Partage de ressources et interblocages

Master 2 Informatique - UFR S.A.T

Pr. Ousmane THIARE

ousmane.thiare@ugb.edu.sn
http://www.ousmanethiare.com/

13 septembre 2025

Chapitre 5 : Partage de ressources et interblocages

Introduction

Blocage : situation qui peut être normale. Ainsi, tout processus peut être bloqué à un moment donné : attente d'entrée/sortie de mémoire, ou simplement du processus.
Inter-blocage : situation où un ensemble de processus sont bloqués de façon défiitive (alors bien sûr que ni le matériel ou ni le système ne sont en panne).
Ainsi, par exemple :

- P_1 dispose de la ressource R_1 et attend la ressource R_2 pour pouvoir continuer

$\Longrightarrow$ inter-blocage : cette situation est à priori définitive si on ne dispose pas d'autres ressources du type R_1 ou R_2.
On parle d'*attente circulaire*.

Introduction

Blocage : situation qui peut être normale. Ainsi, tout processus peut être bloqué à un moment donné : attente d'entrée/sortie de mémoire, ou simplement du processus.
Inter-blocage : situation où un ensemble de processus sont bloqués de façon défiitive (alors bien sûr que ni le matériel ou ni le système ne sont en panne).
Ainsi, par exemple :

- P_1 dispose de la ressource R_1 et attend la ressource R_2 pour pouvoir continuer
- P_2 dispose de la ressource R_2 et attend la ressource R_1 pour pouvoir continuer

$\Longrightarrow$ inter-blocage : cette situation est à priori définitive si on ne dispose pas d'autres ressources du type R_1 ou R_2.
On parle d'*attente circulaire*.

Introduction

On peut montrer que trois conditions sont nécessaires à l'interblocage :

- exclusion mutuelle sur les ressources

Trois approches possibles pour résoudre ce problème.

Introduction

On peut montrer que trois conditions sont nécessaires à l'interblocage :

- exclusion mutuelle sur les ressources
- attente circulaire

Trois approches possibles pour résoudre ce problème.

Introduction

On peut montrer que trois conditions sont nécessaires à l'interblocage :

- exclusion mutuelle sur les ressources
- attente circulaire
- non réquisition : lorsqu'un processus possède une ressource, on ne peut le forcer à le libérer.

Trois approches possibles pour résoudre ce problème.

Introduction

On peut montrer que trois conditions sont nécessaires à l'interblocage :

- exclusion mutuelle sur les ressources
- attente circulaire
- non réquisition : lorsqu'un processus possède une ressource, on ne peut le forcer à le libérer.

Trois approches possibles pour résoudre ce problème.

- **prévention** : on supprime à priori une des conditions

Introduction

On peut montrer que trois conditions sont nécessaires à l'interblocage :

- exclusion mutuelle sur les ressources
- attente circulaire
- non réquisition : lorsqu'un processus possède une ressource, on ne peut le forcer à le libérer.

Trois approches possibles pour résoudre ce problème.

- **prévention** : on supprime à priori une des conditions
  - soit l'exclusion mutuelle ;

Introduction

On peut montrer que trois conditions sont nécessaires à l'interblocage :

- exclusion mutuelle sur les ressources
- attente circulaire
- non réquisition : lorsqu'un processus possède une ressource, on ne peut le forcer à le libérer.

Trois approches possibles pour résoudre ce problème.

- **prévention** : on supprime à priori une des conditions
  - soit l'exclusion mutuelle ;
  - soit la non-réquisition ;

Introduction

On peut montrer que trois conditions sont nécessaires à l'interblocage :

- exclusion mutuelle sur les ressources
- attente circulaire
- non réquisition : lorsqu'un processus possède une ressource, on ne peut le forcer à le libérer.

Trois approches possibles pour résoudre ce problème.

- **prévention** : on supprime à priori une des conditions
  - soit l'exclusion mutuelle ;
  - soit la non-réquisition ;
  - soit on supprime les possibilités d'attente circulaire

Introduction

- **évitement** : un processus est capable de détecter si son action (sa demande) entraîne un inter-blocage. Cette solution nécessite malheureusement bien souvent :

Introduction

- **évitement** : un processus est capable de détecter si son action (sa demande) entraîne un inter-blocage. Cette solution nécessite malheureusement bien souvent :
  - l'utilisation d'un site centralisateur ;

Introduction

- **évitement** : un processus est capable de détecter si son action (sa demande) entraîne un inter-blocage. Cette solution nécessite malheureusement bien souvent :
  - l'utilisation d'un site centralisateur ;
  - une connaissance (ou au moins une estimation du maximum) des besoins "futurs" en ressources de l'ensemble des processus $\Longrightarrow$ dans ce cas une solution : l'algorithme du banquier qu'on verra plus tard.

Introduction

- **évitement** : un processus est capable de détecter si son action (sa demande) entraîne un inter-blocage. Cette solution nécessite malheureusement bien souvent :
  - l'utilisation d'un site centralisateur ;
  - une connaissance (ou au moins une estimation du maximum) des besoins "futurs" en ressources de l'ensemble des processus $\Longrightarrow$ dans ce cas une solution : l'algorithme du banquier qu'on verra plus tard.
- **détection et reprise** : on laisse l'application se dérouler. Puis lorsqu'on soupçonne un arrêt des processus :

Introduction

- **évitement** : un processus est capable de détecter si son action (sa demande) entraîne un inter-blocage. Cette solution nécessite malheureusement bien souvent :
  - l'utilisation d'un site centralisateur ;
  - une connaissance (ou au moins une estimation du maximum) des besoins "futurs" en ressources de l'ensemble des processus $\Longrightarrow$ dans ce cas une solution : l'algorithme du banquier qu'on verra plus tard.
- **détection et reprise** : on laisse l'application se dérouler. Puis lorsqu'on soupçonne un arrêt des processus :
  - on lance éventuellement une détection de la terminaison ;

Introduction

- **évitement** : un processus est capable de détecter si son action (sa demande) entraîne un inter-blocage. Cette solution nécessite malheureusement bien souvent :
  - l'utilisation d'un site centralisateur ;
  - une connaissance (ou au moins une estimation du maximum) des besoins "futurs" en ressources de l'ensemble des processus $\Longrightarrow$ dans ce cas une solution : l'algorithme du banquier qu'on verra plus tard.
- **détection et reprise** : on laisse l'application se dérouler. Puis lorsqu'on soupçonne un arrêt des processus :
  - on lance éventuellement une détection de la terminaison ;
  - en cas de non terminaison, on cherche s'il y a des processus en inter-blocage (détection). Si c'est le cas, on enclenche une phase de *reprise* par une suppression temporaire d'une des conditions : en général, on tue un ou plusieurs processus afin qu'ils libèrent leurs ressources (*réquisition*). la suite de la reprise dépend de l'application : il n'y a pas d'algorithme général.

Technique de prévention
Niveaux ressources

On fixe des niveaux pour les ressources : un processus ne peut demander de ressources de niveau inférieur ou égal à celles qu'ils détient déjà $\Longrightarrow$ il y a nécessairement dans la chaîne des processus en attente, un processus qui n'attend rien (au pire des cas, c'est celui qui détient des ressources de niveau maximal).

Problèmes :

- chaque processus doit connaître ses besoins futurs : en effet, lorsqu'il demande une quantité Q d'une ressource à un instant T, il doit être sûr que plus tard, il n'en aura besoin d'une quantité Q'>Q car il ne pourra plus demander le complément Q'-Q.

Ainsi, par exemple, si on fixe qu'une imprimante est de niveau 1 et la mémoire est de niveau 2, alors il faut s'affecter l'imprimante avant de faire les calculs : on bloque inutilement une ressource

Technique de prévention
Niveaux ressources

On fixe des niveaux pour les ressources : un processus ne peut demander de ressources de niveau inférieur ou égal à celles qu'ils détient déjà $\Longrightarrow$ il y a nécessairement dans la chaîne des processus en attente, un processus qui n'attend rien (au pire des cas, c'est celui qui détient des ressources de niveau maximal).

Problèmes :

- chaque processus doit connaître ses besoins futurs : en effet, lorsqu'il demande une quantité Q d'une ressource à un instant T, il doit être sûr que plus tard, il n'en aura besoin d'une quantité Q'>Q car il ne pourra plus demander le complément Q'-Q.
- comment fixer les niveaux ?

Ainsi, par exemple, si on fixe qu'une imprimante est de niveau 1 et la mémoire est de niveau 2, alors il faut s'affecter l'imprimante avant de faire les calculs : on

Technique de prévention
Niveau processus

On va utiliser un critère quelconque définissant un ordre total strict entre les processus (par exemple, les dates de création) comme niveau de priorités de ceux-ci. Puis par rapport à ces niveaux, est défini un comportement des processus. Ainsi, par exemple, un processus P_i est autorisé à attendre les ressources d'un autre P_j si et seulement si son niveau (son estampille de création, par exemple) est inférieur à celui de P_j (ou le contraire son niveau est supérieur à celle de P_j).

Attendre ou mourir

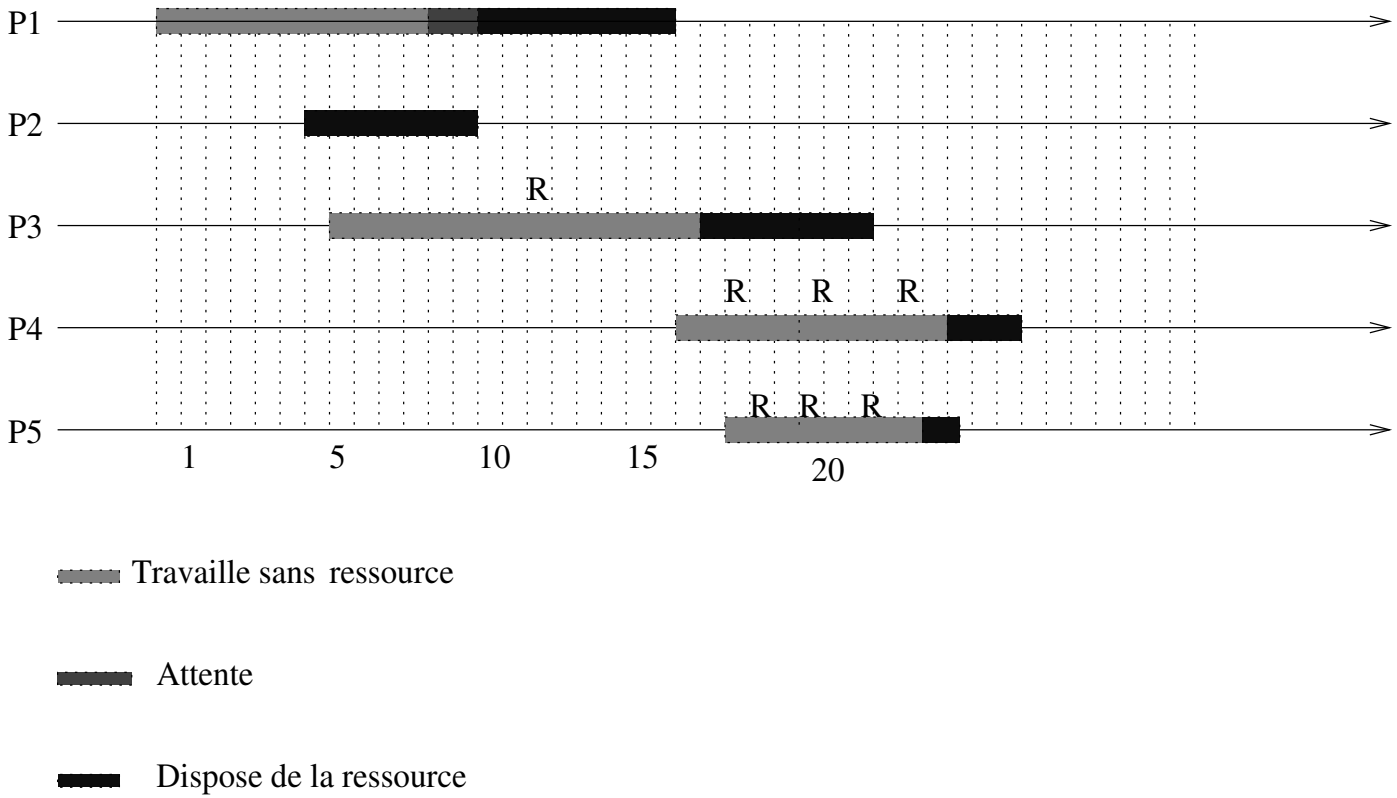
Un processus qui demande une ressource détenue par un autre processus n'est autorisé à attendre que si son estampille (sa date de création) est plus PETITE que celle du processus détenteur de la ressource. Sinon, le processus est "tué" : il libère toutes ses ressources et reprend à zéro.

Soient 5 processus faisant des demandes de la ressource unique R, (avec D_c, date de création, D_d date de demande de ressource (par rapport à la date de création du processus) et D_u durée d'utilisation après avoir reçu la ou les ressources.

Proc	D_c	D_d	D_u
1	0	7	7
2	3	0	5
3	6	5	6
4	15	2	2
5	16	1	1

Attendre ou mourir

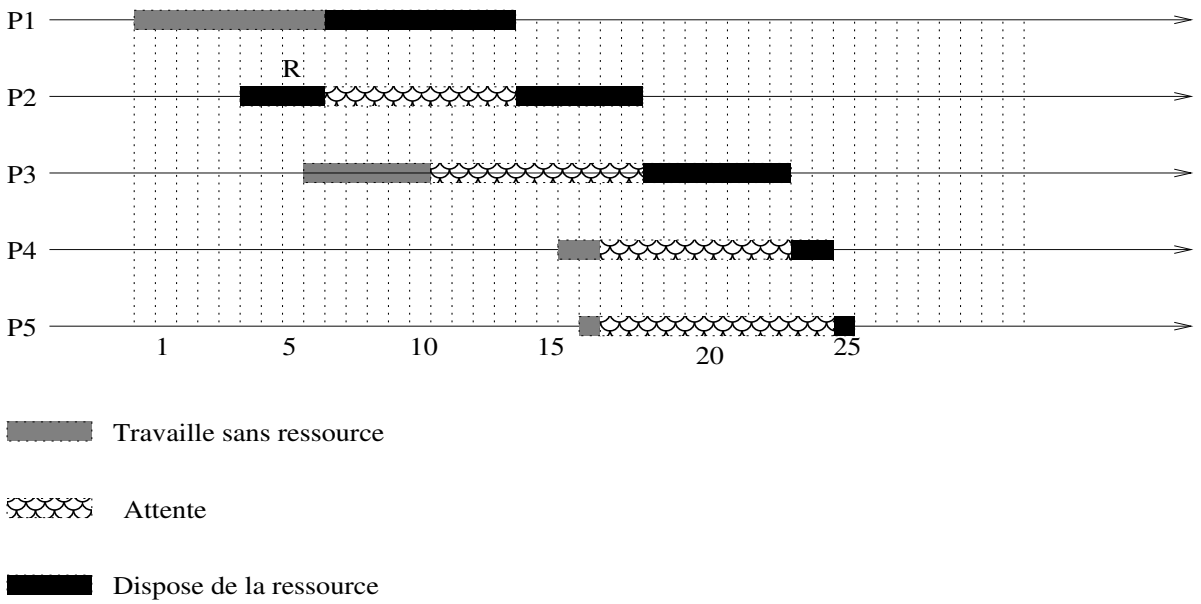
Appliquons cet algorithme aux 5 processus :



Temps total : 25 secondes ; Temps d'attente : 1 seconde ; 9 reprises Temps CPU utilisé : 52 secondes

Blesser ou mourir

Un processus qui demande une ressource détenue par un autre processus n'est autorisé à attendre que si son estampille est plus GRANDE que celle du processus détenteur de la ressource. Sinon, le processus *détenteur* est "tué" : celui-ci libère toutes les ressources et reprend à zéro. Appliquons cet algorithme aux 5 processus :



Évitement
Définition de l'état d'un système

On dispose de M types de ressources $R_1, \cdots, R_m$ avec N_i éléments pour R_i. On appelle *capacité* du système, le vecteur $N = (N_1, \cdots, N_m)$.
Soit $S = P_1 \parallel P_2 \parallel \cdots \parallel P_n$ un système de n processus indépendants.
Chaque processus P_i est décomposable en une suite de np_i tâches $T_i(1), \cdots, T_i(np_i)$ séquentielles telles que :

- à une tâche $T_i(j)$ correspond deux évenements :

Évitement
Définition de l'état d'un système

On dispose de M types de ressources $R_1, \cdots, R_m$ avec N_i éléments pour R_i. On appelle *capacité* du système, le vecteur $N = (N_1, \cdots, N_m)$.
Soit $S = P_1 \parallel P_2 \parallel \cdots \parallel P_n$ un système de n processus indépendants.
Chaque processus P_i est décomposable en une suite de np_i tâches $T_i(1), \cdots, T_i(np_i)$ séquentielles telles que :

- à une tâche $T_i(j)$ correspond deux évenements :
  - $d_i(j)$: début de la j-ième tâche de P_i

Évitement
Définition de l'état d'un système

On dispose de M types de ressources $R_1, \cdots, R_m$ avec N_i éléments pour R_i. On appelle *capacité* du système, le vecteur $N = (N_1, \cdots, N_m)$.
Soit $S = P_1 \parallel P_2 \parallel \cdots \parallel P_n$ un système de n processus indépendants.
Chaque processus P_i est décomposable en une suite de np_i tâches $T_i(1), \cdots, T_i(np_i)$ séquentielles telles que :

- à une tâche $T_i(j)$ correspond deux évenements :
  - $d_i(j)$: début de la j-ième tâche de P_i
  - $f_i(j)$: fin de la j-ième tâche de P_i.

Évitement
Définition de l'état d'un système

On dispose de M types de ressources $R_1, \cdots, R_m$ avec N_i éléments pour R_i. On appelle *capacité* du système, le vecteur $N = (N_1, \cdots, N_m)$.
Soit $S = P_1 \parallel P_2 \parallel \cdots \parallel P_n$ un système de n processus indépendants.
Chaque processus P_i est décomposable en une suite de np_i tâches $T_i(1), \cdots, T_i(np_i)$ séquentielles telles que :

- à une tâche $T_i(j)$ correspond deux évenements :
  - $d_i(j)$: début de la j-ième tâche de P_i
  - $f_i(j)$: fin de la j-ième tâche de P_i.
- un processus ne peut demander des processus "supplémentaires" qu'au début d'une tâche. Il ne peut en libérer qu'à la fin d'une tâche.

Par exemple :

Processus: P_i	tache: $T_i(j)$	demande$_i(j)$	libere$_i(j)$
P1	T1(1) T1(2)	(1,2) (2,0)	(0,1) (3,1)
P2	T2(1) T2(2)	(1,1) (2,2)	(0,0) (3,3)
P3	T3(1)	(1,0)	(1,0)

Définitions

Comportement du système On notera a_k un événement de début ou de fin d'une tâche d'un des processus de S.
Soit $p = 2.(np_1 + \cdots + np_n)$ le nombre total d'événements du système.
Alors $w = a_0 a_1 \cdots a_p$ représente un déroulement possible (ou comportement) du système.
Soit $Dispo(k) = Dispo_1(k), Dispo_2(k), \cdots, Dispo_m(k))$ le vecteur des ressources disponibles après a_k où $Dispo_i(k)$ est le nombre de ressources R_i disponible après a_k.

Définition

état d'un système A chaque événement a_k est alors associé un nouvel état s_k du système. On définit l'état d'un système par deux matrices $n \times m$:

- $Besoin_{i,j}(k)$: nombre d'unités de la ressource du type R_j supplémentaires demandées par P_i après a_k et nécessaires pour lancer la tâche suivante : cette tâche ne pourra donc être lancée que si $Besoin_{i,j}(k) < Dispo(k)$;

Par définition : $Besoin_i(0) = demande_i(1)$ c'est-à-dire les besoins initiaux de P_i sont ceux de sa première tâche. Réciproquement $Tenu_i(0) = (0, \cdots, 0)$ pour tout i.

Définition

état d'un système A chaque événement a_k est alors associé un nouvel état s_k du système. On définit l'état d'un système par deux matrices $n \times m$:

- $Besoin_{i,j}(k)$: nombre d'unités de la ressource du type R_j supplémentaires demandées par P_i après a_k et nécessaires pour lancer la tâche suivante : cette tâche ne pourra donc être lancée que si $Besoin_{i,j}(k) < Dispo(k)$;
- $Tenu_{i,j}(k)$: nombre d'unités de la ressource du type R_j détenues par P_i après a_k.

Par définition : $Besoin_i(0) = demande_i(1)$ c'est-à-dire les besoins initiaux de P_i sont ceux de sa première tâche. Réciproquement $Tenu_i(0) = (0, \cdots, 0)$ pour tout i.

Définitions

Calcul e l'état d'un système suite à un événement a_k

- Cas où a_k correspond à $d_i(t)$ c'est-à-dire au début de la tâche $T_i(t)$:

On peut calculer le nombre de ressources R_j disponibles après l'événement a_k par $Dispo_j(k) = N_j - \sum_{i=1}^{n} Tenu_{i,j}(k)$.

Définitions

Calcul e l'état d'un système suite à un événement a_k

- Cas où a_k correspond à $d_i(t)$ c'est-à-dire au début de la tâche $T_i(t)$:
  - $Besoin_{i,j}(k) = 0$ (le processus a été satisfait, il a commencé la tâche $T_i(t)$.

On peut calculer le nombre de ressources R_j disponibles après l'événement a_k par $Dispo_j(k) = N_j - \sum_{i=1}^{n} Tenu_{i,j}(k)$.

Définitions

Calcul e l'état d'un système suite à un événement a_k

- Cas où a_k correspond à $d_i(t)$ c'est-à-dire au début de la tâche $T_i(t)$:
  - $Besoin_{i,j}(k) = 0$ (le processus a été satisfait, il a commencé la tâche $T_i(t)$.
  - $Tenu_{i,j}(k) = Tenu_{i,j}(k-1) + demande_i(t)$

On peut calculer le nombre de ressources R_j disponibles après l'événement a_k par $Dispo_j(k) = N_j - \sum_{i=1}^{n} Tenu_{i,j}(k)$.

Définitions

Calcul e l'état d'un système suite à un événement a_k

- Cas où a_k correspond à $d_i(t)$ c'est-à-dire au début de la tâche $T_i(t)$:
  - $Besoin_{i,j}(k) = 0$ (le processus a été satisfait, il a commencé la tâche $T_i(t)$.
  - $Tenu_{i,j}(k) = Tenu_{i,j}(k-1) + demande_i(t)$
- cas où correspond à $f_i(t)$ c'est-à-dire à la fin de la tâche $T_i(t)$:

On peut calculer le nombre de ressources R_j disponibles après l'événement a_k par $Dispo_j(k) = N_j - \sum_{i=1}^{n} Tenu_{i,j}(k)$.

Définitions

Calcul e l'état d'un système suite à un événement a_k

- Cas où a_k correspond à $d_i(t)$ c'est-à-dire au début de la tâche $T_i(t)$:
  - $Besoin_{i,j}(k) = 0$ (le processus a été satisfait, il a commencé la tâche $T_i(t)$.
  - $Tenu_{i,j}(k) = Tenu_{i,j}(k-1) + demande_i(t)$
- cas où correspond à $f_i(t)$ c'est-à-dire à la fin de la tâche $T_i(t)$:
  - $Besoin_{i,j}(k) = demande_i(t+1)$ (à la fin de la tâche $T_i(t)$, les besoins initiaux de P_i sont ceux de sa prochaine tâche.

On peut calculer le nombre de ressources R_j disponibles après l'événement a_k par $Dispo_j(k) = N_j - \sum_{i=1}^{n} Tenu_{i,j}(k)$.

Définitions

Calcul e l'état d'un système suite à un événement a_k

- Cas où a_k correspond à $d_i(t)$ c'est-à-dire au début de la tâche $T_i(t)$:
  - $Besoin_{i,j}(k) = 0$ (le processus a été satisfait, il a commencé la tâche $T_i(t)$.
  - $Tenu_{i,j}(k) = Tenu_{i,j}(k-1) + demande_i(t)$
- cas où correspond à $f_i(t)$ c'est-à-dire à la fin de la tâche $T_i(t)$:
  - $Besoin_{i,j}(k) = demande_i(t+1)$ (à la fin de la tâche $T_i(t)$, les besoins initiaux de P_i sont ceux de sa prochaine tâche.
  - $Tenu_{i,j}(k) = Tenu_{i,j}(k-1) + libere_i(t)$

On peut calculer le nombre de ressources R_j disponibles après l'événement a_k par $Dispo_j(k) = N_j - \sum_{i=1}^{n} Tenu_{i,j}(k)$.

Définitions

Exemple

Prenons l'exemple précédent avec le comportement suivant :

$w = d_3(1)d_1(1)f_1(1)f_3(1)d_1(2)f_1(2)d_2(1)f_2(1)d_2(2)f_2(2)$

$$S_0 = \left(\begin{pmatrix} 1 & 2 \\ 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \right);$$

$$S_1 = \left(\begin{pmatrix} 1 & 2 \\ 1 & 1 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{pmatrix} \right);$$

$$S_2 = \left(\begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 2 \\ 0 & 0 \\ 1 & 0 \end{pmatrix} \right); \text{Dispo(1)=(2,3)};$$

Dispo(2)=(1,1).

Définitions
Exemple

$$S_3 = \left(\begin{pmatrix} 2 & 0 \\ 1 & 1 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \right) ;$$

$$S_4 = \left(\begin{pmatrix} 2 & 0 \\ 1 & 1 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \right) ;$$

$$S_5 = \left(\begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 3 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \right) ; \text{Dispo(3)=(1,2)} ;$$

Dispo(4)=(2,2) ; Dispo(5)=(0,2).

Définitions
Exemple

$$S_6 = \left(\begin{pmatrix} 2 & 0 \\ 1 & 1 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \right) ;$$

$$S_7 = \left(\begin{pmatrix} 2 & 0 \\ 1 & 1 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \right) ;$$

$$S_8 = \left(\begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 3 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \right) ; \text{Dispo(6)=(3,3)} ;$$

Dispo(7)=(2,2) ; Dispo(8)=(2,2).

Définitions
Exemple

$$S_9 = \left(\begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 3 & 3 \\ 0 & 0 \end{pmatrix} \right);$$

$$S_{10} = \left(\begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \right); \text{Dispo(9)=(0,0)};$$

Dispo(10)=(3,3).

Évitement
Définition d'un inter-blocage

Définition 1 Un processus P_i est dit bloqué si et seulement si il existe j tel que $Besoin_{i,j}(k) > Dispo_j(k)$

Définition 2 L'état s_k est un inter-blocage :

- s'il existe un sous-ensemble D non vide de $\{1, 2, \cdots, n\}$ de processus **bloqués** en attente d'au moins une ressource,

Définition 1 Un processus P_i est dit bloqué si et seulement si il existe j tel que $Besoin_{i,j}(k) > Dispo_j(k)$

Définition 2 L'état s_k est un inter-blocage :

- s'il existe un sous-ensemble D non vide de $\{1, 2, \cdots, n\}$ de processus **bloqués** en attente d'au moins une ressource,
- pour chaque $P_{i \in D}$, il existe au moins une ressource R_j telle que $Besoin_{i,j}(k) > Dispo_j(k) + \sum_{i \notin D} Tenu_{i,j}(k)$:

  même si tous les processus **non bloqués** libèrent toutes leurs ressources, il restera au moins une ressource en quantité insuffisante (car détenue par des processus bloqués).

Donc, quoique fassent les processus non bloqués, chacun des $P_{i \in D}$ conservera au moins un de ses besoins non satisfait.

Attention : à contrario, on peut avoir un ensemble de processus bloqués en attente de ressources sans que cela soit une situation d'interblocage. Par exemple,

$S_3 = \left(\begin{pmatrix} 2 & 0 \\ 0 & 0 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 0 \end{pmatrix} \right)$ n'est pas un inter-blocage : les processus en attente de ressources sont : $T_1(2)$ et $T_3(1)$. Or $Dispo(3) + tenu_2(3) = (2, 2)$. Comme $Besoin_1(3) < (2, 2)$ et $Besoin_3(3) < (2, 2)$, s_3 n'est pas un blocage.

Reprenons l'exemple précédent avec le comportement suivant :

$w = d_1(1)f_1(1)d_2(1)f_2(1)d_1(2)f_1(2)d_2(2)f_2(2)d_3(1)f_3(1)$

$$S_0 = \left(\begin{pmatrix} 1 & 2 \\ 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \right);$$

$$S'_1 = \left(\begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 2 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \right);$$

$$S'_2 = \left(\begin{pmatrix} 2 & 0 \\ 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \right); \text{Dispo(1)=(2,1)};$$

Dispo(2)=(2,2).

$$S'_3 = \left(\begin{pmatrix} 2 & 0 \\ 0 & 0 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 0 \end{pmatrix} \right);$$

$$S'_4 = \left(\begin{pmatrix} 2 & 0 \\ 2 & 2 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 0 \end{pmatrix} \right); \text{Dispo(3)=(1,1)};$$

Dispo(4)=(1,1).

Après a_4, on ne peut pas lancer

- ni $T_1(2)$ car $Besoin_{1,1}(4) = 2 > Dispo_1(4) = 1$

D'où $D = \{1, 2\}$

$$S'_3 = \left(\begin{pmatrix} 2 & 0 \\ 0 & 0 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 0 \end{pmatrix} \right);$$

$$S'_4 = \left(\begin{pmatrix} 2 & 0 \\ 2 & 2 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 0 \end{pmatrix} \right); \text{Dispo(3)=(1,1)};$$

Dispo(4)=(1,1).

Après a_4, on ne peut pas lancer

- ni $T_1(2)$ car $Besoin_{1,1}(4) = 2 > Dispo_1(4) = 1$
- ni $T_2(2)$ car $Besoin_{2,1}(4) = 2 > Dispo_1(4) = 1$

D'où $D = \{1, 2\}$

Montrons que pour tout $T_i, i \in \{1,2\}$:

$\exists\ j$ *tel que* $Besoin_{i,j}(4) = 2 > Dispo_j(4) + \sum_{i \notin D} Tenu_{i,j}(4)$

avec $\sum_{i \notin D} Tenu_i(4) = (0,0)$

Pour :

- T_1 car $Besoin_{1,1}(4) = 2 > Dispo_1(4) + 0 = 1$

donc on a un interblocage.

Montrons que pour tout $T_i, i \in \{1,2\}$:

$\exists\, j$ *tel que* $Besoin_{i,j}(4) = 2 > Dispo_j(4) + \sum_{i \notin D} Tenu_{i,j}(4)$

avec $\sum_{i \notin D} Tenu_i(4) = (0,0)$

Pour :

- T_1 car $Besoin_{1,1}(4) = 2 > Dispo_1(4) + 0 = 1$
- T_2 car $Besoin_{2,1}(4) = 2 > Dispo_1(4) + 0 = 1$

donc on a un interblocage.

Évitement
Etat sûr

Soit un comportement $w = a_1 a_2 \cdots a_k$ partiel de S. L'état s_k est dit **sûr** s'il existe un comportement valide complet du système commençant par w.
Dans l'exemple précédent :

- s'_1 et s'_2 sont sûrs car on peut facilement montrer que le mot
  $w = d_1(1)f_1(1)d_3(1)f_3(1)d_1(2)f_1(2)d_2(1)f_2(1)d_2(2)f_2(2)$
  est un comportement valide.

Évitement
Etat sûr

Soit un comportement $w = a_1 a_2 \cdots a_k$ partiel de S. L'état s_k est dit **sûr** s'il existe un comportement valide complet du système commençant par w.
Dans l'exemple précédent :

- s'_1 et s'_2 sont sûrs car on peut facilement montrer que le mot
  $w = d_1(1)f_1(1)d_3(1)f_3(1)d_1(2)f_1(2)d_2(1)f_2(1)d_2(2)f_2(2)$
  est un comportement valide.
- par contre $S'_3 = \left(\begin{pmatrix} 2 & 0 \\ 0 & 0 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 0 \end{pmatrix} \right)$ n'est pas un état sûr. (Dispo(3)=(1,1).

Évitement
Etat sûr

- s'_3 n'est pas un blocage : les processus en attente de ressources sont $T_1(2)$ et $T_3(1)$. Or $Dispo(3) + tenu_2(3) = (2,2)$. Comme $Besoin_1(3) \leq (2,2)$ et $Besoin_3(3) \leq (2,2)$, s'_3 n'est pas un blocage

Évitement
Etat sûr

- s'_3 n'est pas un blocage : les processus en attente de ressources sont $T_1(2)$ et $T_3(1)$. Or $Dispo(3) + tenu_2(3) = (2,2)$. Comme $Besoin_1(3) \leq (2,2)$ et $Besoin_3(3) \leq (2,2)$, s'_3 n'est pas un blocage
- les seules transitions permises à partir de cet état s'_3 sont :

Évitement
Etat sûr

- s'_3 n'est pas un blocage : les processus en attente de ressources sont $T_1(2)$ et $T_3(1)$. Or $Dispo(3) + tenu_2(3) = (2,2)$. Comme $Besoin_1(3) \leq (2,2)$ et $Besoin_3(3) \leq (2,2)$, s'_3 n'est pas un blocage
- les seules transitions permises à partir de cet état s'_3 sont :
  - (a) la transition correspondant à $f_2(1)$ qui mène à l'état s'_4 est un blocage ;

Évitement
Etat sûr

- s'_3 n'est pas un blocage : les processus en attente de ressources sont $T_1(2)$ et $T_3(1)$. Or $Dispo(3) + tenu_2(3) = (2,2)$. Comme $Besoin_1(3) \leq (2,2)$ et $Besoin_3(3) \leq (2,2)$, s'_3 n'est pas un blocage
- les seules transitions permises à partir de cet état s'_3 sont :
  - (a) la transition correspondant à $f_2(1)$ qui mène à l'état s'_4 est un blocage ;
  - (b) la transition correspondant à $d_3(1)$ menant à $S''_4 = \left(\begin{pmatrix} 2 & 0 \\ 0 & 0 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 0 \end{pmatrix} \right)$ à partir duquel seuls $f_2(1)$ et $f_3(1)$ sont possibles.

Évitement
Etat sûr

Or :

- $f_2(1)$ mène à $S_5'' = \left(\begin{pmatrix} 2 & 0 \\ 2 & 2 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 0 \end{pmatrix} \right)$ qui est un interblocage car $Dispo(5) + tenu_3(5) = (0,1) + (1,0) = (1,1)$ et ni $Besoin_1(5)$ et $Besoin_2(5)$ n'est **inférieur** à (1,1) donc on a un interblocage avec $D = \{1,2\}$

Or :

- $f_2(1)$ mène à $S_5'' = \left(\begin{pmatrix} 2 & 0 \\ 2 & 2 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 0 \end{pmatrix} \right)$ qui est un interblocage car $Dispo(5) + tenu_3(5) = (0,1) + (1,0) = (1,1)$ et ni $Besoin_1(5)$ et $Besoin_2(5)$ n'est **inférieur** à (1,1) donc on a un interblocage avec $D = \{1,2\}$
- $f_3(1)$ mène à $S_5'' = \left(\begin{pmatrix} 2 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 0 \end{pmatrix} \right)$ dont la seule transition est $f_2(1)$ qui mène à $S_6'' = \left(\begin{pmatrix} 2 & 0 \\ 2 & 2 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 0 \end{pmatrix} \right)$ qui est un blocage car $Dispo(6) + tenu_3(6) = (1,1) + (0,0) = (1,1)$ et ni $Besoin_1(6)$ et $Besoin_2(6)$ n'est **inférieur** à (1,1) donc on

Évitement
Algorithme du banquier

Principe Pour affecter les ressources de s_k à s_{k+1}, on vérifie d'abord que la transition est sûre. Or :

- Si cette transition correspond à une fin de tâche : pas de problème

Problème : Cela nécessite de connaître à priori toutes les demandes et libérations de ressources de chacune des chaînes de tâches. En pratique (sauf tâches "répétitives") cela n'est pas réaliste. Par contre, on peut envisager connaître les MAXIMUM de chaque type de ressources demandées par les processus P_i.

Évitement
Algorithme du banquier

Principe Pour affecter les ressources de s_k à s_{k+1}, on vérifie d'abord que la transition est sûre. Or :

- Si cette transition correspond à une fin de tâche : pas de problème
- Si elle correspond à une demande : test de la transition. Pour cela, l'algorithme teste si s_{k+1} est un état sûr en cherchant une séquence de complétion valide du système pour l'état s_{k+1}.

Problème : Cela nécessite de connaître à priori toutes les demandes et libérations de ressources de chacune des chaînes de tâches. En pratique (sauf tâches "répétitives") cela n'est pas réaliste. Par contre, on peut envisager connaître les MAXIMUM de chaque type de ressources demandées par les processus P_i.

Soit MAX la matrice $n \times m$ représentant ces maximums : $MAX_{i,j}$ représente donc le maximum d'unités de R_j qu'une tâche de P_i peut demander.

Dans l'exemple précédent : $MAX = \begin{pmatrix} 3 & 2 \\ 3 & 3 \\ 1 & 0 \end{pmatrix}$

A l'état $s_k = [Besoin(k), Tenu(k)]$, l'algorithme pour décider si une tâche peut démarrer, se place dans le cas le plus défavorable. Définissons l'état fictif $t_k = [Besoin^t(k), Tenu^t(k)]$ correspondant donc à l'état s_k dans lequel toutes les demandes des tâches de chacun des processus P_i sont égales à $MAX_i - Tenu_i(k)$ par : $Tenu^t(k) = Tenu^(k)$

$$Besoin^t(k) = \begin{cases} MAX_i - Tenu_i(k) \; si \; Tenu_i(k) + Besoin_i(k) > \\ 0 \; sinon \end{cases}$$

Remarque 1 $\forall\ i\ Besoin_i(k) \leq Besoin_i^t(k)$
Remarque 2 Si t_k n'est pas un blocage alors
$\forall\ i\ Besoin_i^t(k) \leq N - \sum_{i \in D} Tenu_i^t(k)$ est vrai d'où s_k est sûr.

L'algorithme

▷ Pour passer de l'état s_{k-1} à s_k par une transition $demande_i(l)$:

- **1.** on calcule l'état s_k résultat de cette transition

▷ Pour passer de l'état s_{k-1} à s_k par une transition $libere_i(l)$ sans condition.

L'algorithme

▷ Pour passer de l'état s_{k-1} à s_k par une transition $demande_i(l)$:

- **1.** on calcule l'état s_k résultat de cette transition
- **2.** on construit l'état t_k associé à s_k

▷ Pour passer de l'état s_{k-1} à s_k par une transition $libere_i(l)$ sans condition.

L'algorithme

▷ Pour passer de l'état s_{k-1} à s_k par une transition $demande_i(l)$:

- **1.** on calcule l'état s_k résultat de cette transition
- **2.** on construit l'état t_k associé à s_k
- **3.** si t_k n'est pas un blocage : s_k est sûr, on accepte la transition
  SINON on ne peut conclure : la requête est rejetée.

▷ Pour passer de l'état s_{k-1} à s_k par une transition $libere_i(l)$ sans condition.

Exemple

Rappelons les demandes

Processus: P_i	tache: $T_i(j)$	demande$_i(j)$	libere$_i(j)$
P1	T1(1) T1(2)	(1,2) (2,0)	(0,1) (3,1)
P2	T2(1) T2(2)	(1,1) (2,2)	(0,0) (3,3)
P3	T3(1)	(1,0)	(1,0)

d'où $MAX = \begin{pmatrix} 3 & 2 \\ 3 & 3 \\ 1 & 0 \end{pmatrix}$.

Rappelons

$$w = d_1(1)f_1(1)d_2(1)f_2(1)d_1(2)f_1(2)d_2(2)f_2(2)d_3(1)f_3(1)$$

Exemple

$$S_0 = \left(\begin{pmatrix} 1 & 2 \\ 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \right);$$

$$S'_1 = \left(\begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 2 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \right);$$

$$S'_2 = \left(\begin{pmatrix} 2 & 0 \\ 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \right);$$

Dispo(1)=(2,1), Dispo(2)=(2,2).

Exemple

▷ Montrons que s'_2 est sûr : $t'_2 = \left(\begin{pmatrix} 2 & 1 \\ 3 & 3 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \right)$.

$D = \{1, 2\}$ or l'inégalité $Besoin_1^t(2) = (2, 1) \leq$

$N - \sum_{i \in D} Tenu_i^t(3) = (3, 3) - (2, 2) = (1, 1)$ est VRAIE d'où

t_k n'est pas un inter-blocage d'où s'_2 est sûr.

Exemple

▷ La transition due à $d_2(1)$ est elle sûre ?

$s'_3 = \left(\begin{pmatrix} 2 & 0 \\ 0 & 0 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 0 \end{pmatrix} \right)$; Dispo(3)=(1,1) or

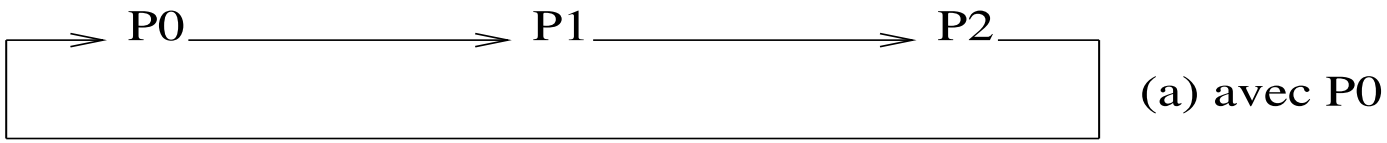
$t'_3 = \left(\begin{pmatrix} 2 & 1 \\ 2 & 2 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 0 \end{pmatrix} \right)$ est un blocage car $D = \{1, 2\}$

et les inégalités $Besoin_1^t(3) = (2,1) \leq N - \sum_{i \in D} Tenu_i^t(3) =$ $(3,3) - (2,2) = (1,1)$ et $Besoin_2^t(3) = (2,1) \leq (1,1)$ sont FAUSSES, d'où t'_3 est un blocage d'où la transition n'est pas autorisée.

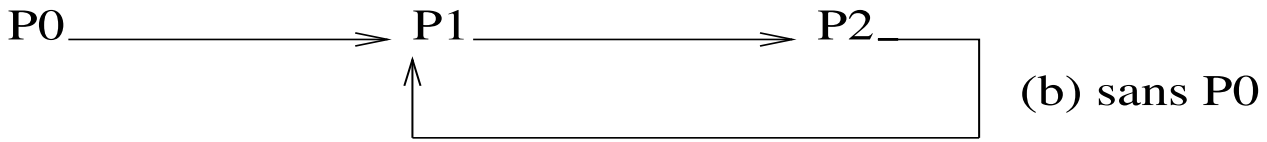
Détection
Principe

Un processus en attente (bloqué) sur une ressource va initier une détection.
On suppose qu'il y a N_p processus répartis sur N_s sites $S_1, \cdots, S_{N_s}$. Sur chaque de ces sites S_r, un contrôleur de ressources C_r a une vue locale de "ses" processus en termes de ressources : il sait pour chacun des processus locaux, les ressources que celui-ci demande et les processus qui les détiennent (même si ces processus sont sur d'autres sites). On dira que le contrôleur mémorise les dépendances de tous ses processus.
Notons $P_i \longrightarrow P_j$ le fait que P_i dépend de P_j (P_i attend des ressources détenues par P_j). Supposons que P_0 initie la détection.
Plusieurs possibilités d'inter-blocage :

1. local

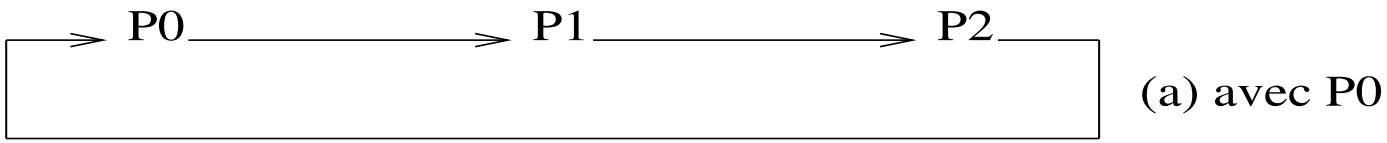


(a) avec P0

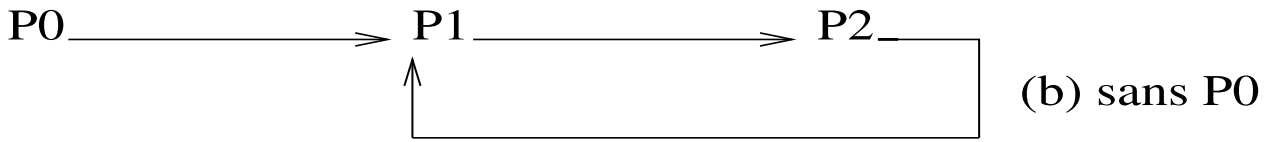


(b) sans P0

1. local



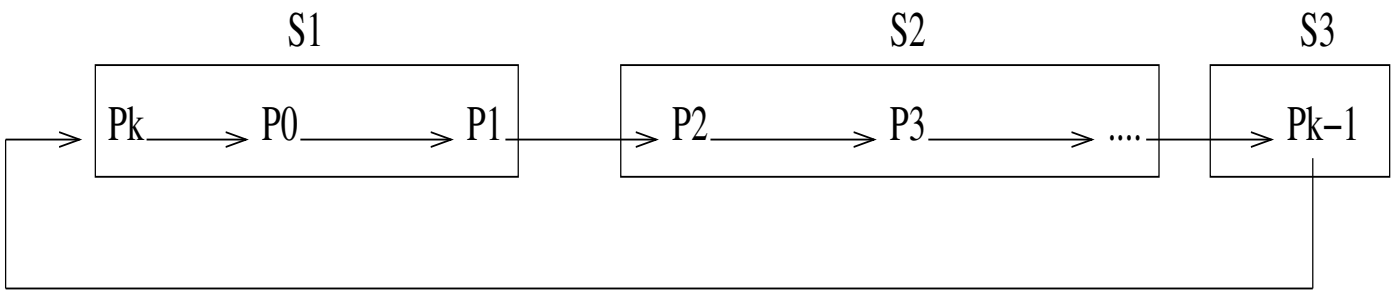
(a) avec P0



(b) sans P0

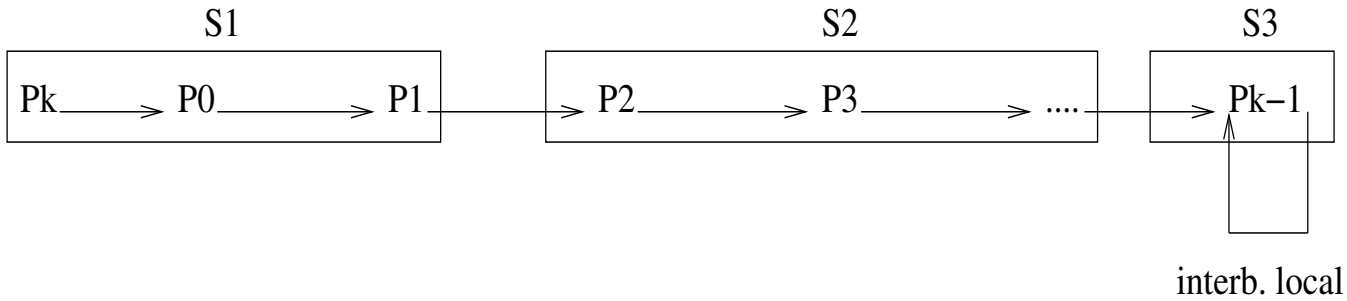
2. avec d'autres sites

(a) avec P0:

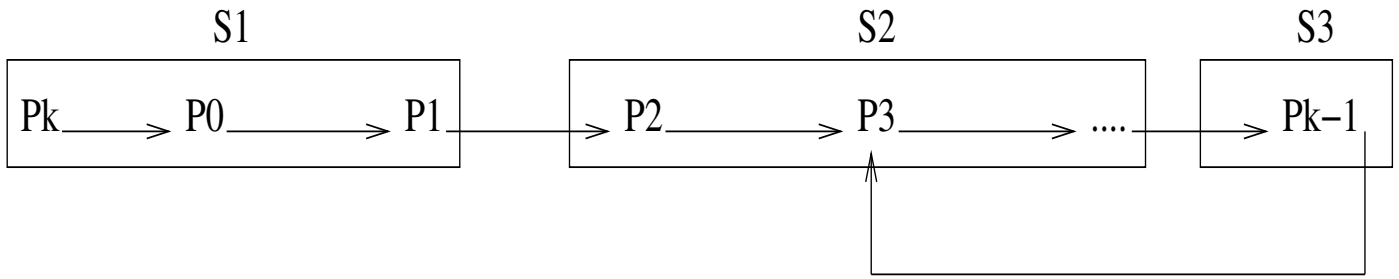


2. avec d'autres sites

(b) sans P0 mais avec du "local"

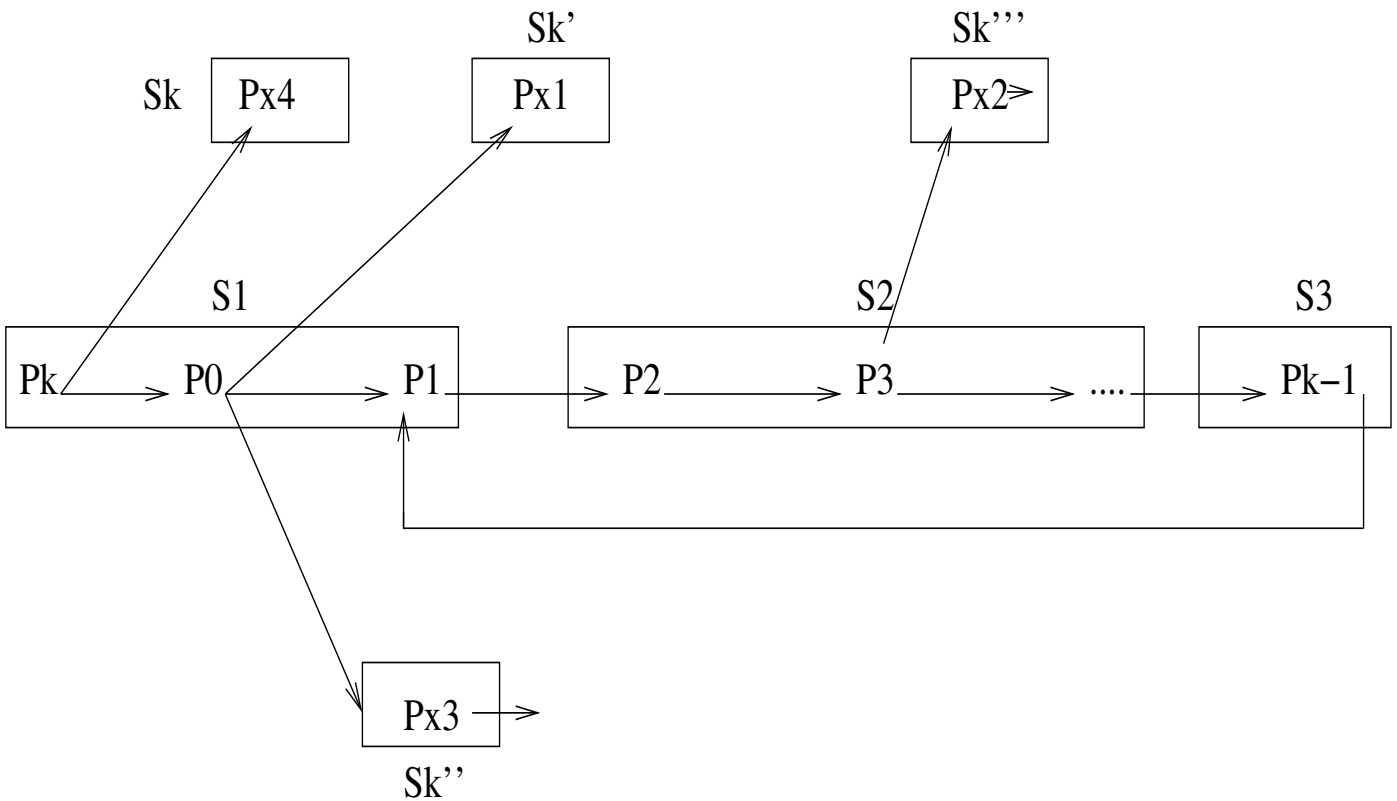


(c) sans P0 et sans "local"



Détection
Principe

Chacun des processus peut lui-même dépendre de plusieurs autres. Ainsi par exemple, nos pouvons avoir :



Principe Soit P_0 le site initiateur.
Un processus désirant détecter un interblocage, va demander à son contrôleur s'il existe un blocage local. Chacun des processus peut lui-même dépendre de plusieurs autres. Ainsi par exemple, nos pouvons avoir :

- si OUI (cas 1a et 1b) :

Principe Soit P_0 le site initiateur.
Un processus désirant détecter un interblocage, va demander à son contrôleur s'il existe un blocage local. Chacun des processus peut lui-même dépendre de plusieurs autres. Ainsi par exemple, nos pouvons avoir :

- si OUI (cas 1a et 1b) :
  - si c'est P_0 : FIN de l'algorithme, l'application est bloquée

Principe Soit P_0 le site initiateur.
Un processus désirant détecter un interblocage, va demander à son contrôleur s'il existe un blocage local. Chacun des processus peut lui-même dépendre de plusieurs autres. Ainsi par exemple, nos pouvons avoir :

- si OUI (cas 1a et 1b) :
  - si c'est P_0 : FIN de l'algorithme, l'application est bloquée
  - sinon : interblocage $\Longrightarrow$ proclamation

Principe Soit P_0 le site initiateur.
Un processus désirant détecter un interblocage, va demander à son contrôleur s'il existe un blocage local. Chacun des processus peut lui-même dépendre de plusieurs autres. Ainsi par exemple, nos pouvons avoir :

- si OUI (cas 1a et 1b) :
  - si c'est P_0 : FIN de l'algorithme, l'application est bloquée
  - sinon : interblocage $\Longrightarrow$ proclamation
- si NON

Principe Soit P_0 le site initiateur.
Un processus désirant détecter un interblocage, va demander à son contrôleur s'il existe un blocage local. Chacun des processus peut lui-même dépendre de plusieurs autres. Ainsi par exemple, nos pouvons avoir :

- si OUI (cas 1a et 1b) :
  - si c'est P_0 : FIN de l'algorithme, l'application est bloquée
  - sinon : interblocage $\Longrightarrow$ proclamation
- si NON
  - s'il ne dépend d'aucun processus distant

Principe Soit P_0 le site initiateur.
Un processus désirant détecter un interblocage, va demander à son contrôleur s'il existe un blocage local. Chacun des processus peut lui-même dépendre de plusieurs autres. Ainsi par exemple, nos pouvons avoir :

- si OUI (cas 1a et 1b) :
  - si c'est P_0 : FIN de l'algorithme, l'application est bloquée
  - sinon : interblocage $\Longrightarrow$ proclamation
- si NON
  - s'il ne dépend d'aucun processus distant
    - si c'est P_0 : FIN de l'algorithme, l'application n'est pas bloquée

Principe Soit P_0 le site initiateur.
Un processus désirant détecter un interblocage, va demander à son contrôleur s'il existe un blocage local. Chacun des processus peut lui-même dépendre de plusieurs autres. Ainsi par exemple, nos pouvons avoir :

- si OUI (cas 1a et 1b) :
  - si c'est P_0 : FIN de l'algorithme, l'application est bloquée
  - sinon : interblocage $\Longrightarrow$ proclamation
- si NON
  - s'il ne dépend d'aucun processus distant
    - si c'est P_0 : FIN de l'algorithme, l'application n'est pas bloquée
    - il va répondre au site demandeur "non interbloqué"

Principe Soit P_0 le site initiateur.
Un processus désirant détecter un interblocage, va demander à son contrôleur s'il existe un blocage local. Chacun des processus peut lui-même dépendre de plusieurs autres. Ainsi par exemple, nos pouvons avoir :

- si OUI (cas 1a et 1b) :
  - si c'est P_0 : FIN de l'algorithme, l'application est bloquée
  - sinon : interblocage $\Longrightarrow$ proclamation
- si NON
  - s'il ne dépend d'aucun processus distant
    - si c'est P_0 : FIN de l'algorithme, l'application n'est pas bloquée
    - il va répondre au site demandeur "non interbloqué"
  - sinon il va émettre vers les processus distants dont il dépend une demande de détection

Principe Soit P_0 le site initiateur.
Lorsqu'un processus reçoit :

- une demande de détection initiée par lui-même, il détecte l'interblocage (cas 2a) $\Longrightarrow$ proclamation

La proclamation va consister à envoyer au processus demandeur la réponse "interblocage"

Principe Soit P_0 le site initiateur.
Lorsqu'un processus reçoit :

- une demande de détection initiée par lui-même, il détecte l'interblocage (cas 2a) $\Longrightarrow$ proclamation
- une demande de détection alors qu'il a déjà été détecté l'interblocage $\Longrightarrow$ re-proclamation (car cela peut venir d'une autre chaîne de processus)

La proclamation va consister à envoyer au processus demandeur la réponse "interblocage"

Principe Soit P_0 le site initiateur.
Lorsqu'un processus reçoit :

- une demande de détection initiée par lui-même, il détecte l'interblocage (cas 2a) $\Longrightarrow$ proclamation
- une demande de détection alors qu'il a déjà été détecté l'interblocage $\Longrightarrow$ re-proclamation (car cela peut venir d'une autre chaîne de processus)
- une demande de détection alors qu'il n'a pas détecté d'interblocage $\Longrightarrow$ il mémorise le processus demandeur et relance l'algorithme s'il ne l'a pas déjà lancé

La proclamation va consister à envoyer au processus demandeur la réponse "interblocage"

Détection
Algorithme de détection

Lorsqu'un processus reçoit :

- une proclamation et que c'est P_0, il détecte l'interblocage (cas 2b et 2c) : FIN de l'algorithme $\Longrightarrow$ l'application est bloquée.

Lorsqu'un processus a reçu une réponse "non interbloqué" de tous les processus dont il dépend

Détection
Algorithme de détection

Lorsqu'un processus reçoit :

- une proclamation et que c'est P_0, il détecte l'interblocage (cas 2b et 2c) : FIN de l'algorithme $\Longrightarrow$ l'application est bloquée.
- une proclamation suite à une demande de détection, il mémorise l'interblocage $\Longrightarrow$ proclamation

Lorsqu'un processus a reçu une réponse "non interbloqué" de tous les processus dont il dépend

Détection
Algorithme de détection

Lorsqu'un processus reçoit :

- une proclamation et que c'est P_0, il détecte l'interblocage (cas 2b et 2c) : FIN de l'algorithme $\Longrightarrow$ l'application est bloquée.
- une proclamation suite à une demande de détection, il mémorise l'interblocage $\Longrightarrow$ proclamation

Lorsqu'un processus a reçu une réponse "non interbloqué" de tous les processus dont il dépend

- si ce processus est P_0, FIN de l'algorithme $\Longrightarrow$ l'application est bloquée.

Détection
Algorithme de détection

Lorsqu'un processus reçoit :

- une proclamation et que c'est P_0, il détecte l'interblocage (cas 2b et 2c) : FIN de l'algorithme $\Longrightarrow$ l'application est bloquée.
- une proclamation suite à une demande de détection, il mémorise l'interblocage $\Longrightarrow$ proclamation

Lorsqu'un processus a reçu une réponse "non interbloqué" de tous les processus dont il dépend

- si ce processus est P_0, FIN de l'algorithme $\Longrightarrow$ l'application est bloquée.
- sinon il émet la réponse "non interbloqué" au(x) processus demandeur(s).

"Code" de l'algorithme on suppose définies dans chaque processus P_i sur S_r les deux fonctions locales suivantes : ▷ $Test\_Local(P_i, C_r)$: booléen
/* Demande à C_r si P_i est dans une attente circulaire locale : retourne VRAI si P_i est dans une attente circulaire locale, FAUX sinon */
▷ $Dependance(P_i, C_r) : \{P_j\}$
/* Demande à C_r tous les processus P_j dont dépend P_i directement ou indirectement connus par C_r : c'est-à-dire ceux qui sont sur S_r (ex : P_1) plus ceux qui sont directement bloquants pour P_i et les P_j locaux (ex : P_2, P_{x1}, P_{x3} */

Initialisation de la détection :
▷ Detecter_interblocage(i) /* dans l'exemple i=0 */
{
si *Test_Local*(P_i, C_r) alors Detection terminée sur interblocage
sinon {
pour tout j et $k \in$ *Dependance*(P_i, C_r) tel que P_j soit sur le même site que P_i mais pas P_k
{ envoyer ((*Test_global*, $i, j, \{n_1, n_2, \cdots ; \}$)) à P_k ; }
}
Si une des réponses est VRAI alors Détection terminée sur interblocage
sinon blocage normal }

Réception d'un message de test par P_c sur le site S_{r2} :
$\triangleright$ *Reception*((*Test_global*, $a, b, \{n_1, n_2, \cdots ;\}$))
sinon $\{$
si c apparait dans la liste $\{n_1, n_2, \cdots ;\}$ alors retourner VRAI à P_a /* FIN */
si *Test_Local*(P_c, C_{r2}) alors retourner VRAI à P_a /* FIN */
pour tout j' *et* $k' \in$ *Dependance*(P_c, C_{r2}) tel que P_j soit sur le même site que P_c mais pas P_k
$\{$ envoyer
((*Test_global*, $c, j', \{n_1, n_2, \cdots ;\} \cup \{n'_1, n'_2, \cdots ;\}$)) à $P_k\}$
/* $n'_1, n'_2, \cdots$: liste des processus dans la chaîne de dépendance de P_c à P_j */
Si une des réponses est VRAI alors retourner VRAI à P_a ;
/* FIN */
sinon retourner FAUX à P_a
$\}$

- Qui lance la détection ?

Ce problème que l'on retrouve dans de nombreux algorithmes en systèmes distribués (terminaison, élection,...) peut être considéré comme "externe" au problème de la détection. D'une façon générale, il peut être réglé d'au moins deux manières différentes :

Détection
Problème lié à la détection

- Qui lance la détection ?

Ce problème que l'on retrouve dans de nombreux algorithmes en systèmes distribués (terminaison, élection,...) peut être considéré comme "externe" au problème de la détection. D'une façon générale, il peut être réglé d'au moins deux manières différentes :

- soit un processus particulier (faisant partie ou non) des processus concernés, est chargé de surveiller le bon déroulement de l'application.
  Ainsi, dans le cas de la détection d'interblocage, lorsqu'il considérera qu'il y a trop de processus en attente, il déclenchera la détection. cela pose bien évidemment le problème de comment fait-il pour connaître le nombre de processus en attente et les temps d'attente respectifs.

- Qui lance la détection ?

- Qui lance la détection ?
- soit chaque processus est capable d'estimer lui-même s'il faut lancer l'algorithme.
  Ainsi, dans le cas de la détection d'interblocage, lorsqu'un P_i considérera que son blocage est anormal, il décidera de lui même déclencher la détection. Mais alors il y a le risque que plusieurs détections aient lieu en même temps.

Détection
Problème lié à la détection

Dans le cas de la détection d'interblocage de la section (principe), lorsqu'il existe une solution intermédiaire où ce sont les contrôleurs qui assurent cette fonction.

- Quel est le coût de l'algorithme ?

Dans tous les cas, de façon évidente, chaque occurrence de l'algorithme a un coût inférieur au nombre de sites :

Détection
Problème lié à la détection

Dans le cas de la détection d'interblocage de la section (principe), lorsqu'il existe une solution intermédiaire où ce sont les contrôleurs qui assurent cette fonction.

- Quel est le coût de l'algorithme ?

Dans tous les cas, de façon évidente, chaque occurrence de l'algorithme a un coût inférieur au nombre de sites :

- dans le cas 1 : il n'y a qu'une concurrence à la fois mais le coût est reporté sur la surveillance des processus

Détection
Problème lié à la détection

Dans le cas de la détection d'interblocage de la section (principe), lorsqu'il existe une solution intermédiaire où ce sont les contrôleurs qui assurent cette fonction.

- Quel est le coût de l'algorithme ?

Dans tous les cas, de façon évidente, chaque occurrence de l'algorithme a un coût inférieur au nombre de sites :

- dans le cas 1 : il n'y a qu'une concurrence à la fois mais le coût est reporté sur la surveillance des processus
- dans le cas 2 : au pire des cas, le nombre d'occurrences peut être égal au nombre total de processus dans l'application

Détection
Problème lié à la détection

Dans le cas de la détection d'interblocage de la section (principe), lorsqu'il existe une solution intermédiaire où ce sont les contrôleurs qui assurent cette fonction.

- Quel est le coût de l'algorithme ?

Dans tous les cas, de façon évidente, chaque occurrence de l'algorithme a un coût inférieur au nombre de sites :

- dans le cas 1 : il n'y a qu'une concurrence à la fois mais le coût est reporté sur la surveillance des processus
- dans le cas 2 : au pire des cas, le nombre d'occurrences peut être égal au nombre total de processus dans l'application
- dans le cas 3 : au pire des cas, le nombre d'occurrences peut être égal au nombre de sites

- Que fait-on après la détection ?

Il n'y a aucune solution miracle générale. Tout va dépendre de l'application : quel(s) processus tuer ? quelle(s) ressource(s) libérer ? etc.