SYS-S5 Interblocage 9-10/12

Florian Cliquet
Note Taking of Online Resources

9 juin 2024

Plagiarism Mention

We attest that the content of this document is original and stems from our personal reflections.

Sommaire

Introduction

This document provides an overview of SYS-S5 concepts written by Cliquet Florian. It's a set of notes on multiple online resources.

1 Interblocage

1.1 Introduction

1.1.1 Rappel

Une **ressource** désigne toute entité dont a besoin un processus pour s'exécuter. Elle peut être matérielle (processeur,périphérique,etc...) ou logicielle (variable,etc...). Les processus doivent souvent communiquer et partager différentes ressources entre eux. Généralement, les SE préfèrent accorder un accès **exclusif** à une ressource par un processus.
L'interblocage se produit quand deux processus s'attendent mutuellement.

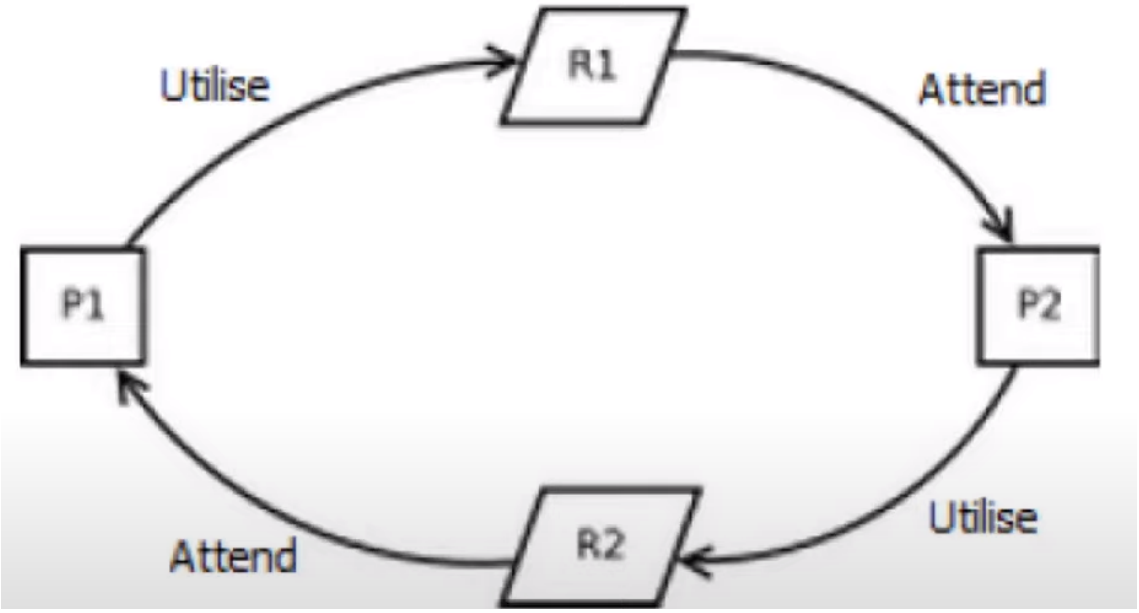

FIGURE 1 – Rappel - Interblocage

Le processus P1 utilise la ressource R1 qui est attendue par le processus P2 qui utilisent la ressource R2 attendue par P1.
Les interblocages peuvent se produire dans une multitude de situations autres que celle décrite précédemment :

— Entre ordinateurs : des ordis connectés au même réseau qui essaient d'utiliser des graveurs, Scanners, etc...

— Dans un SGDB : un programme verrouille les enregistrements qu'il utilise, demandés par un autre programme, et cherche à utiliser d'autres enregistrements verrouillés par le deuxième programme

1.1.2 Les ressources

Il existe deux types de ressource :

— **Retirable :** Une ressource qui peut être retirée sans dommage du processus auquel elle appartient. Elle ne cause pas d'interblocage, la retirer et la donner au processus en cours d'exécution

— **Non Retirable :** Une ressource qui ne peut être enlevée au processus auquel elle appartient sans que le traitement échoue. C'est la ressource concernée par l'interblocage.

La séquence nécessaire pour utiliser une ressource :

— Sollicitation de la ressource

— Utilisation de la ressource

— Libération de la ressource

Selon le SE, Si la ressource n'est pas disponible, le processus qui la sollicite peut-être :

— automatiquement bloqué jusqu'à la libération de la ressource

— mais en sommeil pour un certain temps avant de le réveiller pour qu'il puisse la redemander

`Comment assurer un accès exclusif à deux ressources par le processus A ?`

```
typedef int semaphore;
semaphore resource_1;
semaphore resource_2;
void process_A(void) {
      down(&resource_1);
      down(&resource_2);
      use_both_resources( );
      up(&resource_2);
      up(&resource_1);
}
```

FIGURE 2 – accès exclusif

Et si un processus A demande la ressource_1 et la ressource_2 et que le processus B demande la ressource_2 et la ressource_1 ?

```
semaphore resource_1;
semaphore resource_2;
void process_A(void) {
     down(&resource_1);
     down(&resource_2);
     use_both_resources( );
     up(&resource_2);
     up(&resource_1);
}
```

```
void proces_B(void) {
     down(&resource_2);
     down(&resource_1);
     use_both_resources( );
     up(&resource_1);
     up(&resource_2);
}
```

FIGURE 3 – exemple

Trier la demande de ressources (toujours dans l'ordre 1,2,...) évite l'interblocage.

1.2 Interblocage

Un ensemble de processus est en interblocage si chaque processus attend un événement que seul un autre procesuss de l'ensemble peut provoquer.

1.2.1 Conditions nécessaires pour l'interblocage

De la définition précédente, on peut déduire les conditions nécessaires pour produire un interblocage :

- **Exclusion mutuelle :** chaque ressource est soit attribuée à un seul processus ou elle est libre
- **Détention et attente :** les processus ayant déjà obtenu des ressources peuvent en demander de nouvelles
- **Pas de réquisition :** les ressources déjà détenues doivent être explicitement libérées par le processus qui les détient. Elles ne peuvent pas être retirées de force.
- **Attente circulaire :** il doit y avoir un cycle d'au moins deux processus

1.2.2 Modélisation de l'interblocage

Un carré est une ressource.
Un cercle est un processus.
Un arc orienté d'une ressource à un processus (a) signifie que la ressource est détenue par le processus.
Un arc orienté d'un processus à une ressource (b) signifie que le processus est bloqué en attendant la ressource.

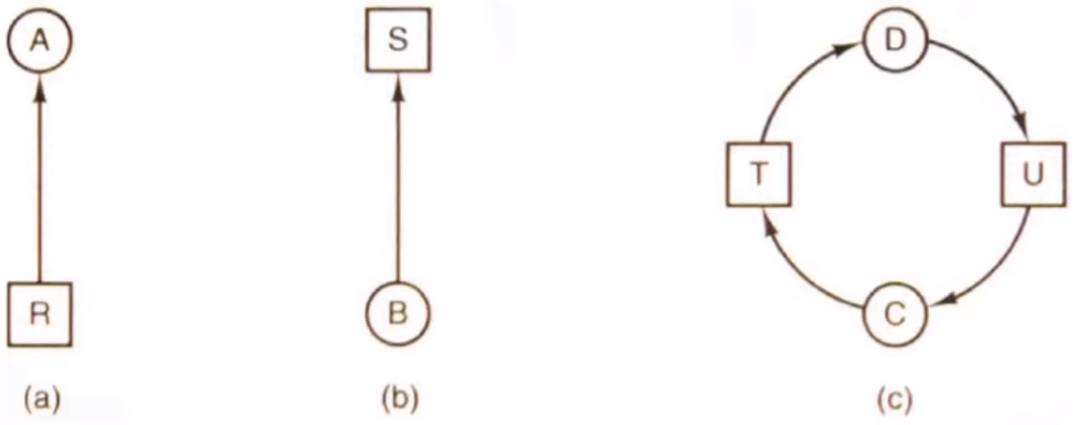

FIGURE 4 – Schéma de modélisation

Situations Exemples :

Situation 1

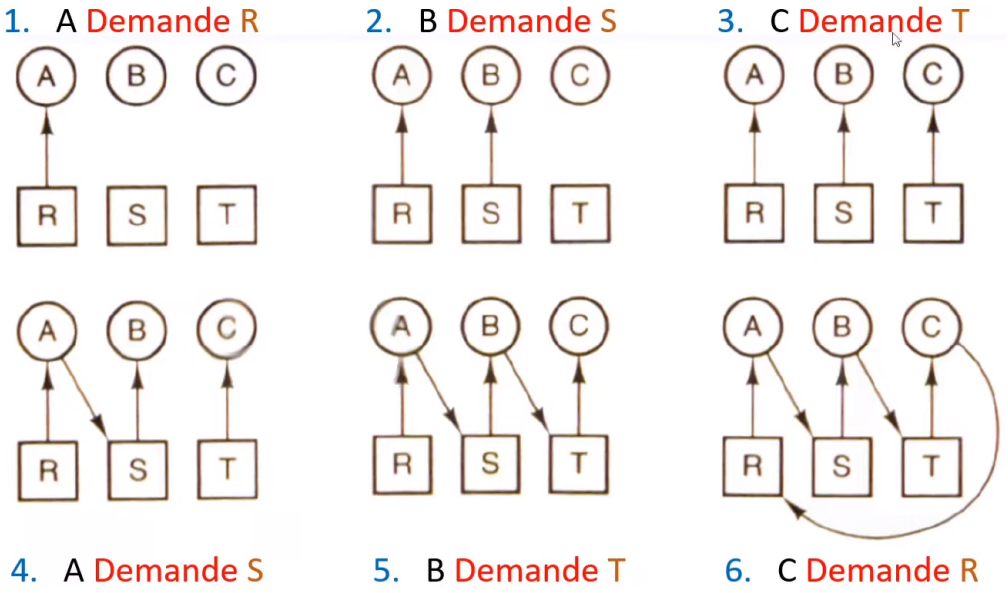

FIGURE 5 – Situation 1

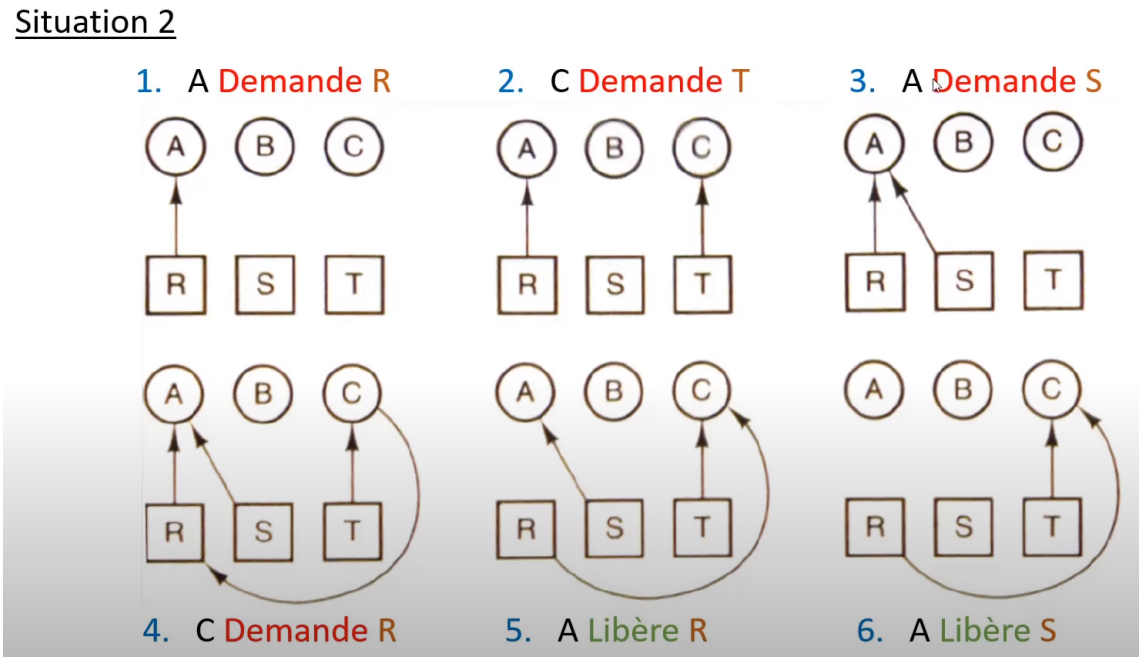

FIGURE 6 – Situation 2

1.2.3 Gestion de l'interblocage

1. **Ignorer les problèmes**
   La politique de l'autruche :
   - Permettre des interblocages
   - N'essayer même pas de les détecter

   Cette stratégie est inacceptable par les mathématiciens. Les chercheurs cherchent à connaître la fréquence du phénomène et le nombre de fois que le SE s'arrête pour d'autres raisons.
   - Si l'interblocage arrive 1 fois par mois alors que le SE s'arrête 1 fois par semaine pour d'autres raisons, il vaut mieux ne pas perdre en performance pour éliminer les interblocages.
2. **La détection et la reprise des interblocages**
   Le système ne cherche pas à empêcher les interblocages mais il les laisse se produire, puis tente de les détecter et d'y remédier **a posteriori**.
   (a) **Détection avec une ressource de chaque type**
   Le premier cas le plus simple est qu'il n'existe qu'une seule ressource pour chaque type : un DVD, un Scanner, une imprimante, etc.. Dans ce cas il suffit de construire un graphe de ressources :
   - Si le graphe contient un ou plusieurs cycles, il existe un interblocage
   - Tout processus faisant partie d'un cycle est bloqué

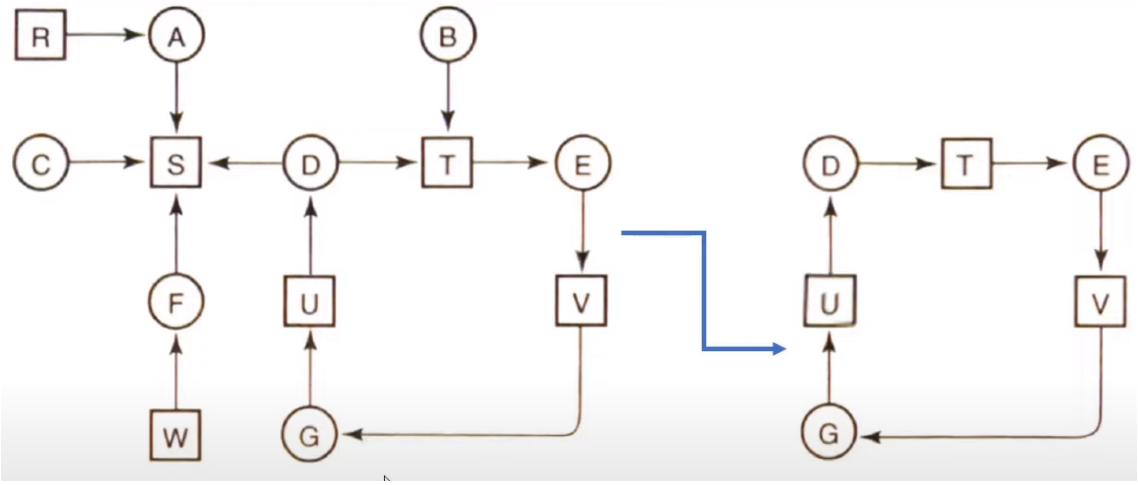

FIGURE 7 – Graphe de ressource

Facile à trouver visuellement si le nombre de noeuds est petit, cependant il faut un algorithme qui permet de détecter un cycle dans un graphe orienté pour les plus grands nombres de noeuds.
Parcours en profondeur (DFS - Depth First Search) :

- Initialiser `L` une pile vide et désignez tous les arcs comme non marqué
- Pour chaque noeuds `n` :
- Si le noeud `n` n'est pas dans la pile `L`, l'ajouter à `L`, sinon -> Il existe un cycle et l'algorithme prend fin
- S'il n'existe pas un arc non marqué sortant de `n`, déplier `n`.
  Si la pile est vide, revenz à l'étape 2, sinon le noeud au sommet de la pile devient `n` et revenez à l'étape 4.
  Sinon choissisez au hasard un arc sortant non marqué et marquez-le, pointer sur le noeud au bout de l'arc choisi qui devient `n` et revenez à l'étape 3

Si l'algorithme ne se termine pas à l'étape 3,il n'y a pas de cycle (interblocage).

(b) **Détection avec plusieurs ressources de chaque type**
Dans ce cas, il existe plusieurs exemplaires de certaines ressources.

FIGURE 8 – Modélisation

L'algorithme proposé, pour `n` processus, utilise :

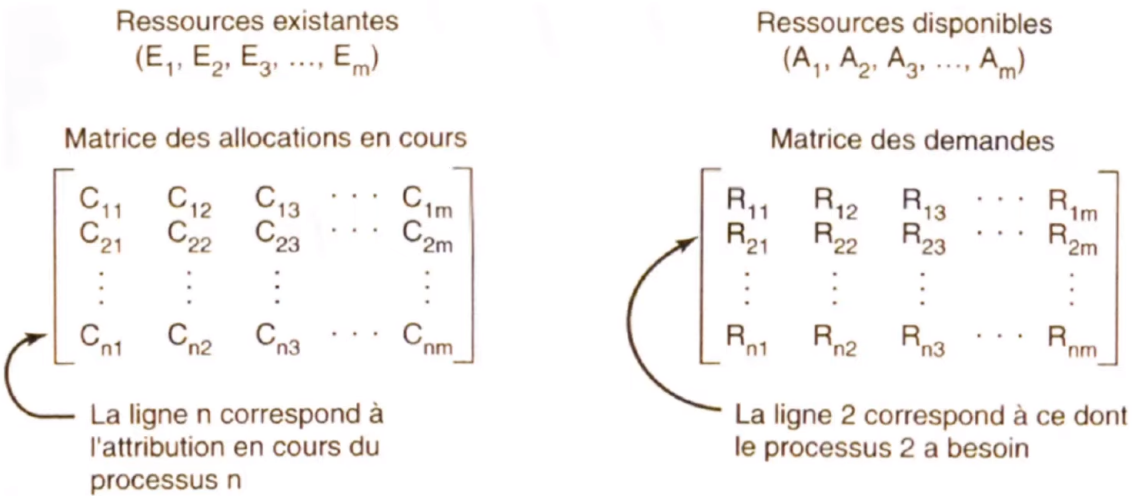

FIGURE 9 – Modélisation Algorithme

`Avec` C_{ij}`: le nombre de ressources j allouées au processus i` et
`Avec` R_{ij}`: le nombre de ressources j demandées par processus i`

— Chaque ressource est soit attribuée, soit disponible :
$\sum C_{ij} + Aij = Ej$
— Définissons la relation :
A $\leq$ si et seulement si $A_i \leq B_i$ pour 1 $\leq$ i $\leq$ m
— Au départ, chaque processus non marqué P_i avec $R_i \leq$ A

— Si on le trouve, on ajoute la $i^{\text{ème}}$ rangée de C à A (-> Nous avons trouvé un processus qu'on peut exécuter et on libère ses ressources après exécution)

— Si un tel processus n'existe pas, l'algorithme se termine

Lorsque l'algorithme se termine, tous les processus non marqués, s'il y'en a, sont en interblocage.

`Exemple :`

Pour l'exemple suivant, est-ce qu'il existe un ou plusieurs interblocages ?
E = (4 2 3 1) et A = (? ? ? ?)

C				R			
0	0	1	0	2	0	0	1
2	0	0	1	1	0	1	0
0	1	2	0	2	1	0	0

— (1) La colonne C_1 à besoin de 2 processus , C_2 1, C_3 3 et C_4 1 -> A = ($(E_1$-2,E_2-1,E_2-3,E_2-1)
Soit A = (2 1 0 0)

— On remarque que la ligne 3 de R est compatible avec A, on barre donc P_3 et supprimons la 3ème ligne de C et de R (remplace par des 0). A devient donc (2 2 2 0) , on lui a ajouté la ligne 3 de C.

— On fait de même avec la nouvelle matrice A, on voit qu'elle est compatible pour la 2ème ligne de R, soit A = (4 2 2 1) et on barre P_2

— De même avec la première ligne et on barre P_1

`En conclusion, on a pu barrer` P_1`,` P_2 `et` P_3`, cela veut dire qu'il n'y pas d'interblocage !`

(c) **Quand chercher l'interblocage**

On peut chercher l'interblocage :

— A chaque demande de ressource (-> modification du graphe). Efficace mais gourmand en terme de temps cpu

— Périodiquement : toutes les `k` minutes. La détection peut être tardive

— Juste lorsque l'utilisation du cpu devient inférieur à un certain seuil. La détection peut être tardive

(d) **Reprendre un interblocage**

Que faire après la détection d'un interblocage ?

— Retirer temporairement une ressource à un processus pour l'attribuer à un autre (mais pas de réquisition donc pas applicable)

— Restaurer un état antérieur (retour arrière) et éviter de retomber dans la même situation

— Supprimer un ou plusieurs processus

-> Reprendre au moyen d'un rollback : Faire passer régulièrement les processus par des points de reprise.

— Inscrire l'état du processus (image de la mémoire et l'état des ressources actuellement attribuées) dans un fichier pour pouvoir le restaurer ultérieurement
— A chaque fois créer de nouveaux rollback et ne pas écraser les anciens
— En cas d'interblocage, avant de retirer une ressource à un processus, le resituer au rollback avant l'acquisition de cette ressource

-> Supprimer un ou plusieurs processus :

— La manière la plus simple consiste à supprimer un ou plusieurs processus du cycle
— Il est préférable de supprimer un processus qui peut-être redémarré depuis le début sans conséquences néfastes
— Par exemple : retirer un compilateur plutôt qu'un programme qui met à jour une DB (incrémenter un attribut). (stateful vs stateless)

3. **L'évitement des interblocages**
Le système doit déterminer si l'attribution de la ressource est **sûre** et ne l'accorde que dans ce cas. Dans ce cas la, le système peut terminer tous les processus après cette attribution.
En 1965, Dijkstra a proposé **l'algorithme du banquier** qui permet d'éviter les interblocages. Il s'est inspiré de la manière dont un banquier accorde des crédits à un groupe de clients.

— Soit 4 clients A,B,C et D qui se font accorder un crédit de 22 unités (Max de la figure)

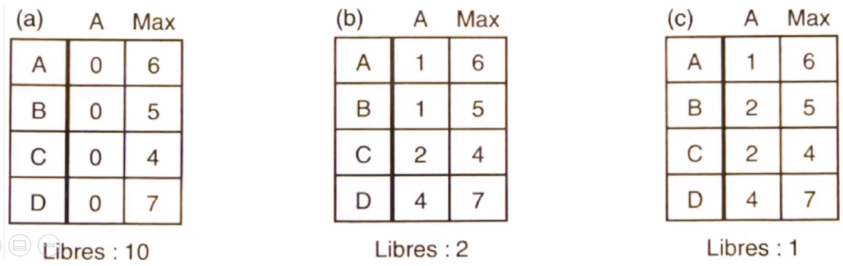

FIGURE 10 – Matrice banquier

— Le banquier sait qu'ils ne demanderont pas leur Max en même temps alors il n'accorde que 10 unités au groupe (ressource disponibles)
— Les clients vont commencer à demander leur crédit et on peut être dans la situation (b)
— La situation (b) est sûre car on peut donner 2 unités à C qui se terminera et libérera ses unités aux autres clients
— Par contre, si B demande une autre unité (la situation (c)), le système sera dans une situation dite **pas sûre** : si on attribue l'unité à B, il nous

reste une unité qui est insuffisante à terminer le crédit d'aucun de nos clients

4. **La prévention des interblocages** L'évitement de l'interblocage est pratiquement impossible car il nécessite des informations relatives aux requêtes futures. Pour prévenir l'interblocage, on peut faire en sorte qu'une des quatre conditions de l'interblocage ne soit jamais satisfaite :
   — Pas d'exclusion mutuelle : Tout traité en différé -> impossible car certaines ressources sont à usage exclusif
   — Pas de "détention et attente" : Demander toutes les ressources dès le départ -> Le processus ne doit pas détenir des ressources et en demander d'autres. Il est difficle de prévoir les besoins du processus
   — préemption (Retire des ressources) : On peut l'envisager pour certaines ressources dont le contexte peut être sauvegardé et restauré
   — Pas d'attente circulaire : établir un ordre total entre les ressources et imposer, à chaque processus, la règle de demande de ressources suivantes : `Un processus peut demander une ressource` R_j `seulement si toutes les ressources qu'il détient sont inférieures à` R_j

1.2.4 Autres considérations

En générale, l'évitement et la prévention des interblocages ne sont pas facile à mettre en oeuvre. Par contre il existe d'excellents algorithmes spécialisés destinés à des applications spécifiques.

1. **Le verrouillage en deux phases** Dans les SGDB (Système de Gestion de Base de Données), un processus peut utiliser certains enregistrements et laisser les autres processus utiliser les autres enregistrements. L'algorithme consiste à ce que le processus fasse ses modifications en deux phases :
   — Le processus tente de verrouiller un à un tous les enregistrements dont il a besoin
   — S'il réussit, il démarre la seconde phase qui consiste à faire les mises à jour et à la libération des verrous
   — Si, au cours de la première phase il trouve un enregistrement qui est déjà verrouillé, il libère tous ses verrous et reprend cette phase ultérieurement
2. **Les interblocages de communication**
   Exemple :
   `A envoie un message à B. Le message se perd. B attend le message et A attend la réponse.`
   Phénomène de **interblocage sans possession de ressources.**
   Solution -> **Temporisation :** après un certain délai, A déduit que le message a été perdu et le réemet autant de fois que nécessaire.