? Y-a-il une solution à ce problème ?

3.1.3 - Que faut-il modifier dans P1 pour qu'il puissent recevoir à la suite les deux signaux différents ? Idem mais deux fois le même signal ?

3.1.4 - On suppose que `S` envoie très rapidement 3 fois de suite le signal SIG_USR1 (On suppose que `P1` fait bien trois `pause()`). Que va-t-il se passer d'après vous ? Y-a-il une solution à ce problème ?

3.1.5 - *(Facultatif)* On suppose que `S` envoie plusieurs fois ces signaux. Modifier `P1` pour qu'il fasse une boucle infinie et compte le nombre de signaux qu'il reçoit. Il doit afficher le nombre final lorsqu'il reçoit "Ctrl-C"

3.2 Signaux et fork

Les questions 3.2.1 et 3.2.2 seront à réaliser en TP. Idem pour la question 3.2.3 mais c'est facultatif.

3.2.1 - Modifiez P1 afin qu'il crée deux fils F1 et F2 puis envoie à F1 le signal SIGUSR1 et à F2 le signal SIGUSR2

3.2.2 - Modifier P1 pour qu'il envoie le signal SIGUSR1 à son fils F1 et SIGUSR2 à son fils F2 mais en plus :

- Le fils F1 pourra traiter SIGUSR1 et ignorera SIGUSR2.
- Le fils F2 pourra traiter SIGUSR2 et ignorera SIGUSR1.
- Le père P1 pourra continuer à traiter les deux signaux.

3.2.3 - Soit un processus `P2` qui lance un fils. Comment peut-il savoir si ce fils a terminé ?

4 Serveur de calcul

Fort de tout ce nouveau savoir, avancez dans la production des versions V1a et V2a du serveur de calcul.