Leçon 14 : Gestion et coordination de multiples fils d'exécution.

Motivation :
- architecture multicoeurs
- partage de données facilité

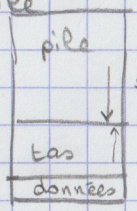
I - Généralités :

A - Définitions :

Def : Un processus est l'ensemble des informations permettant l'exécution d'un programme, notamment un espace d'adressage, ainsi que des registres.

Def : Un espace d'adressage est composé de différentes parties :
- les données : contient les instructions du programme
- le tas : contient les variables globales et celles allouées dynamiquement
- la pile

Ex :



Def : Un fil d'exécution correspond à une exécution d'un processus. Il se compose d'une pile et de registres.

Remarque : Un processus peut avoir plusieurs fils d'exécutions, auxquels cas le tas et les données sont partagées, chaque fil ayant sa propre pile.

B - Changer de fil d'exécution : la commutation :

Un processeur ne peut exécuter qu'un fil d'exécution à la fois. Le procédé par lequel on change le fil en cours d'exécution est la commutation.

La commutation a lieu à différents instants :
- fin du fil d'exécution
- relâchement du processeur (par exemple en attente d'un évènement)
- interruption matérielle (par ex horloge).

Les étapes sont :
- choisir une tâche cible
- sauvegarder les registres
- changer la table des pages si besoin (si le prochain fil ne provient pas du même processus)
- recharger les registres du fil cible.

C - Quels fils exécuter : l'ordonnancement.

L'ordonnanceur est la partie du système responsable de décider quel fil d'exécution s'exécute à un moment donné.

Il doit s'assurer que tous les fils d'exécution aient accès au processeur, éviter les pertes de temps (fil bloqué en exécution) et respecter les priorités de chaque tâche.

Un algorithme d'ordonnancement : Tourniquet (Round-Robin)

Δ : quantum = intervalle de temps

File des tâches à exécuter, commutation à la prochaine tâche au bout d'un temps Δ ou blocage du fil, qui est réenfiler au bout de la file d'exécution.

II - Exclusion mutuelle :

A - Définition :

On considère le cas de fils d'exécution provenant d'un même processus.
Une partie de la mémoire est alors partagée entre ces fils (le tas par ex.)

Ex :

thread 1 exécute :	thread 2 exécute :
if $x > 0$	$x := 0$
⌊ return $1/x$	

Si le thread 2 s'exécute entre les deux instructions du thread 1 : erreur

Def : On appelle section critique (SC) une zone d'un fil dont l'exécution pourrait être influencée involontairement par une exécution concurrente d'un autre fil d'exécution.

Def : Exclusion mutuelle pour toute exécution issue d'une configuration initiale, pour toute configuration de cette exécution, il y a au plus un fil d'exécution en section critique.

B - Outils pour l'exclusion mutuelle

Def : On appelle opération atomique une séquence d'exécution indivisible

Def : Un sémaphore est un objet permettant les opérations atomiques suivantes :
- augmenter (up) : augmenter sa valeur de 1
- diminuer (down) : si la valeur est 1 ou plus, décrémente, sinon reste à 0 et se met en veille sans finir l'exécution de l'appel diminuer pour le moment.

Def : Un mutex est une simplification d'un sémaphore : il n'y a pas de compteur, simplement deux états verrouillé / déverouillé (lock / unlock en anglais)

Ex : Pour éviter des problèmes sur l'exemple précédent :

thread 1 :	thread 2
mutex.lock()	mutex.lock()
if $x > 0$	$x := 0$
return $1/x$	mutex.unlock()
mutex.unlock()	

C. Exemples d'implémentation pour l'exclusion mutuelle

- Boulangerie de Lamport :
DEV 1.

- Algorithme de Peterson : Pour deux threads, notés P (de numéro 0) et Q (de numéro 1)

Principe : . D tableau de fanion indiquant la volonté d'entrer en section critique.
. t : variable de priorité 0 ou 1.

Algo : Entrée (i) :
| D[i] := Vrai
| t := i - 1 # Priorité donnée à l'autre proc.
| Attendre (D[i-1] = Faux ou t = i)

Sortie (i)
| D[i] := Faux

Cet algorithme permet :
- exclusion mutuelle
- pas d'interblocage : si plusieurs fils essaient d'accéder en même temps à la SC, l'un d'entre eux y parvient
- vivacité (absence de famine) un processus demandeur finira par entrer en SC.
(en supposant que la SC elle même n'est pas bloquante)

III- Exemples de problèmes classiques :

A- Problème du rendez-vous

Situation : p threads, chacun en deux phases
La deuxième phase ne peut être exécutée que si tous les fils ont fini la première phase

Solution : Réaliser un bloquage avec un sémaphore

Pseudo code :

```
semaphore rdv := 0
mutex acces
nb_fils := 0
```

```
phase 1
acces.lock()
nb_fils := nb_fils + 1
if nb_fils == p
        rdv.augmenter()
acces.unlock()
```

} section critique

```
rdv.diminuer()  ← bloquage des fils (sauf dernier)
rdv.augmenter() ← chaque fil débloque le suivant

phase 2
```

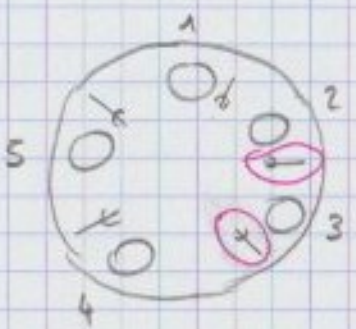
B- Dîner des philosophes :

Idée : Généralisation du problème d'exclusion mutuelle pour l'accès à plusieurs ressources, chacune utilisée de manière exclusive.

Situation : N philosophes, attablé autour d'une table circulaire avec chacun son assiette et N fourchettes.

Un philosophe a besoin de deux fourchettes (celle à sa gauche et à sa droite pour manger).

Ex : N=5



Le philosophe 3 ne peut manger que s'il tient les deux fourchettes entourées

Solution sans interbloquage : DEV 2