Licence Sciences Technologies

UNIVERSITÉ BORDEAUX 1 Sciences Technologies

Année 2003/2004 — Session de Mai 2004

GU : Licence Sciences Technologies — **UE :** MIAS421

Épreuve de : Programmation Système

Date : 14 mai 2004 (Saint Matthias) — **Durée :** 1h30

Notes de cours autorisées

Épreuve de Monsieur NAMYST

Lisez l'intégralité du sujet *avant de commencer à répondre aux questions. Certaines peuvent être traitées même si vous n'avez pas répondu aux précédentes.*

1 Signaux

1.1 Délivrance

Question 1 Un signal n'est délivré à un processus qu'à l'occasion d'un changement d'état bien précis. Indiquez lequel ?

Est-ce à dire qu'un processus exécutant une bouche infinie (`while(1);`) ne se verra jamais délivrer un signal ? Expliquez.

Question 2 Sous Unix, il existe un appel système (`setitimer`) permettant d'armer de manière précise un temporisateur qui déclenchera périodiquement un signal d'alarme (`SIGALRM`) pour le processus à la fréquence demandée. Il est alors aisé, en installant un traitant adéquat, d'effectuer périodiquement certains travaux dans un programme (ramasse miettes, animations graphiques, etc.)

Que se passera-t-il si la durée de l'exécution du traitant de signal 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 de comportement entre ces deux situations.

1.2 Récupération de la terminaison d'un processus

On veut pouvoir récupérer la terminaison d'un processus fils au plus vite pour éviter la présence de processus zombies sur la machine. Puisque l'on sait que lorsqu'un processus se termine, le signal `SIGCHLD` est automatiquement envoyé à son père, on propose la solution suivante :

```
int pid;

void handler(int sig)
{
  int r, status;

  r = waitpid(pid, &status, 0);
  if(r == -1) {
    printf("Le fils %d n'existe pas!\n", pid);
  }
}
```

```
int main()
{
  sigaction(SIGCHLD, handler);

  pid = fork();
  if(pid == 0) { /* fils */
    ...
    exit(0);
  }
  ...
  return 0;
}
```

Question 1 Expliquez pourquoi, dans certaines conditions, le programme ne fonctionnera pas comme prévu, c'est-à-dire que le message d'erreur de la fonction `handler` apparaîtra à l'écran.

Question 2 Proposez une correction du programme qui corrige ce problème.

Question 3 Dans le cas plus général d'un programme créant plusieurs processus durant son exécution, y a-t-il un risque que la fonction `handler` ne soit pas appelée le bon nombre de fois (i.e. un nombre de fois égal au nombre de fils terminés) ? Expliquez.

Donnez une version de la fonction `handler` qui fonctionne quand même dans ces conditions.

1.3 Tracé d'une fonction

Voici la boucle principale d'un programme qui dessine à l'écran le tracé d'une fonction mathématique `f`.

```
float f(float x) { ... }

int main(int argc, char **argv)
{
  float x, y;
  ...
  for(x = MIN_RANGE; x <= MAX_RANGE; x = x + DELTA) {
    y = f(x);
    afficher_point(x, y);
  }
  ...
}
```

On voudrait que ce programme fonctionne même avec les fonctions qui ne sont pas définies en tout point de l'intervalle (ex : $f = \frac{1}{x}$), c'est-à-dire qu'il soit capable de passer à l'itération suivante lorsque le calcul de $f(x)$ échoue. On suppose que le signal `SIGFPE` est levé lorsque l'on tente d'évaluer la fonction f en un point où elle n'est pas définie.

Donnez une nouvelle version de ce programme qui fonctionne comme demandé.

2 Tubes

On dispose d'une fonction `void mystere(void)` (dont on n'a pas le code source) qui génère un certain volume d'octets sur la sortie standard. On voudrait écrire un programme qui récupère ces octets dans un tableau (on supppose que le volume est inférieur à `VERY_BIG_SIZE` octets) pour pouvoir leur appliquer certains traitements ultérieurement. Voici une première proposition pour réaliser cette sorte de redirection :

```
int main()
{
  char buf[VERY_BIG_SIZE];
  int tube[2], n;

  pipe(tube); dup2(tube[1], 1); close(tube[1]);

  mystere();

  close(1);
  n = read(tube[0], buf, VERY_BIG_SIZE);
  ...
}
```

Question 1 Le programme proposé fonctionne lorsque le volume d'octets générés par la fonction `mystere` est petit, mais ne se termine pas lorsque le volume devient trop important (tout en restant inférieur à `VERY_BIG_SIZE`).

Expliquez pourquoi on observe ce comportement. À quoi correspond la valeur seuil à partir de laquelle le programme ne fonctionne plus ?

Question 2 Proposez une nouvelle version du programme qui fonctionne pour tout volume d'octets compris entre 0 et `VERY_BIG_SIZE`. Note : on pourra utiliser un processus supplémentaire au besoin.