Prof. Yann Thoma

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Programmation concurrente 1 (PCO1)

semestre printemps 2008 - 2009

Contrôle continu 1

9,5

29.04.2009

– Aucune documentation n'est permise, y compris la feuille de vos voisins

Question 1: (10 pts) 10

Citez 3 applications où une réalisation multi-threadée pourrait être bénéfique. Pour chaque exemple, expliquez brièvement comment décomposer le programme.

- Un serveur FTP :
  Le serveur attends les connexions clients sur le processus principal puis un nouveau thread est créé à chaque nouveau client.

- Une application cartographique :
  Chaque région d'une carte peut être "représentée" par un thread pour appliquer un traitement. (les résultats des requêtes afflueront plus rapidement).

- Une application de gestion des trains :
  Un convoi est "représenté" par un thread, ceci permettant aisément de synchroniser les tronçons communs. Pour chaque convoi il y a un nouveau thread.

Question 2: (10 pts) 6

```
int nombre;

void *fonctionA(void *arg) {
  nombre=0;
  nombre+=1;
}

void *fonctionB(void *arg) {
  nombre=5;
  nombre+=2;
}
```

Si ces fonctions sont exécutées de manière concurrente, quelles sont les valeurs que peut prendre `nombre` en fin d'exécution. Listez toutes les possibilités.

En fonction des "envies" de l'ordonnanceur, toutes les combinaisons sont possibles :

nombre = 7, 1, 3, 8, 2, 6

Exemples :

nombre = 0 :

nombre = 5;
nombre += 2;
nombre += 1

nombre = 5;
nombre += 2;
nombre = 0;
nombre += 1;

Question 3: (10 pts) 8

Soit le programme suivant :

```
int a,b;
sem_t s1,s2,s3;

void *fonctionA(void *arg) {
  sem_wait(&s1);
  a=a+1;
  sem_wait(&s2);
  sem_wait(&s3);
  b=b+1;
  sem_post(&s3);
  sem_post(&s1);
}

void *fonctionB(void *arg) {
  sem_wait(&s1);
  a=a+1;
  sem_wait(&s3);
  b=b+10;
  sem_post(&s3);
  sem_post(&s2);
  sem_post(&s1);
}

int main (int argc, char *argv[])
{
  pthread_t threadA,threadB;
  sem_init(&s1,0,1); // initialisé à 1
  sem_init(&s2,0,0); // initialisé à 0
  sem_init(&s3,0,1); // initialisé à 1
  pthread_create(&threadA, NULL, fonctionA, NULL);
  pthread_create(&threadB, NULL, fonctionB, NULL);
  pthread_join(threadA,NULL);
  pthread_join(threadB,NULL);
  return 0;
}
```

Quel(s) problème(s) pourrai(en)t être observé(s) durant son exécution ?

Proposez un moyen de le(s) résoudre, en modifiant légèrement le code tout en gardant la sémantique du programme.

interblocage

On peut se trouver dans à un problème de famine. Car il peut arriver un moment où s1 et s2 se retrouvent bloqués tout les deux.
Si la fonction A arrive en 1er sur son sem_wait(&s2),
s1 est bloqué, s2 aussi et du coup la fonction B est bloquée également.
S2 sert à régler un problème de priorité.
Pour résoudre ce problème :
dans la fonction A : déplacer "sem_wait(&s2);" au tout début de la fonction. (c.f. cé-dessus) ~ plutôt déplacer sem_post(&s7);

Question 4: (20 pts) 17

Nous désirons réaliser le contrôle de l'accès à un pont. Une voie unique mène au pont, et les véhicules restent dans la file sans possibilité de dépassement. Les véhicules ont un poids représenté par un nombre entier, et le pont ne peut supporter plus de 100 tonnes.

Proposez l'implémentation des fonctions `accesPont(int poids)` et `sortiePont(int poids)`, ainsi que `initialisePont()`, afin que le pont ne s'écroule pas. (placez votre code sur une des pages à disposition).

```
#define NUM_THREADS 100
// déclarations à compléter
// Exemple: sem_t semaphore;

void initialisePont() {
// à compléter
}

void accesPont(int poids) {
// à compléter
}

void sortiePont(int poids) {
// à compléter
}

void *vehicule(void *arg) {
  int poids=(int)arg;
  while(true) {
    accesPont(poids);
    usleep(1000);
    sortiePont(poids);
    usleep(1000);
  }
}

int main (int argc, char *argv[])
{
  pthread_t threads[NUM_THREADS];
  int t;
  initialisePont();
  for(t=0; t<NUM_THREADS; t++)
    pthread_create(&threads[t], NULL, vehicule, (void *)(t+1));
  for(t=0; t<NUM_THREADS_LECTEUR; t++)
    pthread_join(threads[t],NULL);
  pthread_exit(NULL);
}
```

```
void initialisePort()
{
    poidsMax = 100;          // Tonnes
    chargeActuelle = 0;      // Charge actuelle du port
    sem_init(&semaphore, 0, 1); // Init à 1
}

// Déclarations
int poidsMax, chargeActuelle;
sem_t semaphore;

void accesPort(int poids)
{ while
  if (chargeActuelle + poids >= poidsMax) // On Bloque le port
      sem_wait(&semaphore);

  chargeActuelle += poids;          // Enregistre la charge
}

void sortiePort(int poids)
{
    sem_post(&semaphore);           // Libère le port si Bloqué
    chargeActuelle -= poids;        // Enregistre la charge
}

//
```

problème d'exclusion mutuelle !