SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

PC3R - TD1: Modèle Préemptif

Equipe Enseignante PC2R

31/01/2020

Lien vers les ressources de l'UE: `https://www-master.ufr-info-p6.jussieu.fr/2019/PC2R`

1 Sémantique d'entrelacement

1.1 Entrelacements simples

1. Soient les deux processus (théoriques) suivants, executés dans un environnement où toutes les variables sont initialisées à 0: $[(x := x + 1; x := x + 1)||x := 2 * x]$
   (a) (**Cours**)Donner la sémantique d'entrelacement de ces deux processus *si l'affectation est atomique.*
   (b) Donner les valeurs possibles x apres terminaison des deux processus *si l'affectation est atomique.*
   (c) Donner la sémantique d'entrelacement de ces deux processus *si l'affectation n'est pas atomique*
   (d) Donner les valeurs possibles x apres terminaison des deux processus *si l'affectation n'est pas atomique.*
2. Mêmes questions pour: $[x := x + 1; x := x + 1||\texttt{if}(x == 1)\{x := 2 * x\}]$
3. Mêmes questions pour: $[\texttt{while}(y < 2)\{x = x + 1; y = x\}||x := 2 * x]$

1.2 Combinatoire

Soit un Processus $P_1 = a_1; a_2; ...a_n$ et un processus $P_2 = b_1; b_2; ...b_m$. Chaque a_i ou b_j est une action atomique.

1. Décrire les exécution possibles de $[P_1||P_2]$ en mode non-préemptif; puis en mode préemptif.
2. Donner toutes les exécutions possibles pour `n=2` et `m=2`,
3. Donner nombre d'exécutions possibles dans le cas général.

1.3 *Deadlock Empire*

Le **jeu sérieux** *Deadlock Empire*, propose un exercice de simulation d'entrelacement dans un cadre ludique. Cet exercice peut être traité (à l'aide d'un téléphone ou d'un ordinateur portable) directement sur le site: `https://deadlockempire.github.io/#menu`.

1. Trouver un entrelacement qui amènent les deux threads suivants dans une situation de compétition (executant `sec_critique()` simultanément):

```
int a = 0;
```

```
a = a + 1;
if (a == 1) {
  sec_critique();
}
```

||

```
a = a + 1;
if (a == 1) {
  sec_critique();
}
```

2. Idem pour:

```
bool flag=false;
```

```
while (true) {
  while (flag != false) {
    ;
  }
  flag = true;
  sec_critique();
  flag = false;
}
```

```
while (true) {
  while (flag != false) {
    ;
  }
  flag = true;
  sec_critique();
  flag = false;
}
```

3. Donner un entrelacement qui exécute `echec()`.

```
int first = 0;
int second = 0;
```

```
tache();
first++;
second++;
if (second == 2 && first != 2) {
  echec();
}
```

```
tache();
first++;
second++;
```

4. Donner un entrelacement qui aboutit à une compétition.

```
object mutex;
object mutex2;
object mutex3;
bool flag=false;
```

```
while (true) {
  if (Monitor.TryEnter(mutex)) {
    Monitor.Enter(mutex3);
    Monitor.Enter(mutex);
    sec_critique();
    Monitor.Exit(mutex);
    Monitor.Enter(mutex2);
    flag = false;
    Monitor.Exit(mutex2);
    Monitor.Exit(mutex3);
  } else {
    Monitor.Enter(mutex2);
    flag = true;
    Monitor.Exit(mutex2);
  }
}
```

```
while (true) {
  if (flag) {
    Monitor.Enter(mutex2);
    Monitor.Enter(mutex);
    flag = false;
    sec_critique();
    Monitor.Exit(mutex);
    Monitor.Enter(mutex2);
  } else {
    Monitor.Enter(mutex);
    flag = false;
    Monitor.Exit(mutex);
  }
}
```

1.4 Dîner des Philosophes

1. Donner un modèle simple du dîner des philosophes à 5 participants.
2. Donner une exécution qui aboutit à un *interbloquage*.

2 Modèles de concurrence: *threads*

2.1 [C] Threads vs. Processus

1. Ecrire en `C`, un programme réalisant le comportement suivant:
   - le processus principal crée une variable entière `v` crée un nouveau processus (`fork`),
   - le processus principal modifie `v` puis attends quelques secondes.
   - pendant ce temps, le processus fils attend moins d'une seconde, puis imprime la valeur de `v` sur la sortie standard.
2. Ecrire en `C`, un programme réalisant le comportement suivant:
   - le thread principal crée une variable entière `v` crée un nouveau thread (*POSIX*),
   - le thread principal modifie `v` puis attends la terminaison du thread fils.
   - pendant ce temps, le processus fils attend moins d'une seconde, puis imprime la valeur de `v` sur la sortie standard.
3. Expliciter les différences entre threads et processus.

2.2 [C, Java, Rust] Entrelacement

1. Ecrire en `C`, puis en `Java`, puis en `Rust`, un programme réalisant le comportement suivant:
   - le thread principal crée 5 threads,
   - à chacun des threads créé est assigné un morceau différent de la phrase `"Belle marquise vos beaux yeux me font mourir d'amour"`.
   - chaque thread créé imprime son morceau de phrase sur la sortie standard,
   - le thread principal attend que les cinq thread créés soient terminés avant de terminer lui-même.
2. Quel est le résultat attendu ?

2.3 [C, Java, Rust] Compteur partagé

1. Ecrire en `C`, puis en `Java`, puis en `Rust`, un programme réalisant le comportement suivant:
   - le thread principal crée 10 threads et un *compteur* (entier initialisé à zéro),
   - chaque thread créé copie la valeur du compteur dans une variable locale, puis incrémente cette variable locale, et enfin écrit la valeur du compteur
   - le thread principal attend que les threads créés soient terminés avant d'imprimer sur la sortie standard la valeur du compteur.
2. Quel sont les impressions finales possibles ?
3. Expliquer, pour chaque langage, comment obtenir un compteur "correct".