

Séance 5

Détection et prévention de deadlocks



Ce(tte) œuvre est mise à disposition selon les termes de la Licence Creative Commons Attribution – Pas d'Utilisation Commerciale – Pas de Modification 4.0 International.

Rappels

- Principes de la **synchronisation de processus**
 - Problème de la section critique et conditions d'une solution
 - Solutions software, hardware et apportées par l'OS
- Mécanismes de **communication interprocessus**
 - Deux principaux modèles de communication
 - Mémoire partagée et échange de messages

Objectifs

- Comprendre et les **deadlocks**
 - Quatre conditions d'apparition d'un deadlock
 - Graphe d'allocation des ressources systèmes
- Algorithme de détection et de prévention de **deadlocks**
 - Prévention, algorithme du banquier
 - Détection et récupération

Deadlock



Drôle de loi au Kansas

"When two trains approach each other at a crossing, both shall come to a full stop and neither shall start up again until the other has gone."

- Cette **loi du Kansas** n'a pas dû rester longtemps en vigueur !

Elle aurait bloqué tous les trains de l'état

- Plus aucun train **ne peut progresser** dans son exécution

On se retrouve clairement dans une situation d'interblocage

Modélisation d'un système

- Nombre fini de **ressources** R_1, R_2, \dots, R_m
Cycle CPU, fichier, périphérique E/S, lock...
- Plusieurs **processus en compétition** pour les ressources
Un processus demande une instance d'une ressource
- **Trois étapes** lors de l'utilisation de ressources
 - 1 **Demande** avec éventuelle mise en attente
 - 2 **Utilisation** de la ressource
 - 3 **Libération** une fois terminé avec

Ressource renouvelable

- Ressource utilisée par **un processus à la fois**
Temps CPU, périphérique E/S, mémoire, fichier, sémaphore...
- Ressource **non consommée** suite à son utilisation
Elle sera de nouveau disponible après libération
- Par exemple, avec une **mémoire de 200 Kio** disponible
 - P_1
 - request (P_1 , 80 Kio)
 - request (P_1 , 60 Kio)
 - P_2
 - request (P_2 , 70 Kio)
 - request (P_2 , 80 Kio)

Ressource consommable

- Ressource **créée et détruite** par un processus
Interruption, signal, message...
- Certains appels systèmes liés à ces ressources sont **bloquants**
Pour l'attente d'une telle ressource, par exemple
- Par exemple, **échange de message** entre P_1 et P_2
 - P_1
 - receive (P_2)
 - send (P_2, M_1)
 - P_2
 - receive (P_1)
 - send (P_1, M_2)

Deadlock

- ## ■ Processus en compétition pour des ressources partagées

- 1 Le processus fait une requête pour obtenir une ressource
 - 2 Le processus passe en état *WAITING*

- #### ■ État de deadlock pour un ensemble de processus

■ P_1

■ P_2

acquire (P_1 , DVD player)

acquire (P_1 , printer)

acquire (P_2 , printer)

acquire (P_2 , DVD player)

- Qui doit gérer les deadlocks ? Le programmeur ou l'OS ?

Conditions d'apparition d'un deadlock

- Deadlock si les **quatre conditions** suivantes sont remplies

1 Exclusion mutuelle

Au moins une ressource non partageable

2 Hold and wait

Un processus détient des ressources et en attend des pas libres

3 Pas de préemption

Une ressource est libérée volontairement par le processus

4 Attente circulaire

Ensemble $\{P_1, \dots, P_n\}$ tel que P_i attend $P_{i+1} \dots$

- Les quatre conditions ne sont **pas indépendantes** ($4 \implies 2 \dots$)

Graphe d'allocation des ressources systèmes

- Graphe d'Allocation des ressources représente la situation

$$G = (V, E)$$

- 1 Ensemble de processus et ressources

$$V = \{P_1, \dots, P_n\} \cup \{R_1, \dots, R_m\}$$

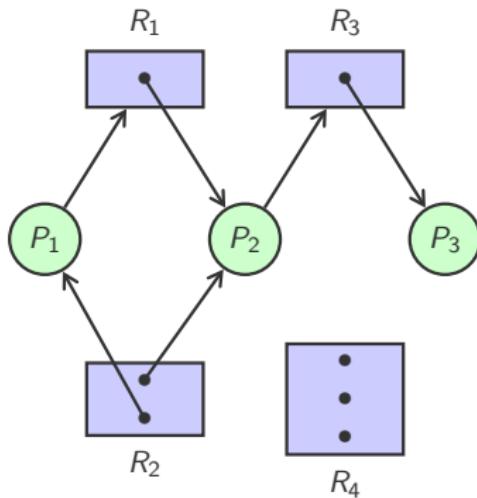
- 2 Lien de demande (d'une ressource par un processus)

$$P_i \rightarrow R_j$$

- 3 Lien d'assignation (d'un processus à une ressource)

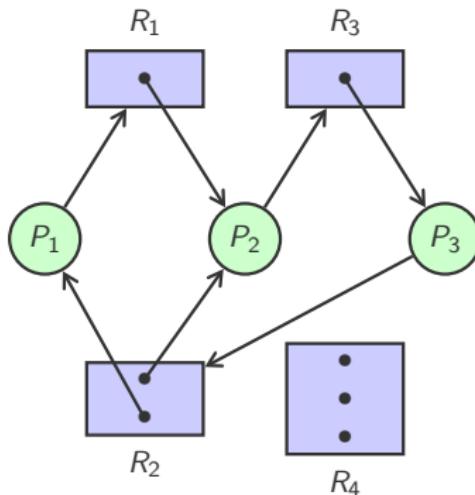
$$R_j \rightarrow P_i$$

Exemple (1)



- P_1 possède R_2
- P_2 possède R_1, R_2
- P_3 possède R_3
- P_1 veut R_1
- P_2 veut R_3
- P_3 ne veut rien

Exemple (2)



- Il existe deux **cycles de longueur minimale** dans ce graphe
 - $P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$
 - $P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$

Détection de deadlock

- Si G ne contient **pas de cycle**, alors il n'y a pas de deadlock

S'il y a un cycle, il peut y avoir un deadlock

- Si **chaque ressource**...

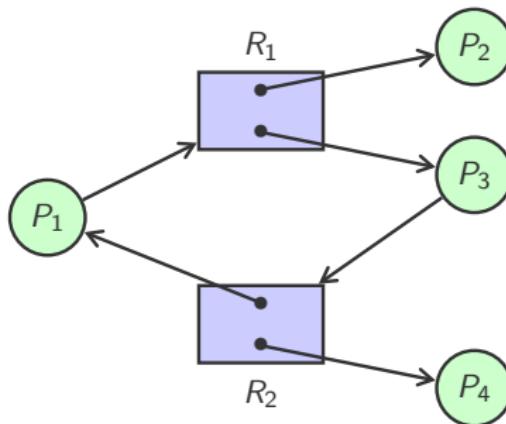
- ...n'a qu'**une instance** : cycle = deadlock

Condition nécessaire et suffisante

- ...peut avoir **plusieurs instances**, on ne peut rien dire

Condition nécessaire

Exemple (3)



- Présence d'**un cycle** dans ce graphe
 - $P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$



Gestion de deadlock

Gestion de deadlock

- Trois possibilités pour gérer un deadlock dans un système

1 Prévention

Protocole pour ne jamais atteindre un état de deadlock

2 Détection

Détection d'un état de deadlock, récupération (en sortir)

3 Ignorance

Supposition que les deadlocks n'arrivent « jamais »

- Linux et Windows utilisent l'ignorance

C'est aux développeurs de faire gaffe dans leurs applications

Prévention de deadlock (1)

! S'assurer que les 4 conditions ne surviennent pas en même temps

1 Exclusion mutuelle

- Doit être présente pour une ressource non-partageable
- On ne peut donc jouer sur cette condition (e.g. mutex lock)

2 Hold and wait

- Un processeur demandeur ne peut détenir de ressources
- Demande et allocation des ressources au début de l'exécution
- Demande satisfaite que lorsque le processus ne détient rien
- Faible utilisation des ressources et problème de famine

Prévention de deadlock (2)

3 Pas de préemption

- Demande d'une ressource détenue par un autre processus libère les ressources détenues
- Liste de ressources attendues par processus
- Redémarré si anciennes et nouvelles ressources disponibles
- Applicables si facilement sauvegardée (registre CPU...)

4 Attente circulaire

- Imposer un ordre total sur les ressources
- Les processus doivent faire leurs demandes suivant cet ordre

Attente circulaire

- Imposer l'ordre ne garantit pas toujours l'absence de deadlock

```
void transaction (Account from, Account to, double amount)
{
    mutex lock1, lock2;
    lock1 = get_lock (from);
    lock2 = get_lock (to);

    acquire (lock1);
    acquire (lock2);

    withdraw (from, amount);
    deposit (to, amount);

    release (lock2);
    release (lock1);
}

// ...

transaction (A, B, 25);
transaction (B, A, 50);
```

Évitement de deadlock

- Deadlock évités en **limitant** les requêtes pour des ressources
Limite l'utilisation des ressources et le débit des processus
- Déclaration du **maximum** de chaque ressource nécessaire
Examen de l'allocation des ressources pour traiter demande
- Une ressource est soit **disponible ou allouée**

État sain

- **État sain** s'il existe une séquence saine des processus

Peut allouer les ressources nécessaires à tous les processus

- **Séquence saine** $P = \langle P_1, P_2, \dots, P_n \rangle$

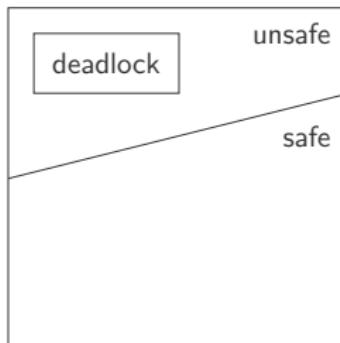
Demandes de ressources de P_i satisfaisables avec...

- ...les ressources actuellement disponibles
- et les ressources détenues par P_j (pour $j < i$)

État du système

- Deux **états** possibles pour le système
 - **Sain** : sans deadlock
 - **Non sain** : avec deadlock ou deadlock possible dans le futur
- Il faut **éviter** les états non-sains

Demande de ressource disponible pourrait être mise en attente



Exemple

- Douze disques à bande magnétique et trois processus
- État actuel, sain ou non ?

	Besoins maximaux	Détentions actuelles
P_0	10	5
P_1	4	2
P_2	9	2

- Si demande de P_2 pour un disque acceptée, sain ou non ?

	Besoins maximaux	Détentions actuelles
P_0	10	5
P_1	4	2
P_2	9	3

Graphe d'allocation des ressources

- Limitation à **une seule instance** de chaque ressource

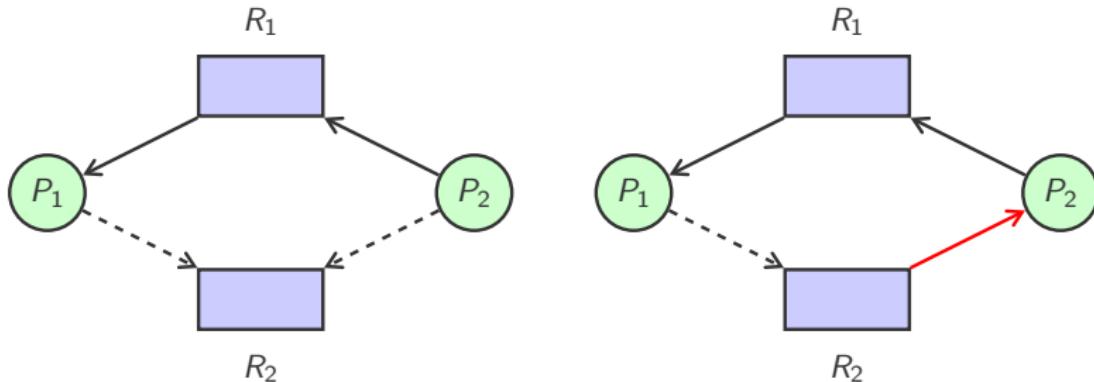
Permet d'utiliser une variante de la technique précédente

- **Lien de requête** (d'une ressource par un processus)

$$P_i \rightarrow R_j$$

- Lien de requête $P_i \rightarrow R_j$ devient lien de demande $P_i \rightarrow R_j$
 - Lien d'assignation $R_j \rightarrow P_i$ devient lien de requête $P_i \rightarrow R_j$
 - Lien de requêtes doivent être connus a priori
-
- P_i demande R_j **acceptée** si $R_j \rightarrow P_i$ ne crée pas un cycle

Exemple



- Étude de la situation si P_2 fait une demande pour R_2
- Demande à **refuser** car création d'un cycle donc état non sain

En effet, on a un deadlock si P_1 prend R_2

Algorithme du banquier

- Plusieurs instances par ressource possible

Mais algorithme moins efficace que celui avec les cycles

- Plusieurs conditions pour pouvoir l'utiliser

- Déclaration à l'avance du nombre maximal de chaque ressource
- Un processus peut devoir attendre pour une ressource
- et une fois reçue, il doit les rendre dans un temps fini

Structures de donnée

- **available**[j] (taille m)

Nombre d'instances disponibles de R_j

- **max**[i][j] (taille $n \times m$)

Demande maximale par P_i pour R_j

- **allocation**[i][j] (taille $n \times m$)

Nombre de ressources de R_j détenues par P_i

- **need**[i][j] (taille $n \times m$)

Nombre de ressources additionnelles nécessaires de R_j pour P_i

need = max - allocation

Algorithme : État sain (1)

1 **work** = **available**

finish[i] = *false* pour $i \in [0, \dots, n - 1]$

2 Trouver un i tel que

1 **finish**[i] == *false*

2 **need** _{i} ≤ **work**

Si un tel i n'existe pas, aller à l'étape 4

3 **work** = **work** + **allocation** _{i}

finish[i] = *true*

Retourner à l'étape 2

4 Si **finish**[i] == *true* pour tous les i , alors système en état sain

Algorithme : État sain (2)

	allocation			max			available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	3	3	2
P_1	2	0	0	3	2	2			
P_2	3	0	2	9	0	2			
P_3	2	1	1	2	2	2			
P_4	0	0	2	4	3	3			

- État du système actuellement **sain**

Séquence d'exécution possible : $\langle P_1, P_3, P_4, P_2, P_0 \rangle$

Algorithme : Demande de ressource (1)

Soit request_i la demande de P_i

- 1 Si $\text{request}_i \leq \text{need}_i$, alors aller à l'étape 2
Sinon générer une exception
- 2 Si $\text{request}_i \leq \text{available}$, alors aller à l'étape 3
Sinon P_i doit attendre
- 3 Effectuer les modifications suivantes
 $\text{available} = \text{available} - \text{request}_i$
 $\text{allocation}_i = \text{allocation}_i + \text{request}_i$
 $\text{need}_i = \text{need}_i - \text{request}_i$

Si l'état atteint est sain, la demande est acceptée

Algorithme : Demande de ressource (2)

	allocation			max			available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	3	3	2
P_1	2	0	0	3	2	2			
P_2	3	0	2	9	0	2			
P_3	2	1	1	2	2	2			
P_4	0	0	2	4	3	3			

- Demande par P_1 : **request**₁ = (1, 0, 2) acceptée

Séquence d'exécution possible : $\langle P_1, P_3, P_4, P_0, P_2 \rangle$

- Ensuite : **request**₄ = (3, 3, 0) et **request**₀ = (0, 2, 0) refusées

Ressources pas disponibles ou mène à un état unsafe

Détection de deadlock

- Laisser le système **atteindre** un état de deadlock

Aucune prévention, on laisse juste les processus vivre leur vie

- **Deux algorithmes** peuvent être proposés par le système

- Examen de l'état du système et détection du deadlock
- Récupération du système

Graphe wait-for (1)

- Limitation à **une seule instance** de chaque ressource

$$G = (V, E)$$

1 Ensemble de processus

$$V = \{P_1, \dots, P_n\}$$

2 Lien d'attente

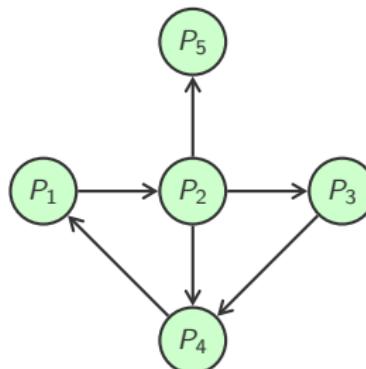
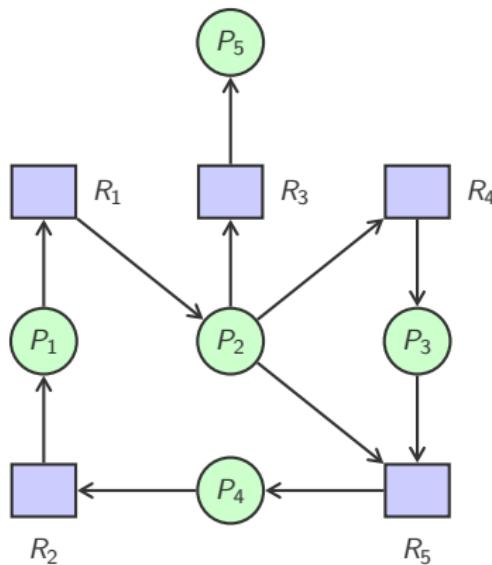
$$P_i \rightarrow P_j$$

P_i attend que P_j libère une ressource dont il a besoin

($P_i \rightarrow R_q \rightarrow P_j$ existe dans le graphe d'allocation des ressources)

Graphe wait-for (2)

- Un deadlock existe s'il y a **un cycle** dans le graphe wait-for



Algorithme de détection (1)

- Plusieurs instances par ressources possible

- Structures de données

- **available**[j] (taille m)

Nombre d'instances disponibles de R_j

- **allocation**[i][j] (taille $n \times m$)

Nombre de ressources de R_j détenues par P_i

- **request**[i][j] (taille $n \times m$)

Demande actuelle additionnelle de P_i pour R_j

Algorithme de détection (2)

1 **work** = **available**

finish[i] = (**allocation** $_i$ == 0) pour $i \in [0, \dots, n - 1]$

2 Trouver un i tel que

1 **finish**[i] == *false*

2 **request** $_i \leq \text{work}$

Si un tel i n'existe pas, aller à l'étape 4

3 **work** = **work** + **allocation** $_i$

finish[i] = *true*

Retourner à l'étape 2

4 Si **finish**[i] == *false* pour un certain i , alors deadlock

Algorithme de détection (3)

	allocation			request			available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	0	0	0	0	0	0
P_1	2	0	0	2	0	2			
P_2	3	0	3	0	0	0			
P_3	2	1	1	1	0	0			
P_4	0	0	2	0	0	2			

- Le système n'est **pas en deadlock**

Séquence d'exécution possible $\langle P_0, P_2, P_3, P_1, P_4 \rangle$

- Demande additionnelle par P_2 : **request** $_2 = (0, 0, 1)$

Plus assez de ressources pour satisfaire les autres processus

Utilisation de l'algorithme de détection

- Quand faut-il **appeler** l'algorithme de détection ?
 - À quelle **fréquence** un deadlock risque-t-il de se produire ?
 - **Combien** de processus faudra-t-il redémarrer ?
- Deadlock se produit lors d'une **demande pas satisfaite**

Exécution à chaque demande ou à intervalle régulier

En cas de deadlock

- Prévenir l'opérateur qui gère le deadlock « à la main »

Fenêtre « le programme ne répond pas »

- Récupération automatique par le système

- Terminaison de processus
- Préemption de ressources

Terminaison de processus

- Terminaison forcée de processus
 - Terminer tous les processus en deadlock
 - Terminer un processus à la fois jusqu'à disparition du deadlock
- Récupération des ressources allouées aux processus quittés

Peut d'un coup débloquer la situation
- La terminaison d'un processus n'est pas triviale

En pleine impression, en train d'écrire un fichier...

Politique de terminaison

- Critères possibles pour choisir les processus à terminer
 - Priorité du processus
 - Temps d'exécution déjà écoulé et temps restant
 - Ressources déjà utilisées
 - Ressources additionnelles demandées
 - Combien de processus à terminer
 - Type du processus : interactif ou batch

Préemption de ressources

- Ressource préemptée à un processus pour la donner à un autre
- Trois problèmes à résoudre
 - Choisir une victime

Processus et ressources
 - Retour en arrière

Remettre le processus dans un état sain passé
 - Famine possible

Éviter de toujours choisir la même victime

Crédits

- <https://www.flickr.com/photos/kulor/3614236257>
- <https://www.flickr.com/photos/1024/858454965>