CONDITIONS D'APPARITION DE L'INTERBLOCAGE

1. Accès exclusif à chaque ressource

2. Les clients (en général, les processus) qui demandent de nouvelles ressources
- **gardent celles qu'ils ont déjà acquises**
- **attendent l'arrivée de leur demande**

3. Pas de réquisition possible des ressources déjà allouées

4. Les clients en attente des ressources déjà allouées forment une chaîne circulaire d'attente.

EXEMPLE : ACCÈS EXCLUSIF À DES FICHIERS F ET G

TRANSACTION 1		TRANSACTION 2	TRANSACTION 3
•		•	•
•		•	•
ouvrir F en écriture		•	•
•		**ouvrir G en écriture**	•
•		•	<== **ouvrir G en écriture**
•		•	•
ouvrir G en écriture	<=>	**ouvrir F en écriture**	**ouvrir F en écriture**
•		•	•
fermer F et G		**fermer F et G**	**fermer F et G**

EXEMPLE : allocation dynamique de mémoire avec 16 cases partageables entre trois processus

PROCESSUS 1	PROCESSUS 2	PROCESSUS 3
•	•	•
•	•	**demande 4 cases**
demande 3 cases	**demande 4 cases**	•
•	•	**demande 2 cases**
demande 2 cases	•	•
•	**demande 1 cases**	•
•	•	•
•	•	•
demande 1 cases ~~~~~~~~~~	**demande 2 cases** ~~~~~~~~~~~	**demande 2 cases**

APPROCHES DU PROBLÈME

1. DÉTECTION GUÉRISON

détection complexe : recherche de cycle dans un graphe ($O(n^2)$)
guérison brutale : destruction de processus
ou coûteuse : points de reprise et retour arrière

2. PRÉVENTION STATIQUE

DEMANDE GLOBALE : tout processus doit demander, acquérir et rendre globalement ses ressources
mauvaise utilisation des ressources

CLASSES ORDONNÉES : classes de ressources ordonnées a priori
1. demande selon l'ordre des classes
2. avec demande globale pour chaque classe
meilleure utilisation des ressources si l'ordre est bien choisi
exemple de OS/MV1 : fichiers -> mémoire -> périphériques

3. PRÉVENTION DYNAMIQUE (ÉVITEMENT) par annonce des ressources et attente forcée éventuelle

• Tout processus annonce son besoin maximum (exact ou non)
• L'allocateur se ménage toujours la possibilité de répondre au besoin maximum de chaque processus en bloquant momentanément certaines demandes de ressources (évolution d'état fiable à état fiable)
- **contraintes sur l'allocation, même s'il y a assez de ressources pour la requête**
- **plus ou moins bonne utilisation des ressources**
- **algorithme coûteux ($O(n^2)$)**

4. PRÉVENTION DYNAMIQUE par estampillage des transactions et abandon forcé éventuel :
- **ordre total pour régler les conflits potentiel**

APPROCHE DU COURS

• comprendre le phénomène : modèles descriptifs

• agir : modèle amenant des algorithmes

INTERBLOCAGE DANS LES SYSTÈMES TRANSACTIONNELS

NOTION DE TRANSACTION INDIVISIBLE

l'exécution d'une transaction se termine • soit par la validation • soit par l'abandon

OBJETS PARTAGÉS ENTRE TRANSACTIONS

la pose dynamique de verrous ("locks") peut entraîner l'interblocage
méthodes de prévention :
- **statique par classes ordonnées : verrouiller les objets partagés dans le même ordre**
- **dynamique par transactions estampillées pour définir un ordre total entre elles**

PRÉVENTION PAR ESTAMPILLAGE DES TRANSACTIONS

- **relation d'ordre total avec estampille unique pour chaque transaction**
- **en cas de conflit, choix selon la relation d'ordre :**

exemple : (règle Wait-Die) la transaction est autorisée à attendre si son estampille est inférieure à celle des transactions responsables de son attente, sinon elle est abandonnée. Une transaction abandonnée redémarre avec son ancienne estampille (absence de famine)

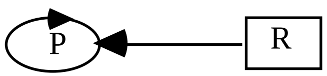
EXEMPLE DE CONFLIT entre Ti et Tj : **Tj (estampille ej) a verrouillé une ressource R**
Ti (estampille ei) demande la ressource R

- **si ei < ej, on dit que Ti est plus ancienne que Tj alors Ti attend**
- **si ei > ej, on dit que Ti est plus récente que Tj alors Ti est abandonnée**

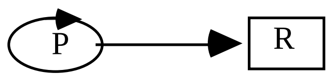
(en effet Ti a peut être déjà verrouillé la ressource S que Tj risque de demander, en créant alors un interblocage)

- **On dit que c'est une méthode pessimiste car Ti est abandonnée même si elle n'utilise aucune ressource**
- **S'applique aussi aux transactions réparties sur plusieurs sites**

GRAPHE DE L'INTERBLOCAGE



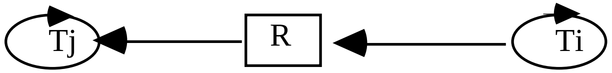
le processus P détient la ressource R



le processus P demande la ressource R

APPLICATION À L'EXEMPLE DES DEUX FICHIERS

TRANSACTION 1 (estampille e1)		TRANSACTION 2 (estampille e2)		TRANSACTION 3 (estampille e3)
•		•		•
ouvrir F en écriture		•		•
•		**ouvrir G en écriture**		•
•		•	<==	**ouvrir G en écriture**
•		•		•
ouvrir G en écriture	<=>	**ouvrir F en écriture**		**ouvrir F en écriture**
fermer F et G		**fermer F et G**		**fermer F et G**



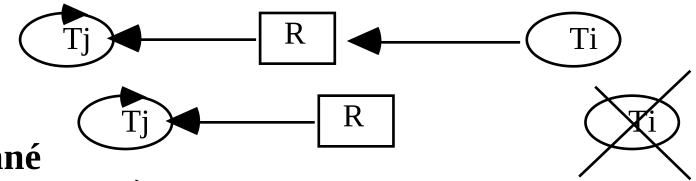
la transaction Tj détient la ressource R
la transaction Ti demande la ressource R

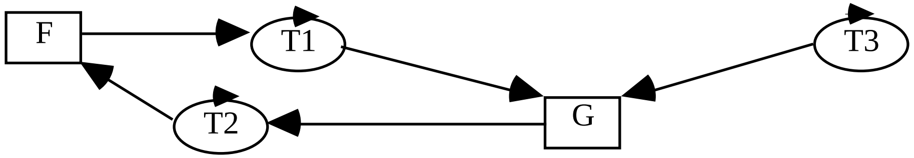
ei < ej

Ti attend

ei > ej

Ti est abandonné

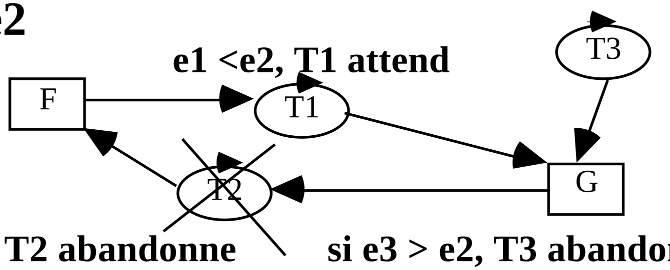




interblocage dans l'exemple avec deux fichiers F et G

e1 < e2

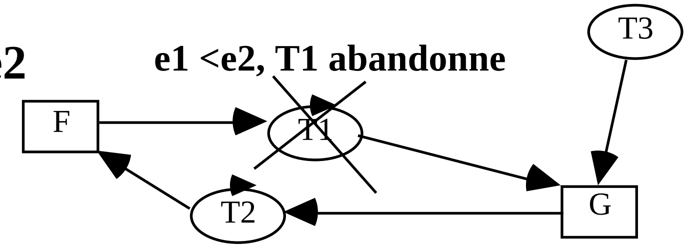
e1 <e2, T1 attend



e2 > e1 , T2 abandonne **si e3 > e2, T3 abandonne**

e1 > e2

e1 <e2, T1 abandonne



e2 > e1 , T2 attend **si e3 > e2, T3 abandonne**

FORMALISATION DE L'INTERBLOCAGE

SYSTÈME INFORMATIQUE = (P, E, X, A, D)

P = {P1, ... , Pn} l'ensemble fixe des processus
E = {R1, ... ,Rm} l'ensemble des classes de ressources
X : état initial des ressources
A(t) : matrice d'allocation des ressources aux processus à la date t
D(t) : matrice des demandes cumulées de ressources à la date t
R(t) =X - ΣAi(t) résidu, ressources disponibles à la date t

$$X = \begin{vmatrix} x_1 \\ : \\ x_m \end{vmatrix} \qquad Ai(t) = \begin{vmatrix} a_{i1}(t) \\ : \\ a_{im}(t) \end{vmatrix} \qquad Di(t) = \begin{vmatrix} d_{i1}(t) \\ : \\ d_{im}(t) \end{vmatrix}$$

A(t) = (A1(t), ... , An(t)) **D(t) = (D1(t), ... , Dn(t))**

C : matrice des annonces maximales des processus
Dist(t) = C - A(t) : distance par rapport à la demande maximale

HYPOTHÈSES

1. n et m constantes, |P|, |E|, |X| de taille fixe
2. un seul utilisateur par ressource (utilisation en exclusion mutuelle)
3. dans une classe, ressources banalisée
4. les ressources sont restituées au bout d'un temps fini
5. on peut faire attendre une requête quand D(t) > A(t)

NOTATIONS

a) U, V vecteurs à m éléments
$U \le V \Leftrightarrow u_i \le v_i \quad \forall i \in [1..m]$
$U < V \Leftrightarrow (U \le V)$ et $(\exists\, i$ tel que $u_i < v_i)$

b) M, N matrices à m ∗ n éléments
$M \le N \Leftrightarrow M_j \le N_j \quad \forall j \in [1..n]$
$M \le N \Leftrightarrow M_j \le N_j$ et $(\exists\, j$ tel que $M_j < N_j)$

ÉTATS DU SYSTÈME

ÉTAT RÉALISABLE	**1) la demande est réalisable : $Di(t) \le X$ pour tout processus i** **2) chaque processus reçoit au plus sa demande : $A(t) \le D(t)$** **3) la somme des demandes est réalisable : $\Sigma Ai(t) \le X$ ou encore $R(t) \ge 0$**
ÉTAT SAIN	**• on n'a pas détecté d'interblocage présent ou inéluctable à l'instant t** - - - - - - - - - - **• si et seulement si aucun processus n'est bloqué définitivement à partir de t** - - - - - - - - - - **• si on peut montrer une exécution séquentielle virtuelle qui permet d'élire tous les processus** <=> **• il faut <u>montrer une suite saine complête</u> de processus i_1, i_2,...i_n telle que :** **$rang(i_1) = 1,...,rang (i_n) = n$** **et telle que pour chaque processus i on ait la relation :** $$Di(t) - Ai(t) \le R(t) + \Sigma Aj(t)$$ **<u>pour 1 ≤ i ≤ n et pour chaque i : on somme tous les j tels que : rang(j) < rang (i)</u>**
ÉTAT FIABLE	**• Étant donné par contrat une annonce de demande max C = (C1, C2, ...,Cn)** - - - - - - - - - - **• pas d'interblocage détecté à l'instant t ni à craindre dans le futur** - - - - - - - - - - **• si on peut montrer qu'il existe une exécution séquentielle virtuelle qui permet d'élire tous les processus, même si au pire ils demandent tous leur annonce contractuelle (solution de survie)** < = > **• il faut <u>montrer une suite fiable complête</u> de processus i_1, i_2,...i_n telle que :** **$rang(i_1) = 1,...,rang (i_n) = n$** **et telle que pour chaque processus i on ait la relation :** $$Ci - Ai(t) \le R(t) + \Sigma Aj(t)$$ **<u>pour 1 ≤ i ≤ n et pour chaque i : on somme tous les j tels que : rang(j) < rang (i)</u>**

{ÉTAT RÉALISABLE} ⊇ {ÉTAT SAIN}⊇ {ÉTAT FIABLE}

ÉTAT SAIN <=> ABSENCE D'INTERBLOCAGE

DÉFINITIONS ET THÉORÈMES

STRATÉGIE POUR L'ALLOCATEUR : voir si l'allocateur peut satisfaire les requêtes actuelles des processus,
• en leur imposant une utilisation strictement séquentielle des ressources, à partir de l'état t actuel :
• à condition que chacun rende ses ressources au bout d'un temps fini
Il existe alors une voie de secours, possible, mais non obligatoire.

(D1) DÉFINITION: S une suite saine de processus i_1, i_2,...i_n telle que : rang(i_1) = 1,...,rang (i_n) = n,
est SAINE si et seulement si elle vérifie l'ensemble de relations

$$Di(t) - Ai(t) \le R(t) + \Sigma Aj(t)$$

pour $1 \le i \le n$ et pour chaque i : la somme des Aj(t) concerne tous les j tels que : rang(j) < rang (i)

soit

$$Di_1(t) - Ai_1(t) \le R(t)$$
$$Di_2(t) - Ai_2(t) \le R(t) + Ai_1(t)$$
$$Di_3(t) - Ai_3(t) \le R(t) + Ai_1(t) + Ai_2(t) \qquad \text{etc...}$$

ÉTAT SAIN : S contient tous les processus de P (suite saine complète)

THÉORÈME : si l'état est sain, il existe au moins une façon d'allouer les ressources sans interblocage dans cet état, c'est de sérialiser les processus selon l'ordre de la suite saine à partir de l'état t actuel

COROLLAIRE : s'il n'existe aucune suite saine complète, alors le système est en interblocage dans l'état t

THÉORÈME : Si on est dans un état sain, la satisfaction de la requête d'un processus k fait passer dans un nouvel état sain s'il existe, dans le nouvel état, une sous-suite saine qui contient ce processus k.

UTILISATION : chercher à placer k le plus tôt possible dans la suite saine. Dès que k est intégré dans la suite, le reste de la suite est saine (on n'a pas à le vérifier, donc on gagne du temps)

EXEMPLES POUR L'ÉTAT SAIN

1) m = 1 n = 2 X = 10

A(t) = (4, 5) R(t) = 1 D(t) = (6, 7) état réalisable, état non sain : non (Di(t) - Ai(t) ≤ R(t), ∀ i)

A(t') = (4, 5) R(t') = 1 D(t') = (5, 9) état sain : car on a la séquence P1P2

2) m = 1 n = 3 X = 10

A(t) = (1, 3, 5) R(t) = 1 D(t) = (2, 6, 8) état réalisable, état non sain

- **échec quand on recherche d'une exécution séquentielle**

P1 D1(t) - A1(t) ≤ R(t)

P2 ou P3 Di(t) - Ai(t) > R(t) + A1(t), ∀ i = 1,2

3) m =2 n = 2 $X = \begin{vmatrix} 1 \\ 1 \end{vmatrix}$

$A(t) = \begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix}$ $R(t) = \begin{vmatrix} 0 \\ 0 \end{vmatrix}$ $D(t) = \begin{vmatrix} 1 & 0 \\ 1 & 1 \end{vmatrix}$ **état sain car on peut proposer la séquence P2 P1**

$A(t') = \begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix}$ $R(t') = \begin{vmatrix} 0 \\ 0 \end{vmatrix}$ $D(t') = \begin{vmatrix} 1 & 1 \\ 1 & 1 \end{vmatrix}$ **état non sain**

EXEMPLE POUR L'ÉTAT FIABLE (voir plus loin)

4) m =2 n = 2 $X = \begin{vmatrix} 4 \\ 5 \end{vmatrix}$ $C = \begin{vmatrix} 4 & 3 \\ 3 & 4 \end{vmatrix}$

$A(t) = \begin{vmatrix} 1 & 1 \\ 1 & 2 \end{vmatrix}$ $R(t) = \begin{vmatrix} 2 \\ 2 \end{vmatrix}$ **état fiable car on peut proposer la séquence P2 P1**

C2 - A2(t) ≤ R(t)

C1 - A1(t) < R(t) + A2(t)

$A(t') = \begin{vmatrix} 2 & 1 \\ 1 & 2 \end{vmatrix}$ $R(t') = \begin{vmatrix} 1 \\ 2 \end{vmatrix}$ **état non fiable**

Exemple pour l'état sain. Allocation dynamique de mémoire avec 16 cases

m = 1 n = 3 X = (16)

Processus	P1			Processus	P2		Processus	P3
	:				:			:
D1 = (0)	A1 = (0)	:	------------t1	D2 = (0)	A2 = (0)	t1-----------t1	D3 = (0)	A3 = (0)
	:	:			:			:
D1 = (3)	A1 = (0)	:		D2 = (4)	A2 = (0)		D3 = (4)	A3 = (0)
D1 = (3)	A1 = (3)	:		D2 = (4)	A2 = (4)		D3 = (4)	A3 = (4)
	:	:			:			:
D1 = (5)	A1 = (3)	:		D2 = (5)	A2 = (4)		D3 = (6)	A3 = (4)
D1 = (5)	A1 = (5)	:	t3---------------.	D2 = (5)	A2 = (5)		D3 = (6)	A3 = (6)
	:	:	:		:			:
D1 = (6)	A1 = (5)	:	t4---------t4, t3	D2 = (7)	A2 = (5)	t3, t4----t4, t3	D3 = (8)	A3 = (6)
D1 = (6)	A1 = (6)	t1	t1 t2---------t2	D2 = (7)	A2 = (7)	t2-----------t2	D3 = (8)	A3 = (8)
	:				:			:
D1 = (0)	A1 = (6)			D2 = (0)	A2 = (7)		D3 = (0)	A3 = (8)
D1 = (0)	A1 = (0)			D2 = (0)	A2 = (0)		D3 = (0)	A3 = (0)

t1 : A(t1) = (6 0 0) R(t1) = (10) D(t1) = (6 0 0) réalisable et sain : toute séquence Pi Pj Pk est possible

t2 : A(t2) = (6 7 8) R(t2) = (-5) D(t2) = (6 7 8) non réalisable

t3 : A(t3) = (5 5 6) R(t3) = (0) D(t3) = (5 7 8) réalisable et sain : la séquence P1 P2 P3 est possible
la séquence P1 P3 P2 aussi

t4 : A(t4) = (5 5 6) R(t4) = (0) D(t4) = (6 7 8) réalisable mais non sain : interblocage

APPLICATIONS DE L'ÉTAT SAIN

PRÉVENTION STATIQUE PAR DEMANDE GLOBALE : Si tout processus demande et obtient globalement ses ressources, il existe toujours une suite saine complète S. Il n'y a jamais interblocage.

constituons la suite S de la sorte : S = S1, S0

Tout demandeur servi est mis dans S1, bloqué est mis dans S0

avec S1, la suite des Pi servis : $\forall$ Pi $\in$ S1, <=> Di - Ai(t) = 0
D1 est toujours vérifiée dans S1 : car $0 \leq R(t) + \Sigma Aj(t)$
avec S0, la suite des Pj bloqués : $\forall$ Pj $\in$ S0, <=> Aj(t) = 0
D1 est toujours vérifiée dans S0 : car $R(t) + \Sigma Ai(t) = X$ et $Di \leq X$

PRÉVENTION STATIQUE PAR CLASSES ORDONNÉES
tout processus doit demander
- **globalement les ressources dont il a besoin dans chaque classe**
- **les ressources des différentes classes dans l'ordre des classes**

Un processus ne peut acquérir des ressources de la classe i que s'il ne possède pas de ressources des classes j d'ordre supérieur telles que $j \geq i$

Soit m classes R1, R2, ... , Rm.
Il existe toujours une suite saine complète $S = S_m\ S_{m-1}...S_h...S_0$ Donc il n'y a jamais interblocage.

S0, la suite des processus qui n'ont reçu aucune ressource
S_h, la suite des processus qui ont reçu les ressources des classes 1 à h :
$P_k \in S_{\underline{h}}$ <=> $d_{jk} - a_{jk} = 0$ pour les ressources j telles que $j \leq h$
$a_{jk} = 0$ pour les ressources j telles que $j > h$

D1 est toujours vérifiée car
Sm est saine car $\forall$ Pk $\in$ Sm , Pk a tout reçu et alors Dk(t) - Ak(t) = 0
$S_m\ S_{m-1}$ est saine, $S_m\ S_{m-1}\ S_{m-2}$ est saine,..., $S_m\ S_{m-1}...S_0$ = S aussi

Tout nouveau demandeur peut être placé dans S_0. Lors d'une libération de ressource, un processus peut être déplacé de Sf vers Sg avec g > f.

ÉTAT FIABLE <=> PRÉVENTION DE L'INTERBLOCAGE

ANNONCE : tout processus déclare au préalable son besoin maximal C = (C1, C2, ... , Cn)

STRATÉGIE : voir si l'allocateur peut rester maître du jeu grâce à sa connaissance des annonces maximales de chacun des processus et grâce au fait que chaque processus doit rendre ses ressources au bout d'un temps fini

• L'allocateur peut refuser de servir certains processus et aller jusqu'à leur imposer à tous une exécution sérialisée.

Cette sérialisation est une voie de secours ultime, utilisable dans le cas le plus défavorable où chaque processus ne libère de ressources qu'après avoir reçu la totalité de son annonce.

On vérifie que cette voie de secours ultime existe toujours quand on passe dans un nouvel état.

ÉTAT RÉALISABLE à t : pour chaque processus i : $Ci \leq X$, $Ai(t) \leq Ci$, $Di(t) \leq Ci$

(D2) DÉFINITION : S une suite fiable de processus $i_1, i_2,...i_n$ telle que : $rang(i_1) = 1,...,rang(i_n) = n$, est FIABLE si et seulement si elle vérifie l'ensemble de relations

$$Ci - Ai(t) \leq R(t) + \Sigma Aj(t)$$

pour $1 \leq i \leq n$ et pour chaque i la somme des Aj(t) concerne tous les j tels que : rang(j) < rang (i)

soit

$$Ci_1 - Ai_1(t) \leq R(t)$$

$$Ci_2 - Ai_2(t) \leq R(t) + Ai_1(t)$$

$$Ci_3 - Ai_3(t) \leq R(t) + Ai_1(t) + Ai_2(t) \quad \text{etc...}$$

THÉORÈMES :

ÉTAT FIABLE : la suite S contient tous les processus de P (suite complète).

• toute suite fiable est une suite saine

• si l'état est fiable, il existe, même dans le cas où aucun processus ne libère de ressource avant d'avoir reçu toute son annonce, au moins une façon d'allouer les ressources sans interblocage : sérialiser, à partir de l'état t, les exécutions selon la suite fiable (réciproquement, s'il y a une suite fiable, on évite l'interblocage).

(T) THÉORÈME : Si on est dans un état fiable, la satisfaction de la requête d'un processus k fait passer dans un nouvel état fiable s'il existe une sous-suite fiable qui contient ce processus k.

UTILISATION : chercher à placer k le plus tôt possible dans la suite fiable. Dès que k est intégré dans la suite, le reste de la suite est fiable (on n'a pas à le vérifier, donc on gagne du temps)

EXEMPLE POUR L'ÉTAT FIABLE

m = 1 n = 6 X = (16) C = (8 7 10 9 12 10)

A(t1) = (5 5 4 0 0 0) R = (2)
C = (8 7 10 9 12 10)
Dist = (3 2 6 9 12 10)
• état fiable car la séquence P2 P1 P3 P4 P5 P6 est fiable

A(t2) = (5 4 6 0 0 0) R = (1)
C = (8 7 10 9 12 10)
Dist = (3 3 4 9 12 10)
• état pas fiable car on ne trouve pas de séquence fiable

A(t3) = (3 5 8 0 0 0) R = (0)
C = (8 7 10 9 12 10)
Dist = (5 2 2 9 12 10)
• état pas fiable car on ne trouve pas de séquence fiable

A(t4) = (1 5 1 1 4 2) R = (2)
C = (8 7 10 9 12 10)
Dist = (7 2 9 8 8 8)
• état fiable car P2 P1 P6 P5 P4 P3 est une séquence fiable

A(t5) = (0 5 2 1 4 2) R = (2)
C = (8 7 10 9 12 10)
Dist = (8 2 8 8 8 8)
• pas fiable car on ne peut prolonger P2 avec une séquence fiable
noter qu'avec X = (17) on obtiendrait un état fiable

APPLICATIONS DE L'ÉTAT FIABLE

PRÉVENTION DYNAMIQUE PAR ALGORITHME DU BANQUIER (Dijkstra, Habermann)

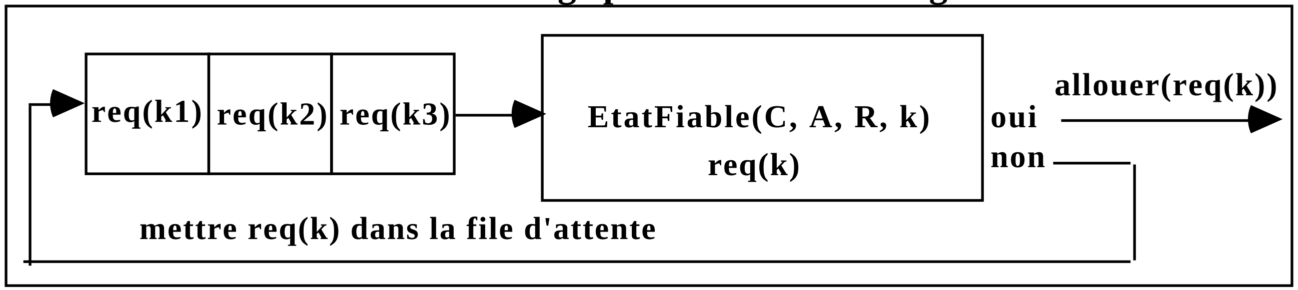
UTILISATION : l'allocateur de ressources ne peut servir une requête que si l'allocation correspondante laisse le système dans un état fiable. Dans la pratique, l'état initial est fiable. On conduit le système en l'obligeant à rester dans des états fiables, et cela en refusant ou en acceptant les requêtes des processus.

HYPOTHÈSE : on part d'un état fiable, on analyse si le service de la requête du processus Pk fait passer le système dans un nouvel état qui reste fiable. Si oui Pk peut être servi, sinon il doit attendre.
MÉTHODE : considérer le nouvel état et essayer de construire une sous-suite fiable contenant Pk

ÉTAT COURANT
• X nombre initial de ressources, annonce C = (C1, ... , Ck, ... , Cn)
ressources allouées A(t) = (A1, ... , Ak, ... , An), R(t) =X - ΣAi(t)

CAS D'UNE NOUVELLE REQUÊTE DE RESSOURCES
• requête req(k) > 0 d'où <u>nouvel état à tester</u>
A(t') = (A1, ... , Ak + req(k), ... , An) et R(t') = R(t) - req(k)
• if EtatFiable(C, A, R, k) then allouer(req(k)); enregistrer le nouvel état;
else bloquer Pk; mettre req(k) dans une file d'attente; rester dans l'ancien état end if;

CAS D'UNE LIBÉRATION DE RESSOURCES : lib(k) > 0
• état résultant A(t) = (A1, ... , Ak - lib(k), ... , An) et R(t) = R(t') + lib(k)
• essayer de servir une requête en attente. Une requête ne peut être servie que si EtatFiable pour le nouvel état
Politiques de service de la file des requêtes en attente : FIFO, priorité, mais contrainte (servir au moins une des requêtes conduisant à un état fiable sinon interblocage possible dû au blocage mutuel dans la file des requêtes)



ALGORITHME DU BANQUIER (Dijkstra, Habermann)

```
-- On suppose que l'état précédent (avant allocation à Pk) est fiable. On teste l'état résultant du service à Pk
type IdProc is range 1..n; type IdRessource is range 1..m;
type Vecteur is array(IdRessource) of Positive; type Matrice is array(IdProc) of Vecteur ;
type FonctionCaracteristique is array (IdProc) of boolean; -- tableau de booléens repérant l'appartenance ou non
EnsembleVide : FonctionCaracteristique := (others => False);
```

```
function EtatFiable (C, A : in Matrice; R : in Vecteur; k : in IdProc) return boolean is
   Total : Vecteur := R ; Dist : Matrice := C - A;       candidat : IdProc := IdProc'First;
   S : FonctionCaracteristique := (others =>False); T : FonctionCaracteristique := (others => True);
   -- S indique l'appartenance ou non à la suite fiable à construire, S(i) = True si Pi appartient à S
   -- T indique les processus encore utilisables pour construire la suite fiable repérée par S
begin
    While (not S(k)) and (T /= EnsembleVide) loop
        if T(k) then candidat := k; else while not T(i) loop candidat := candidat + 1; end if;
        -- si k ∈ T, on choisit k sinon on choisit un candidat tel que candidat  ∈ T;
        T(candidat ) := False; -- on sort de T le processus candidat choisi
        if Dist(candidat ) <= Total then -- la relation D2 est vérifiée pour le candidat
            S(candidat ) := True; --on ajoute le candidat  à la suite fiable
            Total := Total + A(candidat); -- on prépare pour le suivant de la suite fiable S
            T := not S; -- T comprend tous les processus non placés dans S
        end if; -- sinon on passe au processus suivant dans T
    end loop;
    return S(k); -- si Pk est dans la suite fiable, l'état est fiable
end EtatFiable ;
```

COMPLEXITÉ : exécution en $O(n(n + 1)/2)$ soit $O(n^2)$

ALGORITHME DU BANQUIER POUR M CLASSES DE RESSOURCES

(heuristique de Holt)

HEURISTIQUE : pour améliorer la recherche du prochain processus à mettre dans la sous-suite fiable

- **pour chaque classe i de ressource, trier les processus selon Dist(i) croissants**

on a m rangements des processus

- **marquer, dans chaque classe, les processus j tels que Dist(i)(j)<Total(i)**
- **mettre dans S un processus Pe qui est marqué dans toutes les classes.**
- **recalculer Total(i) et recommencer**

- **d'où complexité en O(m*n*logn)**

CAS D'UNE SEULE CLASSE DE RESSOURCES

AMÉLIORATION : classer les processus de T triés par Dist croissant
(c'est possible car une seule classe de ressource)

- **à chaque itération, deux tests, l'un sur k, l'autre sur candidat = premier(T)**

un seul test sur candidat car Dist(candidat) > Total => Dist(succ(candidat)) > Total

- **échec si Dist(premier(T)) > Dist(k) car aucun des succ(i) ne peut faire mieux que i**

Complexité en O(n) si on ne compte pas le tri, en O(nlogn) avec le tri

CAS D'UNE SEULE CLASSE DE RESSOURCES

```
type IdProc is range 1..n; type Matrice is array(IdProc) of Natural ;
type Permutation is array(IdProc) of IdProc;

C, A : Matrice; Dist : Matrice := C - A; -- contiennent les valeurs du nouvel état à tester
R : Natural ; -- résidu disponible après allocation à Pk
Rang : Permutation; -- suite de processus à valeurs Dist croissantes: Dist(rang(1)) ≤ .... ≤ Dist(rang(n))
   -- l'allocateur passe A, Dist, Rang et R qui correspondent à l'état résultant après allocation à Pk
   -- on essaie de servir Pk
   -- et si on ne peut pas, on essaie de servir le processus de plus petit Dist, ce qui permet d’augmenter Total
```

```
function EtatFiable(A, Dist: Matrice; Rang: Permutation; R: Natural; k: IdProc) return boolean is
    Total : Natural := R ;
    candidat, j : IdProc := IdProc'First; -- pour créer la suite fiable S
begin                                    -- hypothèse : l’état avant allocation à Pk est un état fiable
    j := 1 ; candidat:= Rang(j) ; --on examine le processus de plus petite Dist
    while ((Dist(k) > Total) and (Dist(candidat) <= Total)) loop
        Total := Total + A(candidat); -- on prépare pour le processus suivant de S
        j := j + 1 ;   -- le rang suivant dans l'ordre des Dist croissants
        candidat := Rang(j) ; -- le processus correspondant
    end loop;
    return (Dist(k) <= Total); -- si Pk est dans la suite fiable, l'état est fiable
      -- Si elle existe, la suite fiable comprend :
            -- une sous suite : Rang(1 .. j-1),
            -- Pk,
            -- les autres processus qu'on se dispense d'examiner (grâce à l’hypothèse)
end EtatFiable;
```

EXEMPLE POUR UNE CLASSE DE RESSOURCES

Fonction EtatFiable

X = 9

R = 2 ÉTAT 1 : NON FIABLE

Rang	P5	P4	P3	P2	P1
Dist	2	3	4	6	6
A Alloué	1	1	1	2	2
Total : 2	3	4	5	@	
C Claim	3	4	5	8	8

R = 3 ÉTAT 2 : FIABLE

Rang	P5	P4	P3	P2	P1
Dist	2	3	4	6	7
A Alloué	1	1	1	2	1
Total : 3	4	5	6	8	9
C Claim	3	4	5	8	8

R = 1 ÉTAT 3 : P3 demande 2 ress NON FIABLE

Rang	P5	P4	P3	P2	P1
Dist	2	3	2	6	7
A Alloué	1	1	*3*	2	1
Total : 1	@				
C Claim	3	4	5	8	8

R = 2 ÉTAT 4 : P2 demande 1 ressource FIABLE

Rang	P5	P4	P3	P2	P1
Dist	2	3	4	5	7
A Alloué	1	1	1	*3*	1
Total : 2	3	4	5	8	
C Claim	3	4	5	8	8

R = 1 ÉTAT 5 : P4 demande 1 ress NON FIABLE

Rang	P5	P4	P3	P2	P1
Dist	2	2	4	5	7
A Alloué	1	*2*	1	3	1
Total : 1	@				
C Claim	3	4	5	8	8

Toute demande autre que celle de P5 sera refusée

R = 1 ÉTAT 6 : P5 demande 1 ressource FIABLE

Rang	P5	P4	P3	P2	P1
Dist	1	3	4	5	7
A Alloué	*2*	1	1	3	1
Total : 1	3	4	5	8	
C Claim	3	4	5	8	8

autorisations : 2 -> 4 -> 6

refus : 2 ≠> 3 4 ≠> 5

CAS D'UNE SEULE CLASSE DE RESSOURCES ET ANNONCES ÉGALES

CAS PARTICULIER : toutes les annonces sont égales : Ci = C0

Il suffit de garder assez de ressources pour que dans le nouvel état (après allocation de req(k) au processus demandeur), un processus J au moins puisse encore atteindre son annonce max, c'est à dire Dist(J) = 0 :

Condition : le nouvel état est fiable s'il y existe au moins un processus J tel que Dist(J) ≤ R

Le test du banquier devient, pour un processus k qui demande req(k) ressources en plus et dont la distance actuelle est Dist(k)

```
function EtatFiable(k : in IdProc) return Boolean is
        OK : Boolean;  -- on peut encore servir toute l'annonce d'au moins un processus
begin
        if req(k) > R then
                return False ; -- on ne peut même pas passer dans le nouvel état
        else
                R := R - req(k) ;  -- on attribue les req(k) ressources à k, pour simuler le nouvel état
                Dist(k) := Dist(k) - req(k) ;

                for J in 1..IdProc loop
                        OK := (Dist(J) <= R); -- un processus au moins pourra atteindre son annonce
                        exit when OK; -- on en a trouvé 1, c'est bon et cela suffit
                end loop;

                R := R + req(k) ;  -- on retourne à l'état initial
                Dist(k) := Dist(k) + req(k) ;
                return OK;
        end if;
end EtatFiable;
```

CAS D'UNE SEULE CLASSE DE RESSOURCES

PRÉCAUTION PRÉALABLE

On se donne assez de ressources pour qu'il n'y ait jamais interblocage.

Avec

$$\mathbf{X > \Sigma(Ci - 1)}$$

alors, même au pire cas, celui où tous les processus sont bloqués avec le maximum de ressources qu'ils peuvent ne pas rendre, $(X - \Sigma(Ci - 1))$ processus ne sont jamais bloqués

Le test du banquier est inutile (aucun test n'est à faire). C'est une prévention statique.