Université de Bretagne Sud
Concurrence
Controle Continu
Spool de processus

Luc Courtrai

Le rendu du contrôle sera fichier C. Vous n'avez pas nécessairement besoin d'un compilateur C pour le réaliser. Vous pouvez rendre un fichier texte contenant du code C. Cinq minutes avant la fin du TP dont à 11H10 vous devez avoir déposer votre fichier (éventuellement un .zip si vous avez plusieurs fichiers).

Enfin, dans le code vous ajouterez des traces d'information (printf) lors d'un évènement sur un processus (exemple : lancement du processus, mise en attente, sortie de l'attente, dépôt d'un répertoire dans le tampon, lecture d'un répertoire dans le tampon ...

Le contrôle étant en distanciel, je serai très virgulent lors de la correction aux éventuelles fraudes. Je mettrai dans ce cas systématiquement zéro à la copie du tricheur et à celui qui a diffusé son code.

Find : Recherche en parallèle avec un spool de processus

Ecrire en C une version parallèle de la commande *find*. On limitera les arguments et les fonctionnalités du find au cas suivant :

```
usage : findS motif repertoireDeDepart
```

Le motif est limité ici, au nom exact de l'entrée recherchée (pas de caractère wildcard ou d'expression régulière).

Chaque répertoire rencontré sera pris en chaque par un processus mais le nombre de processus est limité. La liste des répertoires à traiter par les processus est mise dans un tampon borné. Au départ, il n'y aura que le repertoireDeDepart dans le tampon.

Vous utiliserez un spool (ensemble) de 4 processus de recherche pour faire cette recherche. Un processus de recherche ne traite d'un répertoire et ne descend pas dans les sous-répertoires. L'algorithme est du type producteur-consommateur. Chaque processus de recherche extrait un travail (consommateur) à partir du tampon. Ce travail consiste à lister un répertoire en recherchant le motif (parcours non récursif) ; puis de redemander un travail et ainsi de suite. Les travaux (répertoires à traiter) sont donc mis dans un tampon partagé (implanté par un tableau ou une structure dans un segment partagé). Lorsqu'un processus de recherche rencontre un sous-répertoire, il l'insère (producteur) dans le tampon. Le programme principal crée le tampon, insère le répertoire de départ de la recherche dans ce tampon, puis crée le spool de processus (boucle de fork). La taille du tampon est de 128 lignes de 4096 caractères. Vous pouvez utiliser ce même segment partagé ou un second pour stoker des variables partagées si nécessaire. Les affichages éventuels des résultats, entrées trouvées, sur la sortie standard par les processus de recherche seront protégés en exclusion mutuelle.

Pour ceux qui souhaitent tester leur programme, une arborescence test de fichiers vous est donnée en test DEPART.zip. La recherche devrait donner les résultats suivant.

```
findS charlie DEPART
DEPART/R5/charlie
DEPART/R2/SR2/charlie
```

Vous avez en annexe et à télécharger le code d'un ls récursif en C pour vous en inspirer (parcours d'un répertoire).

Je vous conseille de commencer avec une structure de ce type :

```
struct buffer{
  int nb;
  char values[MAXLIGNE][PATHMAX];
};
struct buffer *  buf;
```

Ce qui permettra de voir le segment partagé comme une structure et par exemple, de faire

```
strcpy(buf->values[0],  "/tmp");
ou inversement
char str[PATHMAX]
strcpy(str,buf->values[0])
```

On pourrait résumer l'algorithme de cette façon : Le programme principal, crée un ensemble de sémaphores, un segment partagé (le buffer), initialise celui ci en mettant le nom du répertoire de départ, puis crée l'ensemble de processus de recherche (fork). Ceux-ci itère dans la suite d'actions suivantes : j'extraie un nom de répertoire du tampon. Je parcours celui ci en testant le nom des entrées rencontrées. Si je rencontre le motif, j'affiche ce path de cette entrée en me mettant en exclusion mutuelle avec d'éventuels autres affichages. Et si cette entrée est un répertoire je la dépose dans le tampon.

version 2 complète : l'arrêt du programme

Le programme doit s'arrêter "proprement". Il doit s'arrêter lorsque le tampon est vide et que tous les processus de recherche ont traité leur travail et qu'ils sont bloqués en attente de travail.

Vous pouvez éventuellement utiliser la fonction. `int nb=semctl(semid,numsem,GETNCNT,0)` pour connaître le nombre de processus en attente sur le sémaphore numéro *numsem*, même si ce n'est pas ici la meilleure solution.

Annexe

Pour vous aider, voici le code en C d'un ls récursif (*ls -R).*

```
#include <stdio.h>
#include <string.h>
#include <dirent.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
void scanDir  (char * dir ) {
  DIR * nom_directorie ;
  struct dirent * fichier ;
  if ( nom_directorie = opendir(dir) ){
    while (()fichier = readdir(nom_directorie))!=NULL) {
      char fic[PATH_MAX];
```

```
      struct stat etat ;
      sprintf(fic,"%s/%s",dir,fichier->d_name);
      printf("%s\n",fic);
      if ( stat(fic,&etat) != -1 ) {
        if ((etat.st_mode & S_IFMT) == S_IFDIR)
          if (strcmp(fichier->d_name,".") && strcmp(fichier->d_name,"..")) {
            scanDir(fic);
          }
      }
    }
    closedir(nom_directorie);
  }
}
int main ( int argc , char **argv ) {
  if ( argc != 1 )
    scanDir(argv[1]);
}
```