Quatrième partie

Interblocage

Contenu de cette partie

- définition et caractérisation des situations d'interblocage
- protocoles de traitement de l'interblocage
  - préventifs
  - curatifs
- apport déterminant d'une bonne modélisation/formalisation pour la recherche de solutions

Plan

Allocation de ressources multiples

But : gérer la compétition entre activités

- N processus, 1 ressource → protocole d'exclusion mutuelle
- N processus, M ressources → ? ? ? ?

Modèle/protocole ≪ général ≫

- Ressources banalisées, réutilisables, identifiées
- Ressources allouées par un gérant de ressources
- Interface du gérant :
  - **demander** (NbRessources) : {IdRessource}
  - **libérer** ({IdRessource})
- Le gérant :
  - rend les ressources libérées utilisables par d'autres processus
  - libère les ressources détenues, à la terminaison d'un processus.

Garanties sur les réponses aux demandes d'allocation par le gérant

- Vivacité faible (*progression*) :
  si *des* processus déposent des requêtes continûment,
  l'*une* d'entre elles finira par être satisfaite ;
- Vivacité forte (*équité faible*) :
  si un processus dépose sa requête de manière continue,
  elle finira par être satisfaite ;

Négation de la vivacité forte : famine (privation)

Un processus est en famine lorsqu'il attend infiniment longtemps la satisfaction de sa requête (elle n'est jamais satisfaite).

Plan

Le problème

Contexte : allocation de ressources réutilisables

- non réquisitionnables
- non partageables
- en quantités entières et finies
- dont l'usage est indépendant de l'ordre d'allocation

Problème

P_1 demande A puis B,
P_2 demande B puis A
$\rightarrow$ risque d'interblocage :

1. P_1 demande et obtient A
2. P_2 demande et obtient B
3. P_2 demande A
4. P_1 demande B

Interblocage : définition

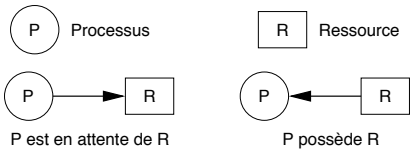
Un ensemble de processus est en interblocage si et seulement si tout processus de l'ensemble est en attente d'une ressource qui ne peut être libérée que par un autre processus de cet ensemble.

Pour l'ensemble de processus considéré :

Interblocage $\equiv$ négation de la vivacité faible (progression)

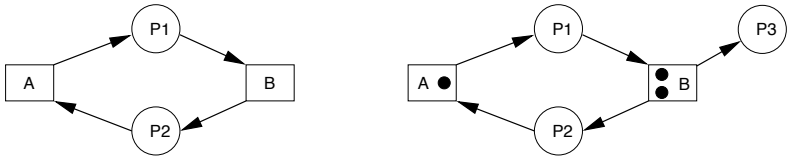
$\rightarrow$ absence de famine (viv. forte) $\Rightarrow$ absence d'interblocage (viv. faible)

Notation : graphe d'allocation



Condition nécessaire à l'interblocage

Attente circulaire (cycle dans le graphe d'allocation)



Solutions

Prévention : empêcher la formation de cycles dans le graphe

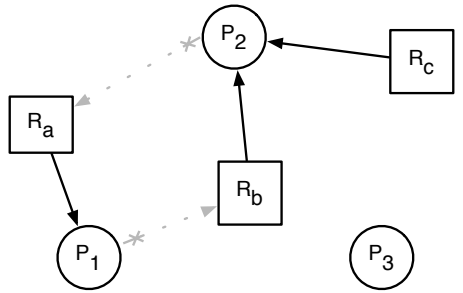
Détection + guérison : détecter l'interblocage, et l'éliminer

Plan

Comment éviter **par construction** la formation de cycles ? (1/4)

Éviter le blocage des processus

→ pas d'attente → pas d'arcs sortant d'un processus



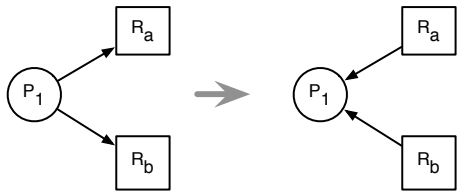
- *Ressources virtuelles* : imprimantes, fichiers
- *Acquisition non bloquante* : le demandeur peut ajuster sa demande si elle ne peut être immédiatement satisfaite

Comment éviter par construction la formation de cycles ? (2/4)

Éviter les demandes fractionnées

Allocation globale : chaque processus demande et obtient en bloc, en une seule fois, toutes les ressources nécessaires

→ une seule demande pour chaque processus
- demande satisfaite → arcs entrants uniquement
- demande non satisfaite → arcs sortants (attente) uniquement



- suppose la connaissance a priori des ressources nécessaires
- sur-allocation et risque de famine

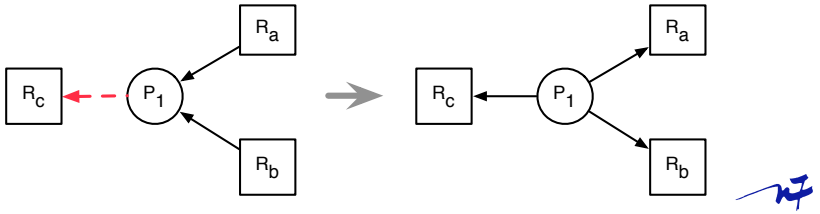
Comment éviter par construction la formation de cycles ? (3/4)

Permettre la réquisition des ressources allouées

→ éliminer/inverser les arcs entrants d'un processus en cas de création d'arcs sortants

Un processus bloqué doit

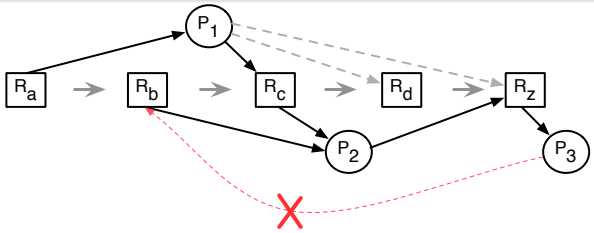
- libérer les ressources qu'il a obtenues
- réobtenir les ressources libérées, avant de pouvoir poursuivre
  → risque de famine



Comment éviter par construction la formation de cycles ? (4/4)

Fixer un ordre global sur les demandes : classes ordonnées

- un ordre est défini sur les ressources
- tout processus doit demander les ressources en suivant cet ordre



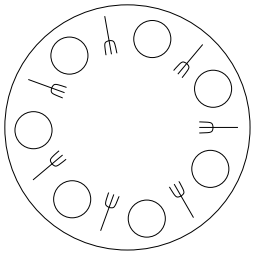
→ pour chaque processus, les chemins du graphe d'allocation vont des ressources inférieures (déjà obtenues) aux supérieures (demandées)

⇒ tout chemin du graphe d'allocation suit l'ordre des ressources

⇒ le graphe d'allocation est sans cycle
(car un cycle est un chemin sur lequel l'ordre des ressources n'est pas respecté)

Exemple : philosophes et interblocage (1/2)

N philosophes sont autour d'une table. Il y a une assiette par philosophe, et une fourchette entre chaque assiette. Pour manger, un philosophe doit utiliser les deux fourchettes adjacentes à son assiette (et celles-là seulement).



Un philosophe peut être :

- penseur : il n'utilise pas de fourchettes ;
- mangeur : il utilise les deux fourchettes adjacentes ; aucun de ses voisins ne peut manger ;
- demandeur : il souhaite manger mais ne dispose pas des deux fourchettes.

Exemple : philosophes et interblocage (2/2)

Risque d'interblocage

Chaque philosophe demande sa fourchette gauche et l'obtient. Puis quand tous ont leur fourchette gauche, chaque philosophe demande sa fourchette droite et se bloque. ⇒ interblocage

Solutions

Allocation globale : chaque philosophe demande simultanément les deux fourchettes.

Non conservation : quand un philosophe essaye de prendre sa seconde fourchette et qu'elle est déjà prise, il relâche la première et se met en attente sur la seconde.

Classes ordonnées : imposer un ordre sur les fourchettes ≡ tous les philosophes prennent d'abord la gauche puis la droite, sauf un qui prend d'abord droite puis gauche.

Esquive

Avant toute allocation, évaluation dynamique du risque (ultérieur) d'interblocage, compte tenu des ressources déjà allouées.

L'algorithme du banquier

- chaque processus annonce le nombre maximum de ressources qu'il est susceptible de demander ;
- l'algorithme maintient le système dans un état fiable, c'est-à-dire tel qu'il existe toujours une possibilité d'éviter l'interblocage dans le pire des scénarios (= celui où chaque processus demande la totalité des ressources annoncées) ;
- lorsque la requête mène à un état non fiable, elle n'est pas traitée, mais est mise en attente (comme si les ressources n'étaient pas disponibles).

Algorithme du banquier : exemple

12 ressources,
3 processus $P_0/P_1/P_2$ annonçant 10/4/9 comme maximum

	max	poss.	dem	
P_0	10	5		
P_1	4	2	+1	oui

$$(5+\mathbf{4}+2\leq 12)\wedge(\mathbf{10}+(0)+2\leq 12)\wedge((0)+(0)+\mathbf{9}\leq 12)$$

P_2	9	2	+1	non

$$(\mathbf{10}+2+3>12)\wedge(5+2+\mathbf{9}>12)\wedge(5+\mathbf{4}+3\leq 12)\wedge(\mathbf{10}+(0)+3>12)\wedge(5+(0)+\mathbf{9}>12))$$

Algorithme du banquier (1/2)

Allocation de Demande ressources au processus IdProc

```
 var Demande, Disponibles : entier = 0,N;
     Annoncées, Allouées : tableau [1..NbProc] de entier;

si Allouées[IdProc]+Demande > Annoncées[IdProc] alors erreur
sinon
  si Demande > Disponible alors <suspendre le processus>
  sinon
    si étatFiable({1..NbProc}, Disponibles - Demande) alors
      Allouées[IdProc] := Allouées[IdProc] + Demande ;
      Disponibles := Disponibles - Demande;
    sinon <suspendre le processus>;
    finsi
  finsi
finsi

fonction étatFiable(demandeurs : ensemble de 1..NbProc,
                    dispo : entier) : booléen {...}
```


Algorithme du banquier (2/2)

fonction étatFiable(demandeurs:ensemble de 1..NbProc, dispo : entier): booléen

```
 var d : 1..NbProc;
     vus, S : ensemble de 1..NbProc := ∅, ∅;
     solution : booléen := (demandeurs = ∅);

début
  répéter
   S := {p∈demandeurs-vus / Annoncées[p]-Allouées[p] <= dispo}
   si S ≠ ∅ alors
      choisir d ∈ S;
      vus := vus ∪ {d};
      solution := étatFiable(demandeurs-{d},
                             dispo+Annoncées[d]-Allouées[d]);
   finsi;
  jusqu'à (S = ∅) ou (solution);
  renvoyer solution;
fin étatFiable;
```

Plan

Détection

- construire le graphe d'allocation
- détecter l'existence d'un cycle

Coûteux → exécution périodique (et non à chaque allocation)

Guérison : Réquisition des ressources allouées à un/des processus interbloqués

- fixer des critères de choix du processus victime (priorités...)
- annulation du travail effectué par le(s) processus victime(s)
  - coûteux (détection + choix + travail perdu + restauration),
  - pas toujours acceptable (systèmes interactifs ou embarqués).

- plus de parallélisme dans l'accès aux ressources qu'avec la prévention.
- la guérison peut être un service en soi (tolérance aux pannes...)
  → Mécanismes de reprise : service de sauvegarde périodique d'états intermédiaires *(points de reprise)*

Plan

- Usuellement : interblocage = inconvénient occasionnel
  → laissé à la charge de l'utilisateur/du programmeur
  - traitement :
    - utilisation de méthodes de prévention simples (classes ordonnées, par exemple)
    - ou détection empirique (délai de garde) et guérison par choix ≪ manuel ≫ des victimes
- Cas particulier :
  systèmes ouverts, (plus ou moins) contraints par le temps
  - systèmes interactifs, multiprocesseurs, systèmes embarqués
  - recherche de méthodes efficaces, prédictibles, ou automatiques
  - compromis/choix à réaliser entre
    - la prévention qui est plus statique, coûteuse et restreint le parallélisme
    - la guérison, qui est moins prédictible, et coûteuse quand les conflits sont fréquents.
  - émergence d'approches sans blocage (→ prévention), sur les architectures multiprocesseurs (mémoire transactionnelle)