FACULTÉ
DES SCIENCES
Unité de formation
et de recherche

UNIVERSITE
BRETAGNE
LOIRE

Système

Fabien GARREAU
sur la base du cours d'André Rossi

Université d'Angers

fabien.garreau@univ-angers.fr

L2 MPCIE, 2018–2019

Threads et processus

Un processus peut être constitué de plusieurs threads s'exécutant « simultanément ». Quelles différences y a-t-il entre diviser un processus en threads, et diviser un processus en plusieurs processus ?

- Chaque processus possède sa mémoire virtuelle propre, alors que les threads partagent la même mémoire virtuelle du processus dont ils font partie
- Chaque thread possède une pile d'appel qui lui est propre
- Le basculement d'un thread à un autre est plus rapide que le basculement d'un processus à un autre.

Un thread est aussi appelé **processus léger**.
Alors que les processus (lourds) sont indépendants et en compétition pour les ressources (ils peuvent avoir été écrits par des programmeurs différents), les threads d'un même processus collaborent pour atteindre un objectif commun. Les threads tirent parti des architectures multi-cœur.

Synchronisation de données

La synchronisation de données assure que les données manipulées par les processus ou les threads sont cohérentes.
Lorsqu'on veut paralléliser un programme séquentiel (typiquement un calcul), on doit vérifier que les **conditions de Bernstein** sont satisfaites. Soit P_1 et P_2 deux parties de programme. Les entrées et sorties de P_1 sont notées I_1 et O_1, celles de P_2 I_2 et O_2.
P_1 et P_2 peuvent être exécutées en parallèle si les trois conditions suivantes sont satisfaites :

- $O_1 \cap I_2 = \emptyset$: aucune sortie de P_1 n'est aussi une entrée de P_2. Si tel était le cas, il faudrait terminer P_1 avant de commencer P_2.
- $I_1 \cap O_2 = \emptyset$: aucune entrée de P_1 n'est aussi une sortie de P_2. Si tel était le cas, il faudrait terminer P_2 avant de commencer P_1.
- $O_1 \cap O_2 = \emptyset$: P_1 et P_2 n'ont pas de sortie commune. Si tel était le cas, la sortie commune serait écrasée soit par P_1, soit par P_2.

Synchronisation de données

Utiliser les conditions de Bernstein pour déterminer si la procédure suivante écrite en C++ peut être parallélisée :

```
void calcul(float a, float b, float &c, float &d)
{
c = a + 5*b;
d = a + c;
}
```

On considère les deux instructions d'affectation.

- $O_1 = \{c\}$
- $I_2 = \{a , c\}$

par conséquent, $O_1 \cap I_2 = \{c\}$ et la première condition de Bernstein, $O_1 \cap I_2 = \emptyset$ n'est pas satisfaite.

La procédure `calcul` n'est pas parallèlisable.

Synchronisation de données

```
void produit(float** A, float* X, float* B, int n)
{
int i,j; /* Calcule B = A * X */
for(i = 0; i < n; i++)
   {
   B[i] = 0;
   for(j = 0; j < n; j++)
      B[i] += A[i][j] * X[j];
   }
}
```

Peut-on découper cette procédure (en C) en `n` parties indépendantes définies pour `k` entre 0 et `n - 1` par :

```
void prod(float** A, float* X, float* B, int n, int k)
{
int j; /* Calcule B[k] = A[k][] * X[] */
B[k] = 0;
for(j = 0; j < n; j++)
   B[k] += A[k][j] * X[j];
}
```

Synchronisation de données

Soient `k1` et `k2` deux entiers distincts dans l'ensemble $\{0, \ldots, n - 1\}$. On teste les trois conditions de Bernstein pour `prod(A,X,B,n,k1)` et `prod(A,X,B,n,k2)` :

1. La sortie de `prod(A,X,B,n,k1)`, `B[k1]` n'est pas contenue dans les entrées de `prod(A,X,B,n,k2)` qui sont la ligne `k2` de `A` et `X`
2. La sortie de `prod(A,X,B,n,k2)`, `B[k2]` n'est pas contenue dans les entrées de `prod(A,X,B,n,k1)` qui sont la ligne `k1` de `A` et `X`
3. Les sorties de `prod(A,X,B,n,k1)` et de `prod(A,X,B,n,k2)` ont une intersection vide : $B[k1] \cap B[k2] = \emptyset$

Il en résulte que la procédure `produit` peut être découpée en `n` threads indépendants ayant pour code `prod(A,X,B,n,k)` avec une valeur unique de `k` dans $\{0, \ldots, n - 1\}$ pour chaque thread.