530 Interblocage

INF3173
Principes des systèmes d'exploitation

Jean Privat

Université du Québec à Montréal

Hiver 2021

Interblocage

Alias

- Deadlock
- Verrou fatal
- Étreinte fatale
- Embrasse mortelle

Principe

- Plusieurs processus* sont bloqués entre eux et ne peuvent progresser

*Sans perte de généralité on utilise « processus », mais ça s'applique pareil aux threads, tâches noyau ou toute autre entité logicielle en cours d'exécution.


BANCO REAL
11455

Ressources

Quelques ressources (au sens large)

- Imprimante
- CPU
- Sémaphore
- Section critique (mutex)

Ressources et interblocages

- Les interblocages découlent de l'allocation des ressources

Ressources

Événements liés aux ressources

- Demander la ressource
- Utiliser la ressource
- Libérer la ressource

Si un processus demande une ressource déjà prise

- Erreur
- Attente
- Attente temporisée

Interblocage

Définition plus formelle

- Un ensemble de processus sont en **interblocage** si chaque processus dans cet ensemble est **en attente** d'un événement que **seulement un autre** processus de ce **même ensemble** peut déclencher — Tanenbaum
- L'événement peut-être est la libération d'une ressource

En cas d'interblocage un processus ne peut

- Ni continuer son exécution
  Car il est bloqué
- Ni débloquer un autre processus
  En libérant une ressource
  Car il est bloqué

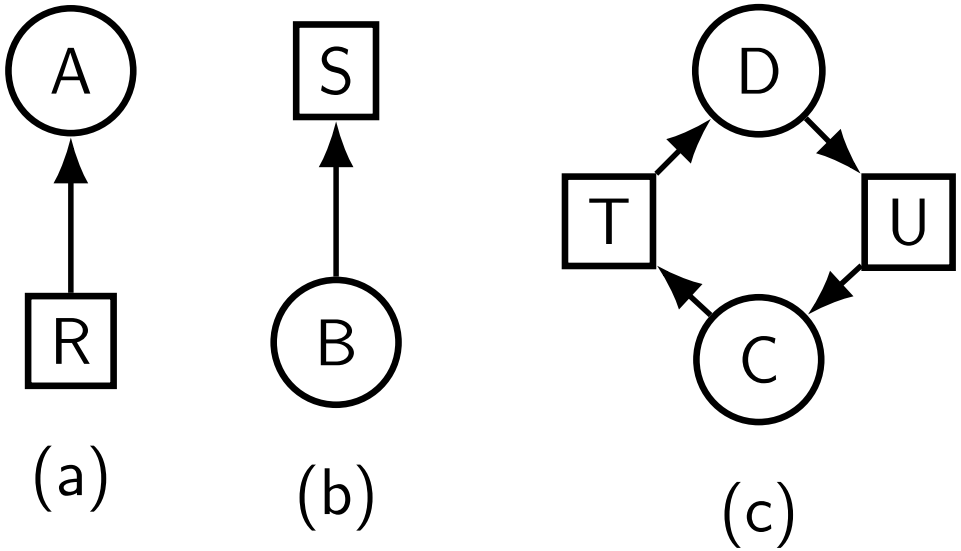
Caractérisation d'un interblocage

Les 4 conditions nécessaires et suffisantes de l'interblocage

- Exclusion mutuelle
  La ressource est soit disponible, soit assignée
- Détention multiple (*hold and wait*)
  Un processus qui détient une ressource peut en demander d'autres
- Pas de réquisition
  Une ressource détenue par un processus doit être libérée par lui
- Attente circulaire
  Il doit y avoir un cycle dans les attentes d'événements

Description des allocations

Graphe biparti orienté



- (a) Ressource R assignée au processus A
- (b) Processus B demande et attend S
- (c) C et D sont en interblocage

Gestion des interblocages

Quatre stratégies

- Ignorer le problème
- Détecter et résoudre
- Prévenir le problème
- Éviter dynamiquement

Ignorer le problème

Principe

- Prétendre que le problème n'existe pas

Raisonnable si

- Interblocages rares
- Autres solutions trop coûteuses/restrictives

Avantage

- Facile à comprendre
- Facile à mettre en œuvre

Détecter et résoudre

Principe

- Vérifier si les processus sont en interblocage et les débloquer

Comment savoir s'il y a interblocage

- Modélisation et analyses de graphe
- Tester

Comment débloquer (sans tout casser)

- Échec du verrouillage
- Retirer de force une ressource
- Restauration d'un état antérieur (*rollback*)
- Éliminer un processus

Prévenir le problème

Principe

- Éliminer une condition de l'interblocage

Exemples

- *Spooling* : seul un processus a la ressource
- Ressources toutes demandées d'un coup
- Permettre la préemption
- Ordonner les ressources (donc les demandes)

$\rightarrow$ Aucun n'est nécessairement faisable

Éviter dynamiquement

Principe

- Forcer l'ordonnanceur à faire le bon choix
- C'est possible via des informations supplémentaires

Idée

- Un état est sûr s'il existe une séquence d'allocations qui permet aux processus d'aller jusqu'au bout
- Exécuter une allocation que si l'état qui en résulte est sûr

Résolutions en pratique

Pas de solution ultime

- Le coût et l'efficacité d'une technique dépendent fondamentalement de la nature des ressources

En pratique,

- Les SE actuels ignorent le problème pour les utilisateurs
- Seuls les SE critiques prennent éventuellement en compte ce genre de problème

Problème cousin - *Livelock*

Jeu de mots sur *deadlock*

- Les processus ne sont pas bloqués
- Mais ne progressent pas non plus

$\rightarrow$ Le CPU est utilisé seulement pour retenter

- Transformer un deadlock en livelock n'est pas un progrès

Problème cousin - Famine

Synonymes

- *Starvation*
- Privation de ressource

Définition

- Un groupe de processus partagent une ressource
- Sans interblocage
- Certains processus n'obtiennent jamais la ressource

$\rightarrow$ Problème de l'attente infinie

Problème cousin - inversion de priorité

Scénario explicatif

- 3 processus de priorité stricte P1 $>$ P2 $>$ P3
- P3 arrive, s'exécute, demande et acquiert une ressource R
- P2 arrive et s'exécute (préempte P3)
- P1 arrive, s'exécute (préempte P2), demande la ressource R
  ... et passe à bloqué
- P2 s'exécute alors à nouveau

Problème

- P2 passe devant tout le monde
- P1 termine dernier, c'était pourtant le plus prioritaire

Solutions

- Revoir la conception: est-ce **normal** que P3 puisse bloquer P1 ?
- Renforcement de la priorité

Renforcement de la priorité

Augmentation temporaire de priorité

- P3 devrait passer avant P2
- Jusqu'à sortir de sa section critique et libérer son mutex
- Pour pouvoir débloquer P1 le plus tôt possible

Nécessite la coopération du système d'exploitation

- Mutex avec priorité statique (`PTHREAD_PRIO_PROTECT`)
  Le processus qui détient le mutex (P3) gagne en priorité
- Héritage de priorité (`PTHREAD_PRIO_INHERIT`)
  Le processus qui attend un mutex (P1) transfert automatiquement sa propre priorité au détenteur du mutex (P3)
- Renforcement aléatoire (exemple Windows)
  Comme P3 détient un mutex, il *peut-être* va gagner un petit *boost* pour *peut-être* sortir de sa section critique.

Dîner des philosophes (Dijkstra)

Données

- 5 philosophes. Chacun pense ou mange (temps inconnu)
- 5 fourchettes
- 5 plats de spaghettis
- 2 fourchettes sont nécessaires pour manger

Problème

Comment faire tourner le système sans bogue ?

- Sans corruption
- Sans famine
- Sans interblocage
- Efficacement