Année 2009-2010
Étape : Licence Sciences Technologies (semestre 1)
UE : INF355 (Programmation Système)
Modalités : Épreuve sans document

Date : 15 décembre 2009
Heure : 8h30
Durée : 1h30

Lisez l'intégralité du sujet *avant de commencer à répondre calmement aux questions.*

1 Signaux (question d'échauffement)

Sous Unix, il existe un appel système (`setitimer`) permettant d'armer un temporisateur qui déclenche périodiquement l'envoi d'un signal d'alarme (`SIGALRM`) au processus à la fréquence demandée.

Que se passe-t-il si le processus installe un traitant de signal pour `SIGALRM` dont la durée d'exécution dépasse l'intervalle de temps fixé entre les signaux ? Même question si, tout au début du traitant de signal, l'on ré-autorise la délivrance de `SIGALRM` avec `sigprocmask` ? Expliquez bien la différence entre ces deux situations.

2 Communication entre processus

On souhaite programmer une application composée de trois processus et dans laquelle un processus sert de « *console* d'affichage » pour les deux autres. L'idée est que chacun de ces deux processus est connecté à la console via un tube distinct dans lequel il envoie des messages qu'il souhaite afficher. Le processus jouant le rôle de la *console* extrait donc les messages qu'il reçoit depuis ces deux tubes, et les affiche en les préfixant par `<Proc0>` ou `<Proc1>` pour indiquer de quel processus ils sont issus. Voici le squelette de ce programme :

```
int fd0, fd1;

void envoyer(char *msg)
{ ... }

int recevoir(int fd, char *buf)
{
  int n;

  if(read(fd, &n, sizeof(int)) != sizeof(int))
    return -1;

  return (read(fd, buf, n) == n ? 0 : -1);
}

void console(void)
{
  int n;
  char buf[1024];

  while (fd0 != -1 || fd1 != -1) {
    if(fd0 != -1)
      if(recevoir(fd0, buf) == 0)
        printf("<Proc0> %s\n", buf);
      else
        fd0 = -1;
    if(fd1 != -1)
      if(recevoir(fd1, buf) == 0)
        printf(<Proc1> %s\n", buf);
      else
        fd1 = -1;
  }
}
```

```
void fils0(void)
{ envoyer("coucou !"); envoyer("ça va ?"); }

void fils1(void)
{ envoyer("salut !"); }

int main(int argc, char *argv[])
{
  int tube[2];

  ... /* Création d'un tube et
         lancement d'un nouveau processus qui
         appelle la fonction fils0() puis se
         termine */

  fd0 = tube[0];
  close(tube[1]);

  ... /* Création d'un second tube et
         lancement d'un nouveau processus qui
         appelle la fonction fils1() puis se
         termine */

  fd1 = tube[0];
  close(tube[1]);

  console();
  return 0;
}
```

Question 1 Complétez le code de la fonction `main` de façon à créer les deux processus fils qui devront être chacun connecté au père par nouveau tube distinct. La sortie standard de chaque fils sera être redirigée le tube correspondant. Veillez à bien fermer tous les descripteurs inutiles de part et d'autre.

Question 2 En observant la fonction `recevoir` (dont les paramètres sont respectivement un descripteur vers l'extrémité d'un tube en lecture, et un pointeur vers une zone où on stockera le message reçu), on remarque que les messages émis par les processus se composent de deux parties : un entier spécifiant la taille du message précède toujours le contenu (une chaîne de caractère) de celui-ci.

Donnez le code de la fonction `envoyer`, qui écrit sur la sortie standard un message construit à partir de la chaîne de caractère passée en paramètre.

Question 3 Indiquez la sortie standard produite par le programme dans sa version actuelle. Même question dans le cas où un appel `sleep(10)` au début de la fonction `fils1`. De manière plus générale, dans quel ordre la *console* affiche-t-elle les messages qu'elle reçoit ? Est-ce que cela correspond toujours à l'ordre dans lequel les messages ont été transmis ?

Question 4 Pour corriger le problème, une idée est d'utiliser deux *threads* dans le processus « *console* » : chaque thread sera ainsi chargé de recevoir et d'afficher les messages reçu depuis l'un des fils. Donnez les modifications à apporter au programme pour implémenter cette solution.

Question 5 Sur un système d'exploitation ne disposant ni des *threads* ni de l'appel système `select`, une autre solution au problème est d'utiliser des signaux. L'idée est qu'un processus envoie un signal à la console à chaque fois qu'il émet un message dans le tube. En utilisant un signal différent pour chaque fils (e.g. `SIGUSR1` et `SIGUSR2`), la console peut alors extraire le message du tube correspondant. Donnez le code de cette solution.

Question 6 Dans la solution donnée à la question précédente, peut-il y avoir perte de signaux ? Pourquoi ? Le cas échéant, indiquez (mais on ne demande pas le code) comment on pourraît corriger le problème ?

Question 7 On souhaite maintenant que le fils 0 lance la commande `ls -l` et récupère sa sortie pour la formater correctement (transformation en messages) avant de l'envoyer vers la console. Donnez le code de la nouvelle fonction `fil0()`. On pourra supposer que la taille des lignes n'excède jamais 1023 caractères.