Module SEL

Interblocages

- Ressources et interblocages
- Conditions amenant à un interblocage
- Solutions aux interblocages
  - Prévention
  - Evitement
  - Détection et guérison

istic Informatique Electronique

Introduction

- Définition : Ressource
  - Une ressource est un objet (matériel, donnée...), géré par le système et dont le système peut concéder temporairement le droit d'utilisation à un, ou plusieurs processus.
  - Etapes de l'utilisation d'une ressource
    - Demande de la ressource au système d'exploitation → plusieurs issues possibles
      - Acceptation et allocation immédiate
      - Blocage en attente de disponibilité de la ressource
      - Refus
      - Utilisation de la ressource allouée

Introduction

- Etapes de l'utilisation d'une ressource
  - Restitution de la ressource, qui peut prendre deux formes :
    - Libération (abandon volontaire)
    - Réquisition (autoritaire). La réquisition peut poser problème pour certaines ressources
- Remarques :
  - Le processeur est une ressource particulière : pas de demande explicite pour l'utiliser
  - L'allocation de la ressource est à l'initiative du système
  - La mémoire est un autre cas particulier : pas d'attente, refus uniquement

Critères d'un bon allocateur de ressources

- Dysfonctionnements à éviter :
  - Privation (ou famine) : un processus attend indéfiniment une ressource
  - Congestion : des allocations mal contrôlées conduisent à une surcharge du système, et à une dégradation de ses performances
  - Interblocage (deadlock) : des processus se trouvent bloqués mutuellement, et indéfiniment
  - Ici, étude de l'interblocage

Interblocage
Définition et caractérisation

- Définition : Interblocage
  - On dit qu'un ensemble de processus subit un interblocage si chaque processus de l'ensemble est bloqué et attend un événement que seul un autre processus de l'ensemble peut provoquer
  - (Ici, l'événement attendu est l'allocation de la ressource).

Interblocage
Exemple 1

- Système doté d'un disque et d'une imprimante
- Deux processus désirent imprimer un fichier
- On note dem(i) la demande, éventuellement bloquante, d'allocation du périphérique i

processus P1;	**processus** P2;
dem(imp); P11	dem(disque); P21
...	...
dem(disque); P12	dem(imp); P22

  - Suite d'actions P11, P21, P12, P22 conduit à un interblocage
  - Suite d'actions P11, P12 ... ne conduit pas à un interblocage ⇒ ce n'est pas une erreur de programmation

Interblocage Exemple 2

- Blocage provenant d'une erreur de programmation

processus P1;	**processus** P2;
...	...
P(mutex);	P(mutex);
P(s);	V(s);
V(mutex);	V(mutex);

  - Ici, le blocage provient du fait que l'on fait un P (bloquant) dans une section critique

Interblocage Exemple 3

- Deux disques, un imprimante, trois processus P1, P2, P3
- P1 et P2 veulent imprimer un gros fichier, chacun stocké sur un disque différent, P3 veut copier un fichier d'un disque sur l'autre

processus P1;	**processus** P2;	**Processus** P3
dem(d1); P11	dem(imp); P21	dem(d2); P31
dem(imp); P12	dem(d2); P22	dem(d1); P32

  - La suite d'actions P_11, P_21, P_31, ... conduit à un interblocage
  - Après les trois premières allocations, aucun processus n'est bloqué, mais l'exécution ultérieure conduit nécessairement à un

Conditions amenant à un interblocage

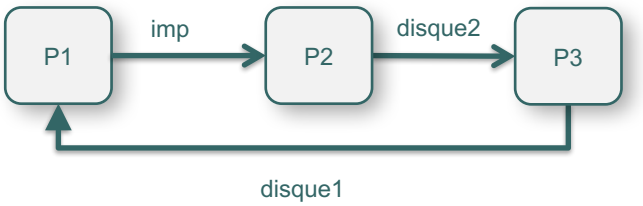
- Exclusion mutuelle : les ressources mises en jeu doivent être partagées et utilisées en exclusion mutuelle
- Attente en détenant des ressources : du aux demandes d'allocation en cours d'exécution et au fait qu'une demande d'allocation peut être bloquante
- Non réquisition : les processus font explicitement les libérations
- Attente circulaire : il existe une suite ordonnée de processus(P_i1, P_i2,...,P_in), telle que P_ik attend une ressource de P_i k+1, pour k ∈[1,n-1] et P_in attend une ressource de P_i1.

Graphe des attentes

- Nœud du graphe : processus
- Arc du graphe P_i → P_j si le processus P_i attend une ressource qui est détenue par P_j
- Identification d'une situation d'interblocage :
  - Le graphe des attentes possède un cycle

Graphe des attentes

- Exemple : graphe des attentes dans l'exemple



Etat sain

- Un interblocage peut devenir inévitable avant que l'on ne constate un blocage effectif de processus (exemple 3)
  ⇒ Modélisation de l'évolution des systèmes pour mettre en évidence ce point
  ⇒ Présentation sur un exemple simple (non sujet à l'interblocage)

Etat sain

- Deux processus P1 et P2 et une ressource partagée R1

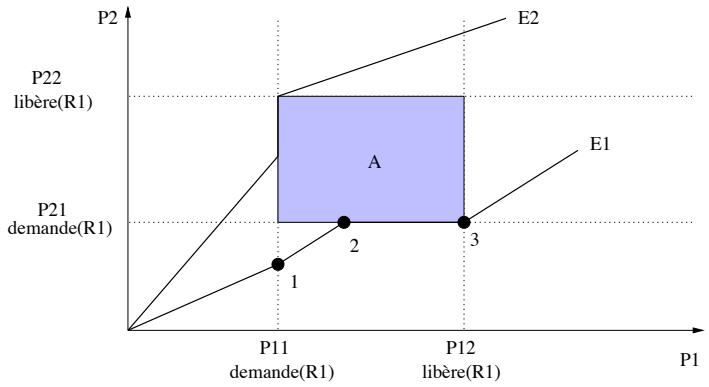
processus P1

```
...
demande(R1); (P11)
...
libère(R1); (P12)
...
```

processus P2

```
...
demande(R1); (P21)
...
libère(R1); (P22)
...
```


Etat sain



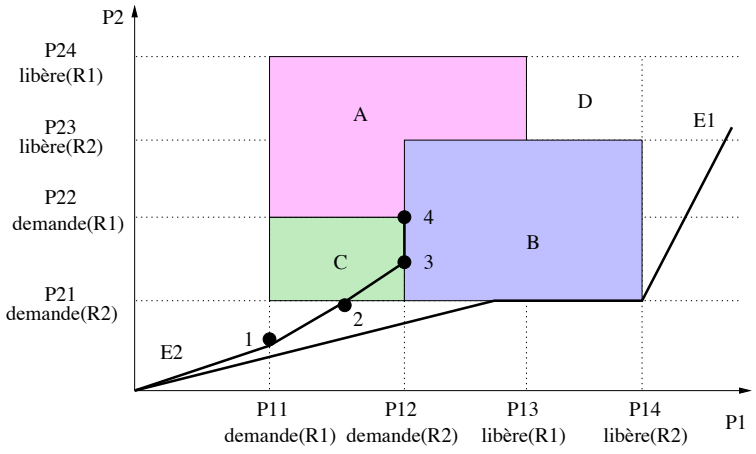
- Zone A interdite (R1 est allouée à P1 et P2)
- Courbe forcément non décroissante en X et Y

Etat sain

- Soient deux processus P1 et P2, utilisant deux ressources exclusives R1 et R2

processus P1;	**processus** P2;
...	...
demande(R1); (P11)	demande(R2); (P21)
demande(R2); (P12)	demande(R1); (P22)
...	...
libère(R1); (P13)	libère(R2); (P23)
libère(R2); (P14)	libère(R1); (P24)
...	...

Etat sain



Etat sain

- Zones A et B correspondent à des zones d'exclusion mutuelle sur R1 et R2 ⇒ zones inaccessibles
  - E1 est une évolution possible, qui ne conduit pas à l'interblocage
  - E2 par contre conduit à un interblocage (à partir du point 2), avec la suite d'événements :
    - allocation de R1 à P1, allocation de R2 à P2
    - demande de R2 par P1, qui se bloque
    - demande de R1 par P2, qui se bloque, il y a interblocage
  - En fait tous les points de la zone C conduisent forcément à l'interblocage

Etat sain

- Définition
  - On dit qu'un état est sain, si à partir de cet état, il existe une stratégie d'allocation qui permet d'éviter l'interblocage
- Sur l'exemple précédent
  - Les zones A et B correspondent à des états non réalisables
  - La zone C correspond à des états non sains
  - La zone D n'est pas accessible à cause de l'impossibilité pour un processus de revenir en arrière
  - Les autres zones correspondent à des états sains

Types de solution à l'interblocage

- Prévention (méthodes pessimistes) : grâce à un contrôle a priori on s'assure qu'il n'y aura pas d'interblocage.
  - Prévention statique : on va rendre l'interblocage impossible. Il s'agit de faire en sorte qu'une des conditions d'interblocage ne soit jamais vérifiées. On limite la liberté des utilisateurs pour rendre l'interblocage impossible
  - Prévention dynamique (évitement) : bien que l'interblocage soit potentiellement possible dans le système, un contrôle effectué lors des allocations permet de faire en sorte qu'il ne se produise jamais
- Détection et guérison (méthode optimiste) : pas de contrôle a priori, mais périodiquement on regarde si le système est interbloqué, si c'est le cas on essaye de briser le blocage avec des méthodes plus ou moins brutales.

Prévention statique

- Caractéristiques des méthodes de prévention statiques
  - Choix dans la structure du système ou de son interface avec l'utilisateur, qui rendent l'interblocage impossible
  - Peuvent être utilisées pour un sous-ensemble des ressources uniquement
    ⇒ possibilités d'interblocage résiduelles devant être gérées par un autre type de méthode

Prévention statique
a. Suppression des ressources partagées

- Elimine la première condition d'interblocage : l'utilisation de ressources en exclusion mutuelle
- Moyen
  - Structuration du système pour qu'un seul processus utilise chaque ressource (structuration en processus, modèle client/serveur)
  - Manipulation de la ressource par un client ⇒ envoi d'une requête au serveur
  - Méthode non généralisable à tout type de ressources (accès à des tables partagées) ⇒ à utiliser en complément d'autres méthodes pour limiter les risques d'interblocage

Prévention statique
b. Réquisition

- Elimine la condition d'interblocage attente en détenant desressources
- Fonctionnement :
  - Blocage en détenant des ressources ⇒ réquisition de toutes les ressources détenues par le demandeur avant de le bloquer
  - Frein : toutes les ressources ne peuvent être réquisitionnées sans dommage.
  - En pratique, peuvent être réquisitionnées facilement les ressources dont l'état peut être sauvegardé
  - Stratégie utilisée dans les bases de données utilisant les transactions. Quand une ressource ne peut pas être allouée, abandon de la transaction en restituant son état initial

Prévention statique
c. Allocation globale

- Elimine également la condition d'interblocage attente en détenant des ressources
- Fonctionnement : on impose à chaque processus de réserver globalement, en une seule fois, toutes ses ressources
  - Impose de grosses contraintes à l'utilisateur qui doit demander le maximum de ressources quelles que soient les circonstances
    ⇒ Mauvaise utilisation des ressources (allocation inutile ou trop précoce des ressources)

Prévention statique
d. Classes ordonnées de ressources

- Elimine la condition d'attente circulaire
- Fonctionnement
  - Les ressources peuvent être demandées en plusieurs fois
  - On impose un ordre sur les demandes de ressources
    - Ressources réparties en classes C_1, C_2,...,C_m
    - Un processus ne peut demander une ressource de la classe C_i, que s'il a déjà obtenu toutes les ressources nécessaire des classes C_j, pour tout j<i

Prévention statique
d. Classes ordonnées de ressources

- Remarques :
  - Contrainte moins forte qu'une réservation en une fois, mais impose quand même de réserver des ressources dont on ne se servira pas tout de suite, à cause de l'ordre de demande des ressources
  - Exemple : utilisation conditionnelle d'une ressource. On peut être amenés à écrire :
    ...
    demande(imprimante);demande(disque);utilisation du disque;...si c alors utilisation imprimante fsi;

Evitement : l'algorithme du banquier

- Dû à du à Dijkstra (1965)
- Permet d'éviter l'interblocage (faire en sorte qu'il ne se produise pas dans un système où il est potentiellement possible)
- Pas de contrainte a priori sur l'utilisation des ressources, hormis celle de fournir au préalable une borne supérieure à ses demandes en ressources

Evitement : l'algorithme du banquier

- Principe du contrôle de l'allocation
  - Lors de chaque demande de ressource, on vérifie si l'état obtenu après allocation est sain (i.e. il sera possible d'éviter l'interblocage à partir de cet état).
    - Blocage si on détecte que l'état ne sera pas sain.
  - Lors d'une libération de ressources : examen des processus bloqués.
    - Réveil des processus bloqués si l'allocation de la ressource libérée laisse le système dans un état sain

Evitement : l'algorithme du banquier

- Ressources regroupées en classes (disques, imprimantes, etc)
- Nombre fixé de ressources de chaque classe
- Structures de données pour l'allocation

```
// n et m représentent le nombre de processus et de classes de ressources
int Exist[m]; // Nombre de ressources existantes
int Dispo[m]; // Nombre de ressources disponibles
int Alloc[n,m]; // Ressources allouées
int Max[n,m]; // Demande maximum
```

Evitement : l'algorithme du banquier

- Définition : Etat réalisable
  - Un état réalisable est tel que :
    $\forall p \in [0,n-1]$ et $\forall r$ \in [0,m-1],
    $0 \leq Alloc[p,r] \leq Max[p,r] \leq Exist[r]$
    $\forall \in [0,m-1], \Sigma_{k=0..n-1} Alloc[k,r] \leq Exist[r]$
    $\forall \in [0,m-1], Dispo[r] = Exist[r] \Sigma_{k=0..n-1} Alloc[k,r]$

Algorithme du banquier

- Principe de la détection d'un état réalisable sain :
  - Vérifier qu'il existe un ordre d'exécution ne conduisant pas à l'interblocage
  - Basé sur les demandes maximales des processus
  - Construction d'une suite ordonnée de processus P_i1,P_i2,..., P_in telle que l'on puisse :
    - satisfaire les demandes maximales de P_i1, avec les ressources disponibles et celles déjà allouées à P_i1
    - satisfaire les demandes maximales de P_i2, avec les ressources disponibles plus celle de P_i1 et celles déjà allouées à P_i2, (revient à considérer que P_i1 s'est terminé et a libéré toutes ses ressources)...

Algorithme du banquier

- Définition : Suite saine
  - Une suite ordonnée de processus P_i1,P_i2,..., P_ik est une suite saine si et seulement si $\forall r \in [0,m-1]$, $\forall ik \in i1,...,iq$, $Max[ik,r] \leq Dispo[r] + \Sigma_{ij \leq ik} Alloc[ij,r]$
- Définition : Etat sain
  - Un état est sain si et seulement si :
    - Il est réalisable,
    - Il existe une suite saine P_i1, P_i2,..., P_in contenant tous les processus

Algorithme du banquier Exemple

- Trois processus P0, P1, P2, trois classes de ressources C0, C1, C2
  Situation initiale :
  Exist = (3,4,2) Dispo = (1,2,1)

Max =		Alloc =	
	2 2 2		1 1 0
	1 2 2		0 1 1
	2 3 2		1 0 0

- Besoins maximaux non satisfaits de chaque processus
  P_0 : (1, 1, 2)
  P_1 : (1, 1, 1)
  P_2 : (1, 3, 2)

Algorithme du banquier
Exemple

- Les demandes restantes de P_0 et P_2 ne peuvent être satisfaites avec ce qui est disponible
- Par contre, on peut satisfaire les besoins maximaux de P_1 avec :
  - Ce que possède P_1 : (0, 1, 1)
  - Ce qui est disponible : (1, 2, 1)
  - Total (1, 3, 2) ≥ (1, 2, 2)
- On peut ensuite satisfaire les besoins maximaux de P_0 avec :
  - Ce qui est disponible : (1, 2, 1)
  - Ce que possède P_1 : (0, 1, 1)
  - Ce que possède P_0 : (1, 1, 0)
  - Total : (2, 4, 2) ≥ (2, 2, 2)

Algorithme du banquier
Exemple

- On peut ensuite satisfaire les besoins maximaux de P_2 avec :
  - Ce qui est disponible : (1, 2, 1)
  - Ce que possède P_2 : (1, 0, 0)
  - Ce que possède P_1 : (0, 1, 1)
  - Ce que possède P_0 : (1, 1, 0)
  - Total: (3, 4, 2) ≥ (2, 3, 2)
- La suite P_1, P_0, P_2 est saine. L'état est donc sain, le système n'est pas interbloqué

Algorithme du banquier Exemple

- Demande de dj ressources de la classe j par Pi :

```
if (Alloc[i,j]+dj > Max[i,j])
    erreur "demande trop forte »
else if (dj > Dispo[j])
    <mettre Pi en attente> // Pas assez de ressource
else
    Alloc[i,j] = Alloc[i,j]+dj;
    Dispo[j] = Dispo[j]-dj;
    if (nouvel état sain)
            <effectuer l'allocation>
    else
            Alloc[i,j] = Alloc[i,j]-dj;Dispo[j] = Dispo[j]+dj;
            mettre Pi en attente // Allocation dangereuse
```


Algorithme du banquier

- Situations de blocage du processus demandeur :
  - Pas assez de ressources disponibles pour satisfaire la demande
  - Assez de ressources disponibles mais allocation dangereuse, car elle peut conduire à l'interblocage

Algorithme du banquier
Algorithme de test pour savoir si on est dans un état sain

- Recherche d'une permutation de processus parmi toutes les permutations possibles
- Complexité a priori : O(n!) opérations.
  - Peut être réduit à $O(n^2)$ en exploitant deux props. :
    - Si l'état d'allocation est sain, et si la suite partielle S=P_i1,...,P_ik, k < n est saine, alors S est le début d'une suite saine contenant tous les processus ⇒ pas besoin de retour arrière
    - Si un état sain est transformé par une allocation à un processus P_k, et si on peut construire une suite partielle saine contenant P_k, alors le nouvel état est sain ⇒ pas nécessaire de construire une suite complète (on s'arrête dès que la suite contient le processus demandant la ressource)

Détection et guérison de l'interblocage

- Principe
  - On se préoccupe pas a priori de l'interblocage, on le laisse s'installer
  - Exécution périodique d'un algorithme de détection d'interblocage
    - Lorsqu'un interblocage est détecté, on prend les mesures qui s'imposent (réquisition de ressources, arrêt des processus)

Détection de l'interblocage

- Etat de départ :
  - Certaines ressources sont déjà allouées, des processus sont en attente de ressources
- Algorithme utilisable
  - Similaire à l'algorithme du banquier, en travaillant sur les ressources demandées Req à un instant donné au lieu des ressources maximales Max
  - Absence d'interblocage quand on est dans un état sain, i.e. on est capable d'ordonner tous les processus dans une suiteP_i1, P_i2,... de telle façon que :on peut satisfaire entièrement les demandes de P_i1 avec les ressources disponibles on peut satisfaire les demandes de P_i2 avec les ressources disponibles, plus celles de P_i1...)

Détection de l'interblocage

```
// n et m représentent respectivement le nombre de
processus et de classes de ressources
// Variables décrivant l'état d'allocation
int Dispo[m]; // Nombre de ressources disponibles
int Alloc[n,m]; // Ressources allouées
int Req[n,m];  // Demandes en cours à l'instant considéré
// Variables pour l'agorithme de détection
int Util[m];  // Nombre de ressources utilisables
int Marque[n]; // Tableau de marquage des processus
int  interblocage;
```

Détection de l'interblocage

```
for (p = 0 ; p < n ; p++) {
        Marque[p] = 0;Util = Dispo; interblocage = 0;
}
while (interblocage ==0 && ∀p ∈[0,n-1] such that
(!Marque[p])) {
        if (∀p such that (!Marque[p] && ∀r Req[p,r] ≤
Util[r] ) {
                Marque[p] = 1;
                for (j=0 ; j < m ; j++)  Util[j] = Util[j]-Alloc[p,j];
        }
        else interblocage = 1;
}
```


Détection de l'interblocage

- Si on détecte un interblocage, l'ensemble des processus interbloqués est constitué des processus non marqués à la fin de l'algorithme

Détection de l'interblocage dans Linux et Windows

- Linux noyau 2.6.17 :
  - Détection paramétrable
- Windows XP et suite :
  - Configurable, associé aux Windows Driver Development Tools

Guérison

- Méthodes de guérison forcément autoritaires, avec des effets plus ou moins néfastes pour les processus
- Méthodes de guérison :
  - Destruction de processus pour récupérer leurs ressources
    ⇒ travail fait par les processus détruits est évidemment à refaire
    ⇒ Peut conduire à des comportements incorrects (mise à jour incomplète de données)

Guérison

- Méthodes de guérison :
  - Réquisition. Beaucoup de ressources ne peuvent pas être réquisitionnées sans remettre en cause l'activité passée ou future du processus
    - Réquisition avec retour en arrière : On peut périodiquement sauvegarder un état des processus, constituant ainsi un point de reprise, ré-installé en cas de réquisition de ressource (retour arrière)
    - Sans retour arrière, ressources peuvent être dans un état incohérent