Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Licence 3ème année

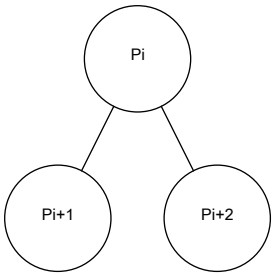
UE Parallélisme I

janvier 2013

Durée : 2h — Documents autorisés

Signaux & tubes – (8 points)

1 – 8pts On veut écrire un programme pour créer une structure arborescente de processus communicant.



Un processus P_i va :

1. créer deux processus enfants P_{i+1} et P_{i+2} ;
2. créer quatre tubes qu'il partagera entre ses deux enfants :
   - ◇ un tube de $P_i \longrightarrow P_{i+1}$;
   - ◇ un tube de $P_i \longrightarrow P_{i+2}$;
   - ◇ un tube de $P_{i+1} \longrightarrow P_i$;
   - ◇ un tube de $P_{i+2} \longrightarrow P_i$;

a. Si l'on veut partager un tube entre un processus et son processus enfant, à quel moment doit-on faire l'appel système `pipe` ?

b. Dans la structure arborescente binaire présentée, est-il possible d'utiliser des signaux différents pour prévenir le processus parent de la fin du travail de P_{i+1} et de P_{i+2} et, si oui, lesquels ?

c. Écrire un « squelette » de programme C permettant de créer la structure aborescente décrite précédemment (deux processus et 4 tubes) qui exécutera les fonctions :
   - ◇ `void travail_enfant_1()`, chargée de faire le travail du processus P_{i+1} ;
   - ◇ `void travail_enfant_2()`, chargée de faire le travail du processus P_{i+2} ;
   - ◇ `void travail_parent()`, chargée de faire le travail du processus P_i ; *Ces trois fonctions sont définies dans le fichier « `travail.c` » de fichier d'entête « `travail.h` » qui utiliseront les tubes mis en place pour communiquer et que vous n'avez pas à écrire.*

   et enverra un signal vers le processus parent P_i :
   - ◇ depuis P_{i+1} après l'exécution de `travail_enfant1`, pour informer P_i de la fin du travail ;
   - ◇ depuis P_{i+2} après l'exécution de `travail_enfant2`, pour informer P_i de la fin du travail ;

d. Est-ce qu'il serait plus intéressant d'utiliser des *threads* plutôt que des signaux pour « surveiller » la fin du travail d'un processus enfant et pourquoi ?

e. Si on voulait réécrire le programme avec des *threads*, avec une thread par fonction de travail, comment faudrait-il implémenter les communications entre les threads pour qu'elles soient réalisées correctement ?

Threads & Sémaphores – (8 points)

2 – 4pts Soient les 4 threads suivantes, s'exécutant chacune **une seule fois** :

```
Thread A {          Thread B {          Thread C {          Thread D {
   P(s1);              P(s1);              P(s2);              P(s2);
   x = x * y;          y = x + y;          y = y + 5;          x = x * 2;
   V(s3);  }           V(s2);  }           V(s3);  }           V(s3);  }
```

Conditions initiales :

⋆ Elles utilisent pour leur synchronisation trois sémaphores `s1`, `s2` et `s3`.

⋆ La valeur initiale de `s1` est 1, celle de `s2` est 0 et celle de `s3` est 0.

a. Quelles sont les différentes valeurs possibles des variables `x` et `y` à partir du moment où toutes les *threads* sont bloquées ou terminées, si `x` est initialisé à 0 et `y` à 1 ?

b. Qu'est-ce que cela change si la valeur initiale de `s1` est 2, celle de `s2` est 0 et celle de `s3` est 1 (`x` reste à 0 et `y` à 1) ?

3 – Écrire un algorithme pour résoudre le problème suivant (vous pouvez utiliser la notation de l'exo 2) :

4pts

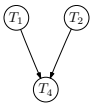
Un programme a été décomposé en 7 threads $\{T_1, T_2, T_3, T_4, T_5, T_6, T_7\}$.
L'analyse du parallélisme pouvant être exploité entre ces différentes threads est la suivante :

- les threads $\{T_2, T_3, T_4, T_6\}$ peuvent être réalisées en parallèle ;
- les threads $\{T_2, T_3, T_4\}$ ne peut être exécutées qu'après la thread T_1 ;
- la thread T_5 ne peut être faite qu'après les threads $\{T_2, T_3, T_4\}$;
- la thread T_7 ne peut s'exécuter qu'après les threads T_5 et T_6 ;
- la thread T_6 ne peut s'exécuter qu'après la thread T_1.

Questions :

a. Donnez le graphe de précédence du travail de ces 7 threads ;

Vous utiliserez la notation suivante :



$\Longrightarrow$ *Ce graphe de précédence correspond à la description suivante :*

- $\{T_1, T_2\}$ *peuvent être réalisées en parallèle ;*
- T_4 *ne peut être faite qu'après les threads* T_1 *et* T_2.

b. Combien de Sémaphores faudra-t-il pour réaliser ce travail ?

c. Donnez l'algorithme qui permet de synchroniser les threads entre elles.

Création de processus avec fork – (4 points)

4 – Soit le programme C suivant :

4pts

```
#include <stdio.h>

void enfant(int parametre)
{  int i=0; int resultat = 0;

   resultat = fork();
   for(i=0; i <2; i++)
   {
      if (resultat)
      {
         parametre = parametre * 2;
         printf("Le parametre=%d\n", parametre);
      }
      else
      {
         parametre = parametre + 5;
         printf("Le parametre=%d\n", parametre);
      }
   }
}

int main()
{  int resultat = 0;
   resultat = fork();

   if (resultat)
   {
      enfant(1);
   }
   else
   {
      enfant(2);
   }
}
```

Que va-t-il afficher lors de son exécution ?