Cours IFT1215 - Automne 2008
Introduction aux systèmes informatiques

16. Synchronisation et communication entre processus

Professeur:
Victor Ostromoukhov

16 Synchronisation et communication entre processus

Accès à des ressources communes

- Exemple: compte bancaire dont montant en A sur disque
  - programme pour ajouter 100:
    ```
    lire(N,A)
    N := N+100
    écrire(N,A)
    ```
  - 2 processus exécutent le même programme, le processeur peut exécuter

	processus P1	processus P2	
`LOAD N`	`lire(N,A)`		
		`lire(N,A)`	`LOAD N`
		`N := N+100`	`ADD 100`
		`écrire(N,A)`	`STORE N`
`ADD 100`	`N := N+100`		
`STORE 100`	`écrire(N,A)`		

  - valeur 1000 initiale => 1100 au lieu de 1200 à la fin
  - le compte bancaire A est une ressource critique
  - même chose si N est une variable commune aux deux processus

Les verrous

- Verrou (lock) = objet système avec 2 opérations
  - verrouiller (v) — si verrou disponible => le processus l'obtient
    si indisponible => processus bloqué
  - déverrouiller (v) — s'il y a des processus en attente, un seul l'obtient
    si aucun en attente, verrou rendu disponible
- Opérations systèmes => indivisibles
  - le système garantit l'absence d'interférence entre les exécutions par plusieurs processus

  ➪ notion de primitive

Les sémaphores

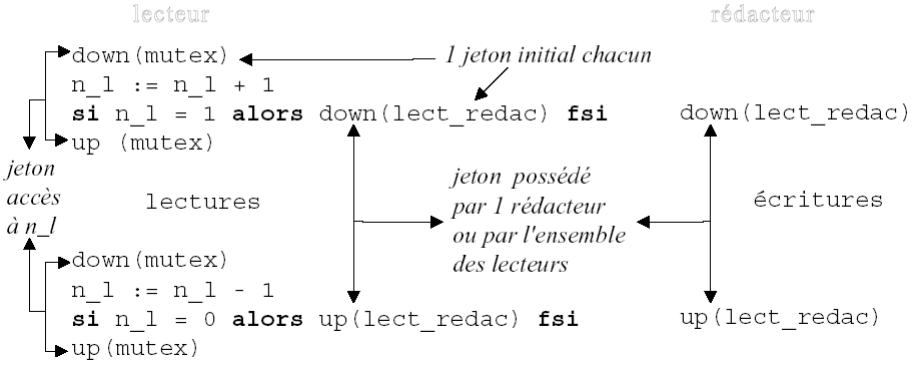
- Objet système plus général que les verrous (Dijkstra)
- Distributeur de jetons
  - nombre de jetons fixe, non renouvelable, rendu après utilisation
  - P (s) = down (s) — s'il y a un jeton disponible, le processus l'obtient
    s'il n'y en a pas, le processus est bloqué
  - V (s) = up (s) — s'il y a des processus en attente, un seul obtient le jeton
    s'il n'y en a pas, le jeton est rendu disponible
- À la création du sémaphore, on fixe le nombre de jetons
  - 1 jeton = verrou
- Ressource à n points d'accès => sémaphore avec n jetons

utilisation →

```
down (s)
accès à la ressource
up (s)
```

Mécanismes plus élaborés (1)

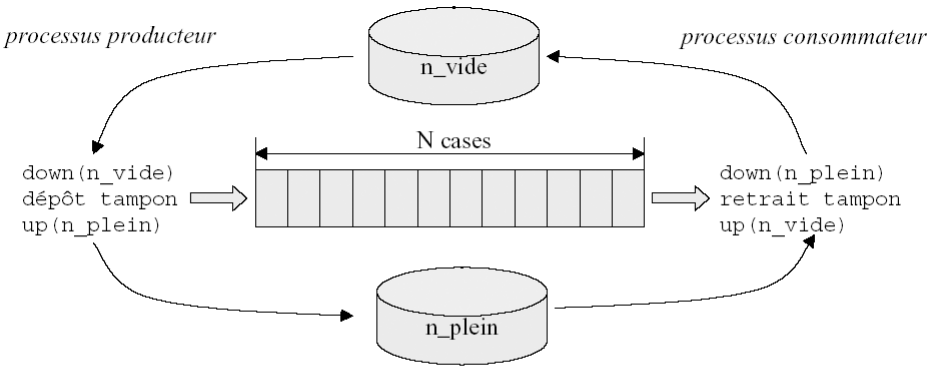
- Politique d'allocation plus complexe
- Mémoire commune + algorithme



Mécanismes plus élaborés (2)

- Besoin: garantie du respect de la règle du jeu
- Définition de politiques par le système
  - gestion de certaines ressources
  - contrôle du respect de la règle du jeu
- Exemple: accès aux fichiers dans un SGF
  - paramètre d'ouverture précisant le type de partage désiré
  - accès exclusif => processus bloqué si déjà ouvert
  - accès partagé => processus bloqué si ouvert en exclusif
  - note: algorithme lecteurs-rédacteurs
- Exemple: accès aux données dans un SGBD
  - règles du jeu définies par l'administrateur de la base et imposées à tous
  - base de données = ressource virtuelle (partage caché)

Schéma producteur-consommateur



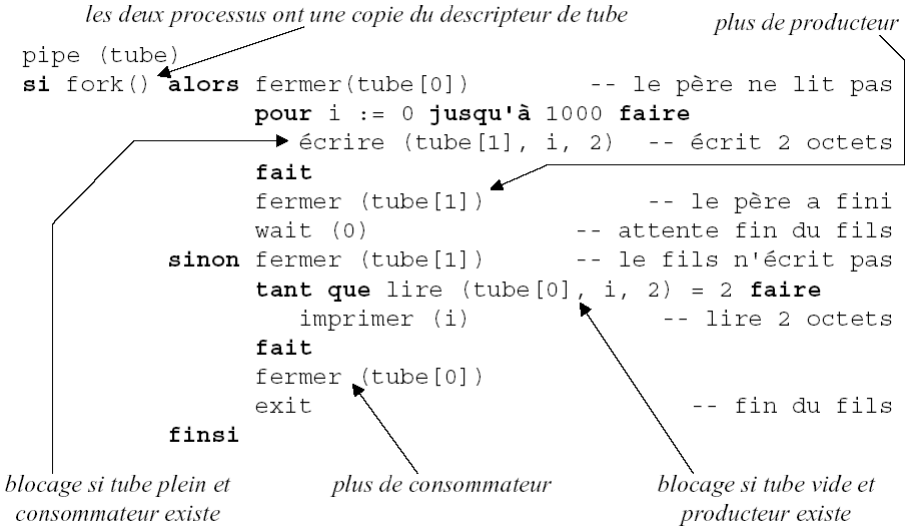
nombre total de jetons (n_plein et n_vide) est N

pas de dépôt si pas de place libre
pas de retrait si pas de message
traitement FIFO => consommateur et producteur ne travaillent pas sur la même case

Les tubes Unix (1)

- Implantation du schéma producteur-consommateur
- Tube = 2 fichiers logiques
  - 1 ouvert en écriture (producteur)
  - 1 ouvert en lecture (consommateur)
- Assimilés à des fichiers séquentiels de caractères
  - lecture n octets
    - bloquante s'il n'y a pas assez d'octets à délivrer
    - sauf si fin de fichier
  - écriture p octets, bloquante si manque de place
  - fin de fichier lorsque plus de processus accèdent au tube en écriture
- Pas de conservation de la structure des messages déposés

Les tubes Unix (2)



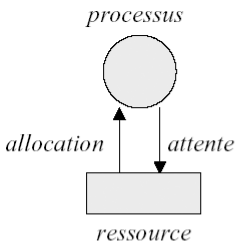
Communication par boîte aux lettres

- Implantation du schéma producteur-consommateur avec conservation de structure des messages
- Désignation
  - anonyme, comme pour les tubes Unix
  - par nom symbolique => table gérée par le système
- Opérations
  - bal := création_bal (nom) => retourne le descripteur de la BAL créée
  - bal := relier_bal (nom) => recherche la BAL et retourne son descripteur
  - envoi (bal, m) => envoi de m dans la BAL, avec ou sans blocage si pleine
  - recevoir (bal, m) => réception de m depuis la BAL, avec/sans blocage
- Variantes: messages typés, filtre à réception,...

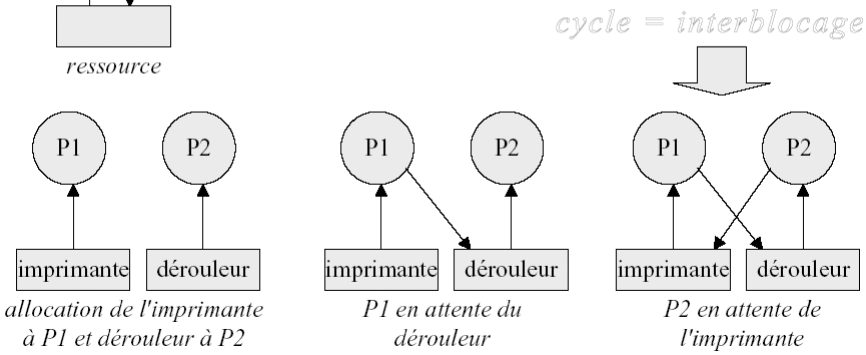
Effet sur les temps de réponse

- Temps de réponse = délai entre envoi commande et réponse
- Composé de:
  - temps processus actif => dépend du programmeur
  - temps processus prêt => dépend de la charge du système
  - temps processus bloqué => attente de ressource
    - entrées-sorties => dépend du programmeur
    - conflit d'accès à une ressource avec d'autres processus
    - limiter ces conflits:
      - multiplier les ressources
      - minimiser les réservations

Notion d'interblocage



P1 demande et obtient l'imprimante
P2 demande et obtient le dérouleur
P1 demande le dérouleur => bloqué
P2 demande l'imprimante => bloqué



allocation de l'imprimante à P1 et dérouleur à P2

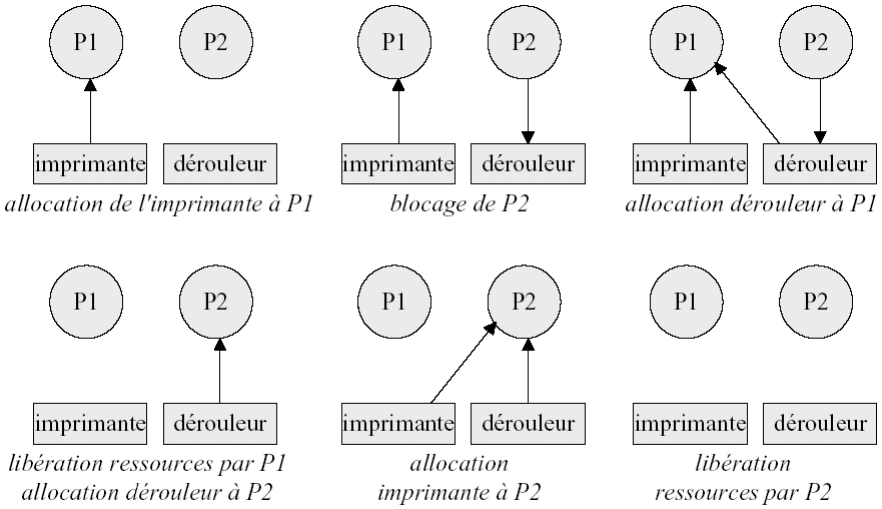
P1 en attente du dérouleur

P2 en attente de l'imprimante

Solutions à l'interblocage

- Politique de l'autruche => *ça n'arrivera pas* ← *premiers systèmes, Unix*
  - si cela arrive, relancer le système!
- Méthode de détection-guérison ← *transactionnel*
  - tuer un processus dans un cycle, jusqu'à suppression des cycles
- Méthode préventive
  - imposer des contraintes sur les demandes pour éviter les cycles
  - ordre des demandes => imprimante avant le dérouleur,...
- Méthode de l'évitement ← *traitement par lot*
  - retarder une allocation possible si elle peut conduire à terme à interblocage
  - nécessite de connaître à l'avance les besoins futurs des processus
- Problème difficile: solution dépend de ce que l'on veut faire

Évitement de l'interblocage



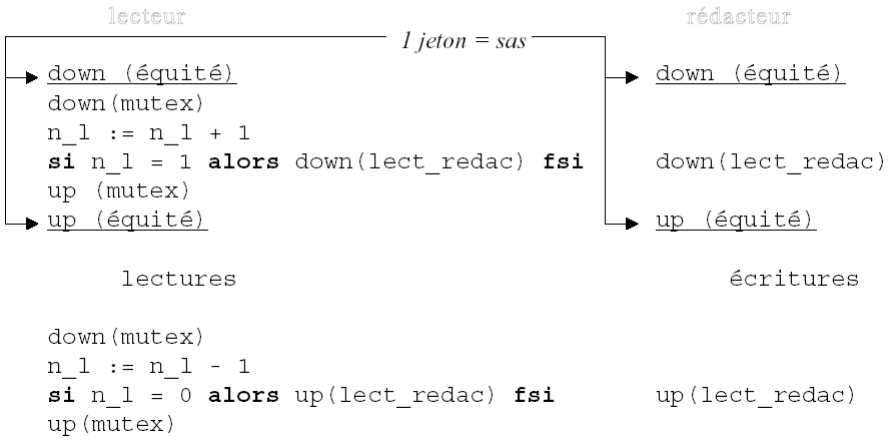
Notion de famine (1)

- Toujours un lecteur => les rédacteurs sont bloqués
- Pas interblocage: les lecteurs arrêtent => OK

```
     lecteur                                          rédacteur

down(mutex)
n_l := n_l + 1
si n_l = 1 alors down(lect_redac) fsi          down(lect_redac)
up (mutex)

      lectures                                        écritures

down(mutex)
n_l := n_l - 1
si n_l = 0 alors up(lect_redac) fsi            up(lect_redac)
up(mutex)
```

Notion de famine (2)

- Conséquence de la politique d'allocation => la changer



Conclusions

- Sémaphores
  - distributeur de jetons non renouvelables et rendu après utilisation
  - primitives demande / restitution, blocage si aucun disponible
  - verrou = sémaphore avec 1 jeton
- Mécanismes plus élaborés avec sémaphores => règle du jeu
- Schéma producteur-consommateur pour la communication
  - tubes Unix, boîtes aux lettres
- Blocage des processus influe sur les temps de réponse
- Interblocage (attente mutuelle) => tuer un processus
- Famine (attente infinie par coalition)
  - changer le comportement des processus, ou changer la politique

Résumé

+ Le mécanisme de verrouillage permet de faire en sorte qu'au plus un seul processus possède le verrou à un instant donné. Les opérations sur les verrous sont des primitives systèmes.

+ Un sémaphore est une généralisation du verrou. Il s'apparente à un distributeur de jetons non renouvelables qui sont rendus après utilisation. Lorsqu'un processus désire un jeton et qu'aucun n'est disponible, il est mis en attente jusqu'à ce qu'un autre processus rende son jeton.

+ Des mécanismes plus élaborés peuvent être construits à l'aide des sémaphores et de variables communes aux processus. Le respect d'une règle du jeu est souvent imposé par le système en fournissant directement ces mécanismes, comme par exemple l'accès exclusif ou partagé à un fichier.

+ Le schéma producteur-consommateur est un mécanisme général de communication entre processus, dont les tubes Unix sont une implantation fournie par le système. Les boîtes aux lettres sont aussi une implantation de ce schéma qui tient compte de la structure des messages échangés.

+ Le blocage des processus en attente de ressources a des conséquences évidentes sur les temps de réponse, et ceci ne doit pas être négligé.

+ La conséquence du blocage des processus est le risque d'interblocage. Ceci se présente lorsqu'on a un ensemble de processus qui attendent des ressources qui sont toutes possédées par des processus de l'ensemble. Lorsqu'il survient, il est nécessaire de tuer des processus de l'ensemble.

+ Un processus peut également être en famine, lorsqu'il attend une ressource qu'il n'obtient pas parce que les autres processus se coalisent pour l'en empêcher. On l'évite par une allocation équitable.

+ La différence entre la famine et l'interblocage est qu'un changement de comportement des processus permettrait de satisfaire le processus en famine, alors que l'on ne peut sortir de l'interblocage qu'en retirant autoritairement une ressource à l'un des processus.