500 Synchronisation

INF3173
Principes des systèmes d'exploitation

Jean Privat

Université du Québec à Montréal

Hiver 2021

Systèmes concurrents

- Des éléments logiciels (voire matériels)
- Sont capables de s'exécuter en « **même temps** »
- Indépendamment du **moment** ou de l'**ordre** de leur exécution
- **Sans tout briser**

Questions

- Ça veut dire quoi en « même temps » ?
- Quel rapport avec les systèmes d'exploitation ?

En même temps ?

Concurrence

Un élément logiciel s'exécute avant que les autres finissent

L'**ordre** des exécutions de chacun est variable

- Changements de contexte et politiques d'ordonnancements
- Interruptions matérielles
- Signaux logiciels
- Programmation événementielle
- Invocation de sous-programmes (en tirant *vraiment* l'élastique)

Parallélisme

Exécution physiquement au même **moment**

- Architectures multiprocesseurs et multicœurs
- Voire systèmes distribués

  Mais rendu là on a d'autres difficultés en plus

Exemple: incrément

- Suspects: Deux threads t1 et t2 d'un même processus
- Victime: Une variable globale `i` partagée
- Code du crime, exécuté par les deux threads:

```
i++;
```

Où est le problème?

Exemple: incrément

- Suspects: Deux threads t1 et t2 d'un même processus
- Victime: Une variable globale `i` partagée
- Code du crime, exécuté par les deux threads:

```
i++;
```

Où est le problème?

```
; assembleur x86    ; pep8       ; Français
movq i(%rip), %eax  ; LDA  i,d   ; Charge i dans A
addq $1, %eax       ; ADDA 1,i   ; Incrémente A
movq %eax, i(%rip)  ; STA  i,d   ; Sauve A dans i
```

On a pas de problème !

Thread 1	Thread 2	i	A(t1)	A(t2)
		0	?	?
LDA i,d				
ADDA 1,i				
STA i,d				
	LDA i,d			
	ADDA 1,i			
	STA i,d			

- `i` commence à 0
- Chacun des deux threads incrémente `i`
- `i` vaut combien à la fin ?
- Alors ?

On a pas de problème !

Thread 1	Thread 2	i	A(t1)	A(t2)
		0	?	?
LDA i,d		0	**0**	?
ADDA 1,i		0	**1**	?
STA i,d		**1**	1	?
	LDA i,d	1	1	**1**
	ADDA 1,i	1	1	**2**
	STA i,d	**2**	1	2

- `i` commence à 0
- Chacun des deux threads incrémente `i`
- `i` vaut 2 à la fin
- Voilà !

On a un problème !

Thread 1	Thread 2	i	A(t1)	A(t2)
		0	?	?
LDA i,d				
ADDA 1,i				
	LDA i,d			
	ADDA 1,i			
	STA i,d			
STA i,d				

- `i` commence à 0
- Chacun des deux threads incrémente `i`
- `i` vaut combien à la fin ?
- Alors ?

On a un problème !

Thread 1	Thread 2	i	A(t1)	A(t2)
		0	?	?
LDA i,d		0	**0**	?
ADDA 1,i		0	**1**	?
	LDA i,d	0	1	**0**
	ADDA 1,i	0	1	**1**
	STA i,d	**1**	1	1
STA i,d		**1**	1	1

- `i` commence à 0
- Chacun des deux threads incrémente `i`
- `i` vaut 1 à la fin
- Aïe !

Instructions machines

Concurrence vs parallélisme

- Monoprocesseur
  Problème quand changement de contexte au mauvais moment
- Multiprocesseur
  Probabilité de problème bien plus grande

$\rightarrow$ Débogage difficile : syndrome « chez moi ça marche »

Architectures matérielles

`i++` peut être **une seule** instruction « `addq $1, i(%rip)` »

- Monoprocesseur: Plus de vraiment de problème
  - Changement de contexte *avant* ou *après* l'instruction
- Multiprocesseur: Toujours problème de concurrence
  - 1 instruction, mais plusieurs cycles
  - RAM partagée
  - Caches et cohérence mémoire (INF4170)

Quel rapport avec les systèmes d'exploitation ?

Traitement d'événements logiciels et matériels

- Des évènements fondamentalement **imprévisibles**
- Interruptions matérielles
- Appels système de processus (si vrai parallélisme)

Performance des systèmes d'exploitation

Exploitation des possibilités de concurrence et parallélisme

- Traitements parallèles internes : threads système
- Préemption système : les noyaux modernes sont préemptifs
- Une approche « un seul processus en appel système à la fois » fonctionne, mais est très limitante côté performance

Centre de service

- Offre de **mécanismes** de **synchronisation** pour les processus

Classification des programmes concurrents

Disjointe

- Pas d'interaction entre entités logicielles

$\rightarrow$ Facile mais ça n'arrive pas souvent

Compétitive

- Des ressources partagées existent
- On veut s'assurer de leur disponibilité et cohérence

$\rightarrow$ C'est un travail pour le système d'exploitation

Coopérative

- Des éléments logiciels coopèrent
- La concurrence fait partie du programme
- C'est des modèles de programmation spécifiques

$\rightarrow$ Le système d'exploitation offre des services de synchronisation

$\rightarrow$ Mais il y a aussi des ressources à gérer

Situation de compétition (*race condition*)

- Situation où le **résultat** est **différent**
- Dépendamment du **moment** ou de l'**ordre** d'exécution

→ C'est souvent **problématique**

Résultats différents

- Tous pas forcément **corrects**

→ Bogue, y compris de sécurité (INF600C)

Ordre et moment

- Pas connus ou pas contrôlables
- Car ordonnancement, latence matérielle, événements externes...
- Donc situations difficiles à **reproduire** et à **tester**

→ Indéboguable (*Heisenbogues*)

Problématiques de concurrence partout

- Retrait bancaire

```
if(montant < solde) {
    solde =- montant;
    return montant;
} else {
    return 0; // Solde insuffisant
}
```

- Suppression d'un maillon d'une liste doublement chainée

```
if (current->next != NULL)
  current->next->prev = current->prev;
if (current->prev != NULL)
  current->prev->next = current->next;
```

Question

- Trouver des scénarios problématiques

Problématiques de concurrence partout

- Deux processus parallèles font `fork(2)`
  - Attribuer correctement un PID différent
  - Ne pas corrompre la table des processus
- Deux threads parallèles font `malloc(3)`
  - Attribuer correctement une zone mémoire distincte
  - Ne pas corrompre les structures internes du tas
- Deux processus lisent écrivent en même temps dans un tube
  - Attribuer des octets différents (sans en perdre)
  - Ne pas corrompre les structures internes du tube
- Résoudre un chemin (`path_resolution(7)`)
  - Alors qu'un processus renomme ou déplace des répertoires
- Problèmes théoriques classiques de synchronisation
  - Diner des philosophes
  - Producteurs et consommateurs (file bornée)
  - Coiffeur endormi (file d'attente)
  - Écrivains et lecteurs (accès concurrents en lecture ou écriture)

Besoin d'ordre et de discipline !

- Éviter les accès simultanés qui rendraient le système incohérent
- Garantir une certaine équité
- Maintenir la performance
- Éviter que le système ne se blo