R. Namyst
LaBRI, bureau 322

le 6 août 2004

Licence informatique — Programmation Système
Corrigé de l'examen de juin

1 Signaux

1.1 Délivrance

Question 1 Les signaux sont délivrés à un processus lors d'un passage de l'état *actif noyau* à l'état *actif utilisateur*.

Un processus exécutant une boucle infinie (`while(1);`) pourra quand même recevoir des signaux, car il effectue régulièrement des passages à l'état noyau grâce aux interruptions de l'horloge.

Question 2 Si la durée de l'exécution du traitant de signal dépasse l'intervalle de temps fixé, alors les signaux arriveront pendant l'exécution du traitant. Dans le premier cas de figure, ces signaux (en fait, un seul d'entre eux à chaque fois) seront *différés* à la fin de l'exécution du traitant, puisqu'ils sont automatiquement masqués pendant son exécution. Le traitant sera donc exécuté continuellement en séquence sans retour au programme principal.

Dans le cas où l'on ré-autorise la délivrance de `SIGALRM` avec `sigprocmask` en début de traitant (et en supposant que les signaux soient délivrés après l'appel à `sigprocmask`), l'arrivée d'un signal provoquera immédiatement l'interruption du traitant en cours et un appel supplémentaire ("récursif") du traitant sera effectué. Il y aura donc empilement des exécutions de traitant (sans jamais dépiler) et le programme finira par s'interrompre à cause d'un débordement de la pile...

1.2 Récupération de la terminaison d'un processus

Question 1 Si l'exécution du processus fils se termine avant même que le processus père ne soit "sorti" du `fork`, alors l'exécution du traitant de signal `handler` aura lieu avant même que la variable `pid` soit initialisée. Par conséquent, l'appel à `waitpid` aura lieu avec un paramètre ayant une valeur arbitraire, et échouera dans certains cas.

Question 2 Pour corriger le programme, il suffit de remplacer `waitpid(...)` par un appel à `wait(...)` qui n'utilise pas de paramètre `pid`.

Question 3 Dans le cas plus général d'un programme créant plusieurs processus durant son exécution, il se peut que la fonction `handler` ne soit pas appelée le bon nombre de fois. En effet, si plusieurs fils se terminent de manière très proche, les signaux `SIGCHLD` correspondants ne pourront être accumulés dans la table des signaux "pendants", autrement dit certains signaux seront perdus.

Pour récupérer correctement la terminaison de tous les fils, il suffit de modifier la fonction `handler` comme suit :

```
int pid;

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

  do {
    r = wait(&status);
  } while(r != -1);
}
```

1.3 Tracé d'une fonction

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

jmp_buf buf;

void handler(int sig)
{
   siglongjmp(buf, 1);
}

int main(int argc, char **argv)
{
  ...
  sigaction(SIGFPE, handler);

  for(x = MIN_RANGE; x <= MAX_RANGE; x = x + DELTA) {
    if(sigsetjmp(buf, 1) == 0) {
      y = f(x);
      afficher_point(x, y);
    }
  }
  ...
}
```

(en utilisant la notation simplifiée pour `sigaction` vue en cours)

2 Tubes

Question 1 Si le volume des données générées par la fonction `mystere` dépasse la *capacité du tube*, alors l'écriture devient bloquante jusqu'à ce qu'une éventuelle opération de lecture libère à nouveau de la place dans le tube, ce qui n'arrivera jamais dans ce programme puisque l'opération `read` est placée derrière l'appel à la fonction `mystere`...

Question 2

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

  pipe(tube);
  if(fork() == 0) {
    close(tube[0]); dup2(tube[1], 1); close(tube[1]);

    mystere();

    exit(0);
  }

  close(tube[1]);

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