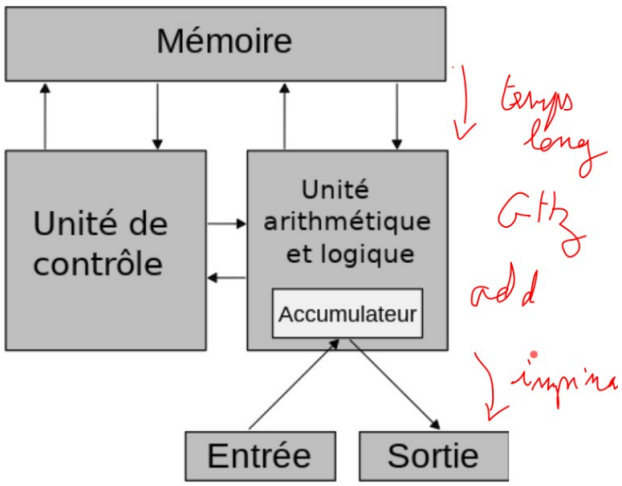
Cours système d'exploitation

Cours du 19/01

Sommaire :
- La synchronisation
- Communication
- Cycle de vie d'un processus (sous linux)

La synchronisation



L'exclusion mutuelle

Il existe différent type de ressources :
- ressource privée (accès local du processus)
- ressource commune (connue de tous, plusieurs processus peuvent la posséder mais pas en même temps)
- ressource partageable (plusieurs processus peuvent la posséder au même instant)
- ressource critique (un seul processus ne peut le posséder à la fois)

Différents type de processus :
Des processus sont des petits programmes qui s'exécutent en mémoire
- processus indépendants (utilisent que des ressources privées)
- processus concurrents ou parallèles (ressources communs avec compétition pour leurs possession)

L'exclusion mutuelle
Il va prendre comme hypothèse que l'on a plusieurs processus d'un côté et des ressources de l'autre.

- prologue :
C'est l'entrée en section critique, c'est un morceau de programme que l'on appellera avant d'utiliser la ressource (le jeton)

- section critique :
C'est l'utilisation de la ressource critique

- l'épilogue :
C'est ce qui gère la sortie de la section critique

Objectif va être de proposer des algorithmes pour les prologue et épilogue réalisant les contraintes suivantes :

- Un seul processus est à la fois en section critique
- si des processus sont en attente devant la section critique et qu'il n'y a aucun processus en section critique, alors il faut garantir un temps d'attente fini
- Le blocage d'un processus hors section critique ne doit pas empêcher un autre processus d'entrer en section critique
- il n'y a pas de processus privilégie

Algorithme 9 : Prologue d'entrée en section critique de l'*attente active*

```
Prologue:
lire P                          /* on teste la variable commune */
si P = 0 alors
    P ← 1                       /* la ressource critique est libre */
sinon
    Aller à Prologue            /* la ressource critique est prise, on
    retourne tester */
finsi
```

Modèle pour la réalisation de l'Exclusion Mutuelle

- L'attente active
- Les verrous
- Les sémaphores

L'attente active

Principe :

- Tester une variable (flag) commune entre les processus
- Mémoriser le droit de passage
- Le processus teste la valeur de variable en boule jusqu'à obtention du droit de passage

Inconvénients **:**

- Consommation inutile de temps CPU juste pour tester une variable
- Réalisation de l'indivisibilité dans le temps délicate
- Sur une machine monoprocesseur : instruction spéciale Test-And-Set
- Sur une machine multiprocesseur : blocage de la mémoire centrale

L'algorithme de Peterson (1981)

- Permet de réaliser l'exclusion mutuelle d'une ressource en permettant une alternance entre les acquéreurs (chacun son tour)
- Synchronise 2 processus avec une seule variable P

Algorithme 15 : Prologue pour l'algorithme de Peterson

Entrées : EntrerRegion(Entier processus) `/* processus = 0 ou 1 */`
S[processus] ← Vrai ; `/* le processus souhaite entrer en section critique */`
P ← processus
tant que *(P = processus)* **et** *(S[1-processus] = Vrai)* **faire**
fintq

Algorithme 16 : Epilogue pour l'algorithme de Peterson

Entrées : SortirRegion(Entier processus) `/* processus = 0 ou 1 */`
S[processus] ← Faux

Les verrous

- Structure composée de
  - . Une variable booléenne commune entre les processus
  - . Une file d'attente de processus
- Valeur de la variable à 0 = ressource critique libre
- Un processus n'obtenant pas le jeton → File d'attente endormi
- Un processus en attente sera réveillé quand il pourra passer

Algorithme 19 : Prologue pour la méthode des verrous, primitive *Verrouiller*

`Verrouiller:`
lire P
si *P = 0*
alors
  P ← 1
sinon
  mettre le processus dans la file d'attente et le mettre dans l'état endormi
finsi

Algorithme 22 : Epilogue pour la méthode des verrous, primitive *Déverrouiller*

`Déverrouiller:`
si *file d'attente non vide*
alors
  sortir un processus de la file et le réveiller
sinon
  P ← 0
finsi

fifo

Les sémaphores

Structure

- Une variable entière
- Une fille d'attente de processus

Des fonctions de manipulation

Algorithme 23 : Création d'un sémaphore : `creation(s,val)`

récupération d'une zone de mémoire
création de la structure de données de nom s
$E(s) \leftarrow$ `val`
pointeur de file d'attente ← nil

Algorithme 24 : Destruction d'un sémaphore

vérification qu'il ne reste aucun processus en attente et en section critique
libération de la mémoire

Prologue et épilogue pour les sémaphores

Algorithme 25 : Prologue pour les sémaphores, primitive P

```
E(s) ← E(s) − 1 ;    /* prise systématique d'un jeton, mémorise
ainsi la demande */
si E(s) < 0 alors
  | le processus est placé dans la file d'attente et s'endort (état bloqué)
finsi
```

Algorithme 26 : Epilogue pour les sémaphores, primitive V

```
E(s) ← E(s) + 1 ;                          /* on rend le jeton */
si E(s) ≤ 0 ;              /* il y a des processus en attente */
alors
  | on réveille un processus de la file d'attente (il redevient actif)
finsi
```

Propriétés

- On ne peut initialiser un sémaphore avec une valeur négative
  . Mais la valeur courante peut devenir négative
- E(s) = E0(s) - nombre d'exécution de P(s) + nombre d'exécution de V(s), avec E0(s) la valeur initiale du sémaphore et E(s) sa valeur courante
- Si E(s) > 0, E(s) représente le nombre de processus pouvant passer
- Si E(s) < 0, | E(s) | représente le nombre de processus en attente
- Si E(s) = 0, aucun processus n'attend et aucun processus ne peut passer

Points importants

- Il est interdit de manipuler un sémaphore (ou un verrou) autrement qu'avec les primitives dédiées ; P/V pour les sémaphore

- Un processus doit exécuter P avant d'entrer en section critique
- Un processus doit exécuter V en sortant de section critique
- Si aucun processus n'est en section critique, alors il ne doit pas y avoir de processus bloqué par le sémaphore

Points auxquels il faut prêter attention

- Quand un processus est tué en section critique, il faut remettre le système en état de fonctionnement, par exemple exécuter V(s) avant de mourir.

Dans le cas contraire, il est possible d'avoir un blocage définitif des processus en attente

- L'écriture et la vérification des algorithmes utilisant des verrous ou des sémaphores n'est pas une chose facile : les primitives sont dispersées dans le texte et sont de bas niveau.

- La stratégie de gestion de la file d'attente est importante car sinon il y a le risque de famine, (i.e qu'un processus est indéfiniment en attente d'une ressource) il faut faire attention à ce que :
  . La mise en file et la sortie de file respectent les mêmes stratégies, par exemple FIFO ou gestion par priorité

. La gestion de la file ne doit pas permettre à un sous-ensemble de processus de bloquer indéfiniment un autre sous-ensemble de processus