ENSEIRB MATMECA BORDEAUX

Année 2011-2012 — 1ère session

PROGRAMMATION SYSTÈME
PG 204
DENIS BARTHOU

Filière : Informatique — Année : 1 — Semestre :1

Date de l'examen : 11/01/2011 — Durée de l'examen : 2h

Documents autorisés ☒ sans document ☐
Calculatrice autorisée ☒ non autorisée ☐

Autre :

SUJET

1 Descripteurs de fichier

Ecrivez un programme qui écrit vers une FIFO et vers un fichier tout ce qu'il lit de l'entrée standard. La FIFO est nommée "fifo" et on supposera qu'elle existe déjà (ne pas la créer). Le fichier s'appellera "output". On écrira le programme en utilisant les fonctions `open`, `read`, `write` et `close`. On ne fera aucune gestion des erreurs et on omettra les includes. Attention, votre code ne doit pas excéder 15 lignes !

2 Signaux

Ecrivez un programme `run` qui prend le nom d'un fichier et le nom d'un exécutable `exec` en ligne de commande et lance cet exécutable en ayant redirigé sa sortie standard vers le fichier. L'exécutable `exec` ne prendra pas d'argument. Dès que l'exécutable `exec` termine son exécution, le programme `run` le relance. Si une exécution de `exec` se termine anormalement, le programme `run` affiche sur la sortie standard un message avec la raison de sa terminaison, avant de le relancer. On utilisera entre autres les fonctions `fork`, `waitpid`, `dup` et `execvp`.
Pour simplifier l'écriture du programme, on ne fera aucune gestion des erreurs et on supposera qu'il y a bien toujours 2 paramètres en ligne de commande (le fichier et le nom de l'exécutable). On omettra par ailleurs les includes.

3 Programmation muti-threads

Considérez les deux fonctions suivantes:

```
void multiplie() {
  int i,j,k;
  for (i=0; i<N; i++)
    for (k=0; k<N; k++)
      for (j=0; j<N; j++)
        C[i][j] += A[i][k] * B[k][j] ;
}
void somme() {
  int i,j;
  for (i=0; i<N; i++)
    for (j=0; j<N; j++)
      D[i][j] = C[i][j] + B[i][j];
}
```

La première fonction fait un produit matriciel, la seconde fait une somme de matrices. Les variables `A,B,C,D` sont des variables globales.

1. Dans le cas où un thread exécute la fonction `multiplie` et un second thread exécute la fonction `somme`, expliquez en détail le déroulement d'une exécution pouvant mener à un calcul dont le résultat ne correspond à aucune exécution séquentielle.
2. Identifiez la section critique dans ces deux fonctions.
3. Modifiez le code de `somme` et celui de `multiplie` avec des `mutex` pour que le résultat corresponde à un résultat qu'on pourrait obtenir par une exécution séquentielle. On n'écrira pas le code de création et d'initialisation de ces mutex.
4. On considère maintenant la fonction suivante:

```
void multipliei(int i) {
  int j,k;
  for (k=0; k<N; k++)
    for (j=0; j<N; j++)
      C[i][j] += A[i][k] * B[k][j];
}
```

effectuant la mise à jour de la ligne i de C. Un seul thread exécute la fonction `somme`, et un nombre quelconque de threads exécute `sommei`. On veut que l'exécution multithread soit telle que soit un thread exécute `multiplie`, soit un nombre quelconque de threads exécutent pour des i différents, `multipliei`.

Modifier le code de ces fonctions pour garantir ce fonctionnement. On n'écrira pas la création et l'initialisation d'éventuels mutex.

4 Mémoire, threads et processus

On considère le programme suivant:

```
int a[10];
void f(int n) {
  int b;
  char *c=malloc(sizeof(char)*n);
  ...
}
```

1. Dans le cas où plusieurs processus exécutent la fonction `f` (suite à un `fork`), est-ce que les adresses de `a,b,c,f` sont les mêmes entre processus ? Est-ce que ces variables sont partagées ? Expliquez.
2. Modifiez ce code pour que toute modification faite sur `c` par un des processus soit visible de tous les autres. On omettra les codes d'erreur et les includes. Votre code ne doit pas dépasser 15 lignes.
3. Dans le cas où plusieurs threads exécutent la fonction `f`, est-ce que les adresses de `a,b,c,f` sont les mêmes entre processus ? Est-ce que ces variables sont partagées ? Expliquez.
4. Est-ce que les adresses de `a,b,c,f` sont différentes dans les différents threads ou non ? Justifiez.