université angers
FACULTÉ
DES SCIENCES
Unité de formation et de recherche

UNIVERSITE
BRETAGNE
LOIRE

Système

Fabien GARREAU
sur la base du cours d'André Rossi

Université d'Angers

fabien.garreau@univ-angers.fr

L2 MPCIE, 2018–2019

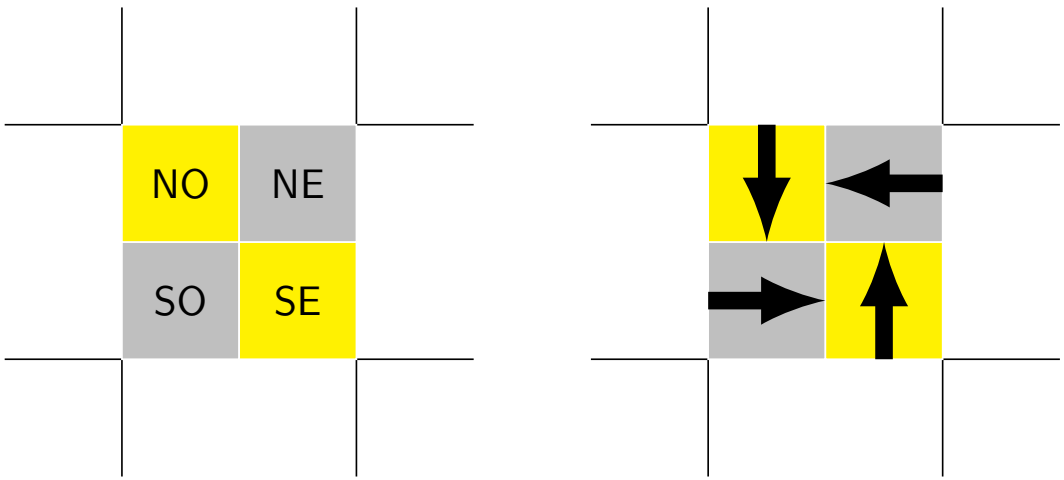
Interblocages

Un ensemble de processus est dans un état d'**interblocage** si chaque processus est en attente d'un événement qui ne peut être provoqué que par un autre processus de cet ensemble.
Cette situation se produit quand des processus concurrents cherchent à acquérir des ressources (imprimante, écran, lecteur, fichiers, mémoire centrale).
Un interblocage se produit si les 4 conditions suivantes sont simultanément vérifiées.

1. **Exclusion mutuelle** : les ressources ne peuvent pas être partagées.
2. **Détention et attente** : il existe un processus détenant au moins une ressource, et qui en demande une autre, mais la ressource demandée n'est pas disponible.
3. **Non réquisition** : les ressources déjà allouées à un processus ne peuvent pas lui être retirées. Une ressource ne peut être libérée que par le processus qui l'a réservée.
4. **Attente circulaire** : Il existe un ensemble de n processus $\{P_0, P_1, \ldots, P_{n-1}\}$ en attente tel que pour tout $i \in \{0, \ldots, n-1\}$, le processus P_i attend une ressource détenue par $P_{(i+1) \bmod n}$.

Exemple d'interblocage

On considère quatre véhicules à un carrefour : chacun occupe une partie du carrefour et en réclame une autre pour pouvoir avancer.



1. **Exclusion mutuelle** : emplacements non partageables
2. **Détention et attente** : chaque véhicule occupe un emplacement et en demande un autre
3. **Non réquisition** : impossible de demander à un véhicule de reculer
4. **Attente circulaire** : NO $\rightarrow$ SO $\rightarrow$ SE $\rightarrow$ NE $\rightarrow$ NO

Évitement des interblocages

On tente d'éviter les interblocages à tout moment sur la base des ressources disponibles, de celles qui seront libérées par les processus en cours, et sur les besoins futurs des processus.
Un état du système est dit **sûr** si

1. Il existe une séquence de processus $(P_1, P_2, \ldots, P_n)$ telle que pour tout $i \in \{1, \ldots, n\}$, la libération des ressources détenues par tous les processus $j < i$ (ainsi que les ressources disponibles) suffisent à satisfaire les demandes du processus P_i
2. Les demandes du processus P_1 peuvent être satisfaites par les ressources disponibles

Sinon, le système est dans un état **non sûr**.

Évitement des interblocages

Un état sûr implique l'absence d'interblocage. Réciproquement, un interblocage est associé à un état non sûr. Cependant, un état non sûr ne conduit pas toujours à un interblocage, parce que les processus ne demandent pas toujours toutes les ressources dont ils auront besoin en une seule fois.
Exemple : 3 processus, 12 ressources au total (dont 3 sont libres)

	Demande maximale	Allocation	Besoin courant
P_0	10	5	5
P_1	4	2	2
P_2	9	2	7

Cet état est-il sûr ?
Oui, car la séquence (P_1, P_0, P_2) permet d'éviter tout interblocage.
Vérifions-le.

Évitement des interblocages

	Demande maximale	Allocation	Besoin courant
P_0	10	5	5
P_1	4	2	2
P_2	9	2	7

Considérons la séquence (P_1, P_0, P_2) :

- P_1 peut commencer immédiatement (2 ressources sont nécessaires, il y en a 3). Une fois terminé, il libère ses 4 ressources, et on a 5 ressources disponibles
- P_0 nécessite 5 ressources, et 5 sont disponibles. Il se termine et 10 ressources sont disponibles
- P_2 nécessite 7 ressources, et 10 sont disponibles. Il se termine et 12 ressources sont disponibles

A l'issue de la séquence, toutes les ressources du système doivent avoir été restituées.

Évitement des interblocages

On reprend l'exemple précédent, mais on a alloué une ressource libre supplémentaire à P_2, ce qui conduit à un état non sûr :

	Demande maximale	Allocation	Besoin courant
P_0	10	5	5
P_1	4	2	2
P_2	9	**3**	**6**

Initialement, 2 ressources sont disponibles. Seul le processus P_1 peut commencer. Quand il se termine, il y a 4 ressources disponibles. C'est insuffisant pour permettre à P_0 ou à P_2 de se terminer. Ces deux processus sont alors en interblocage.

Algorithme du banquier

L'algorithme du banquier permet de décider s'il faut accepter ou refuser les demandes de ressources émises par les processus afin de demeurer dans un état sûr.
Quand un nouveau processus est créé, il doit indiquer le nombre maximum de ressources de chaque type dont il a besoin. Ce nombre doit être inférieur ou égal à la capacité du système. Les processus émettent des demandes de ressources, et partant de l'état initial (sûr), on décide, pour toute demande, si on la satisfait, ou si le processus demandeur doit attendre que d'autres ressources se libèrent. Pour ce faire, on utilise la procédure `TestSur` pour vérifier que le nouvel état potentiel (après allocation des ressources) est sûr.

Algorithme du banquier : structures de données

- n : nombre de processus demandant des ressources
- m : nombre de types de ressources dans le système
- $Disponible(j)$ est le nombre de ressource de type j actuellement disponibles (non allouées) $\forall j \in \{1, \ldots, m\}$
- $Max(i, j)$ est le nombre maximal de ressources de type j que peut demander le processus i, $\forall (i, j) \in \{1, \ldots, n\} \times \{1, \ldots, m\}$
- $Allocation(i, j)$ est le nombre de ressources de type j actuellement détenues par le processus i, $\forall (i, j) \in \{1, \ldots, n\} \times \{1, \ldots, m\}$
- $Besoin(i, j)$ est le nombre maximal de ressources de type j que le processus i peut encore demander dans l'état actuel, $\forall (i, j) \in \{1, \ldots, n\} \times \{1, \ldots, m\}$
- $Termine(i)$ est vrai si le processus i est terminé, faux sinon $\forall i \in \{1, \ldots, n\}$
- $DispoTest(j)$ est le nombre de ressources de type j actuellement disponibles (variable locale de `TestSur`)

Algorithme du banquier : notations

On a la relation matricielle $Max = Allocation + Besoin$
Notations : Soient deux vecteurs X et Y ayant k éléments.

- $X \leq Y$ signifie que $X(i) \leq Y(i)$ pour tout $i \in \{1, \ldots, k\}$
- $X < Y$ signifie que $X \leq Y$ et que $X \neq Y$
- $Allocation_i$ désigne la ligne i de la matrice $Allocation$, ce vecteur fait apparaître l'allocation de ressource du processus i
- $Besoin_i$ désigne la ligne i de la matrice $Besoin$, ce vecteur fait apparaître le nombre maximal de chaque ressource que le processus i peut demander dans l'état courant
- $Demande_i$ représente une demande émise par le processus i, où $Demande_i(j)$ est le nombre de ressources de type j demandées $\forall j \in \{1, \ldots, m\}$

Algorithme du banquier : `TestSur`

L'algorithme `TestSur` permet de savoir si un état est sûr.

1. $Termine(i) = faux$ pour tout i
   $DispoTest$ est initialisé à $Disponible$
2. Trouver un processus i dans $\{1, \ldots, n\}$ tel que
   - $Termine(i) == faux$
   - $Besoin_i \leq DispoTest$

   Si un tel i n'existe pas, aller à l'étape 4.
3. `/* Simulation de la fin du processus` i `*/`
   $DispoTest = DispoTest + Allocation_i$
   $Termine(i) = vrai$
   Aller à l'étape 2
4. Si $Termine(i) == vrai$ pour tout $i \in \{1, \ldots, n\}$, alors retourner `vrai`, sinon retourner `faux` (l'état testé est non sûr).

Algorithme du banquier

1. Si $Demande_i \leq Besoin_i$, aller à l'étape 2. Sinon émettre une erreur : le processus demande plus qu'il ne devrait.
2. Si $Demande_i \leq Disponible$, aller à l'étape 3. Sinon le processus i doit attendre, faute de ressources disponibles.
3. `/* Satisfaire la demande de` i `conduit-elle à un état sûr ? */`
   $Disponible = Disponible - Demande_i$
   $Allocation_i = Allocation_i + Demande_i$
   $Besoin_i = Besoin_i - Demande_i$
   Si `TestSur` retourne `faux` alors faire :
   Le processus i doit attendre, restauration de l'état précédent :
   $Disponible = Disponible + Demande_i$
   $Allocation_i = Allocation_i - Demande_i$
   $Besoin_i = Besoin_i + Demande_i$
   Sinon fin (l'état courant est sûr).

Algorithme du banquier : exemple

Exemple : $n = 5$ processus, et $m = 3$ types de ressources A, B, C.

	Allocation A B C	*Max* A B C
P_0	0 1 0	7 5 3
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

Disponible A B C
3 3 2

On en déduit

	Besoin A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

Quelles sont les ressources disponibles maximales du système ?
$A : 10$, $B : 5$, $C : 7$.
L'état courant est sûr car la séquence $(P_1, P_3, P_4, P_2, P_0)$ conduit à un état sûr. Vérifions-le
[3 3 2] $\rightarrow$ [5 3 2] $\rightarrow$[7 4 3] $\rightarrow$[7 4 5] $\rightarrow$[10 4 7] $\rightarrow$[10 5 7]

Algorithme du banquier : exemple

Supposons que P_1 demande les ressources $Demande_1 = [1\ 0\ 2]$ pour A, B et C respectivement.
Les étapes 1 et 2 du banquier sont franchies avec succès.
On accepte la demande, ce qui conduit à

$$\begin{aligned} Disponible &= [2\ 3\ 0] \\ Allocation_1 &= [3\ 0\ 2] \\ Besoin_1 &= [0\ 2\ 0] \end{aligned}$$

On appelle `TestSur` :

`Termine = [0 0 0 0 0]` et `DispoTest = [2 3 0]`

Recherche d'un processus i non terminé tel que $Besoin_i \leq DispoTest$

- $Besoin_1 \leq [2\ 3\ 0]$
- `DispoTest = [2 3 0] + [3 0 2] = [5 3 2]`
- `Termine(1) = vrai`

Recherche d'un processus i non terminé tel que $Besoin_i \leq DispoTest$

- $Besoin_3 \leq [5\ 3\ 2]$
- `DispoTest = [5 3 2] + [2 1 1] = [7 4 3]`
- `Termine(3) = vrai`

Algorithme du banquier : exemple

```
Recherche d'un processus i non terminé tel que Besoin_i ≤ DispoTest
      Besoin_0 ≤[7 4 3]
      DispoTest = [7 4 3] + [0 1 0] = [7 5 3]
      Termine(0) = vrai
Recherche d'un processus i non terminé tel que Besoin_i ≤ DispoTest
      Besoin_2 ≤ [7 5 3]
      DispoTest = [7 5 3] + [3 0 2] = [10 5 5]
      Termine(2) = vrai
Recherche d'un processus i non terminé tel que Besoin_i ≤ DispoTest
      Besoin_4 ≤ [10 5 5]
      DispoTest = [10 5 5] + [0 0 2] = [10 5 7]
      Termine(4) = vrai
TestSur retourne vrai
```

L'algorithme du banquier se termine en acceptant la demande du processus 1. La séquence des processus ainsi construite est $(P_1, P_3, P_0, P_2, P_4)$

Décrire le nouvel état en spécifiant *Allocation*, *Max*, *Disponible* et *Besoin*.

Algorithme du banquier : exemple

	Allocation A B C	*Max* A B C
P_0	0 1 0	7 5 3
P_1	**3 0 2**	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

Disponible A B C
2 3 0

On en déduit

	Besoin A B C
P_0	7 4 3
P_1	**0 2 0**
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

1. A partir de cet état, peut-on accepter la $Demande_4 = [3\ 3\ 0]$? La demande est inférieure au besoin courant de P_4, mais les ressources disponibles ne sont pas suffisantes. Demande rejetée.
2. Peut-on accepter la $Demande_0 = [0\ 2\ 0]$? La demande est inférieure ou égale au besoin courant maximal de P_0, et les ressources disponibles sont suffisantes. L'état atteint en cas d'acceptation de la demande est-il sûr ?

Algorithme du banquier : exemple

On accepte temporairement $Demande_0 = [0\ 2\ 0]$, voici l'état obtenu :

	Allocation A B C	*Max* A B C
P_0	0 **3** 0	7 5 3
P_1	3 0 2	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

Disponible A B C
2 **1** 0

et

	Besoin A B C
P_0	7 **2** 3
P_1	0 2 0
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

On appelle `TestSur`

`Termine` = `[0 0 0 0 0]` et `DispoTest` = `[2 1 0]`

`Recherche d'un processus` i `non terminé`

`tel que` $\texttt{Besoin}_i \leq$ `DispoTest`

`TestSur retourne faux`

Par conséquent, on revient à l'état précédent (qui est sûr) et on rejette $Demande_0 = [0\ 2\ 0]$.