Systèmes d'exploitation
Gestion des processus

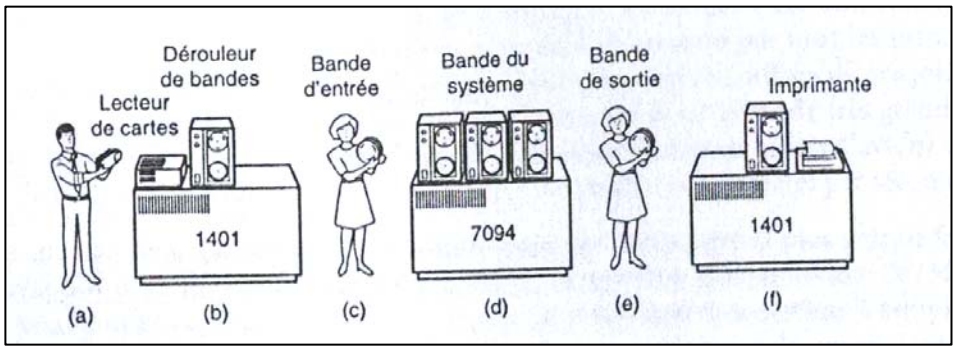
Mohamed Fathi KAROUI

LAGIS : Laboratoire d'Automatique, Génie Informatique & Signal

Karoui.mohamed@ec-lille.fr

historique

- 1er ordinateur numérique créé par Charles Babbage 1850
- 1ère génération (1945 – 1955) : tube à vide et tableaux d'interrupteur. L'ENIAC (université de Pennsylvanie).
- la 2ème génération (1955 – 1965)
  - Apparition du transistor 1955
  - Mainframes (ordinateur centraux) : programmes en FORTRAN



Système de batch

historique

- la 3ème génération (1965 – 1980)
  - IBM présente son system/360 (1er machine dotée de circuits intégrés)
  - Apparition de la multiprogrammation
  - MIT, Bell Labs & General Electric conçoivent le système *MULTICS* (*MULTIplexed information and Computing Service*)
  - 1er mini-ordinateur le DEC PDP-1 en 1961
  - En 1969 Ken Thomson (ancien programmeur de Bell Labs) écrit une version mono-utilisateur de MULTICS sur un PDP-7 qui lui a servi de base pour produire un système Multi-Utilisateurs et multitâches : UNIX

historique

- la 4ème génération (1980 – aujourd'hui) : les ordinateurs personnels
  - Apparition des circuits LSI (Large Scale Integration circuits)
  - Intel sort le 8080 (1er processeur 8 bits), Gary Kildall conçoit un système d'exploitation orienté disque : le CP/M (*Control Program for Microcomputers*) créant ainsi le 1er micro-ordinateur équipé d'un disque.
  - Kildall fonde *Digital Reserch* pour commercialiser le CP/M.
  - Début 1980, IBM crée l'IBM PC.
  - Gates fonde Microsoft et commercialise sont MS-DOS sur les IBM PC.
  - S. Jobs d'Apple sort le Machintosh : 1er ordinateur avec un SE à interface graphique
  - Microsoft lance Windows

Historique: quelques chiffres

généralisation	**1ère**	**2ème**	**3ème**	**4ème**
Nom de la machine	L'ENIAC	PDP-1	PDP-8/1	LSI-11
Composants	Tubes électronique	Transistor	Circuits intégrés	Microprocesseur
Année	1950	1960	1965	1976
Encombrement	Un bâtiment	Armoires	Un rack	Une carte
Dimensions	15*15*6 m	2.4*0.75*1.8 m	60*60*60 cm	22*25*1.2 cm
Consommation (watts)	150 000	2 500	250	50
Nombre d'accès mémoire par seconde	80 000	200 000	600 000	900 000
Prix ($)	?	120 000	10 000	650

Unix

- 1969
  - Développement d'un environnement de programmation sur DEC PDP/7 par Ken Thomson (BELL-AT&T)
- Années 70
  - Nouvelles versions sur PDP/11
  - Création du langage C par D. Ritchie et B. Kernigham
  - Indépendance d'Unix vis à vis des machines
- 1988
  - Deux grandes versions concurrentes :
    - BSD 4.3 Université de Berkeley (Californie)
    - System 5 AT&T
- 1991, 25 Août à 20h57min et 8s
  - Naissance de Linux

Linux : le début

```
                          Begin post from LinusFrom:
torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: What would you like to see most in minix?
Summary: small poll for my new operating system
Message-ID: <1991Aug25.205708.9541@klaava.Helsinki.FI>
Date: 25 Aug 91 20:57:08 GMT
Organization: University of Helsinki
Hello everybody out there using minix -
I'm doing a (free) operating system (just a hobby, won't be big and
professional like gnu) for 386(486) AT clones. This has been brewing since
april, and is starting to get ready. I'd like any feedback on things people
like/dislike in minix, as my OS ressembles it somewhat (same physical
layout of the file-system (due to practical reasons) among other things.

I've currently ported bash (1.08) and gcc (1.40), and things seem to work.
This implies that I'll get something practical within a fex months, and I'd
like to know what features most people would want. Any suggestions are
welcome, but l won't promise I'll implement them :-)

Linus (torvalds@kruuna.helsinki.fi)

PS. Yes -it's free of any minix code, and it has a multi-threaded fs. It is
NOT portable (uses 386 task switching etc), and it probably never will
support anything other than AT-hard disks, as that's all l have :-(.
```

Linux

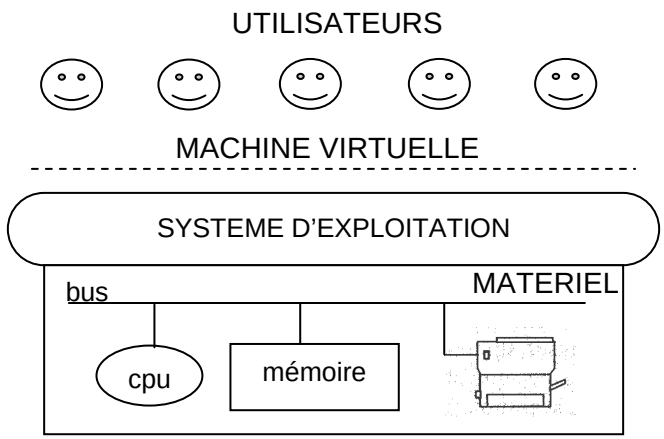
- Linus voulait mettre au point son propre système d'exploitation pour son projet de fin d'étude. Il avait pour intention de développer une version d'UNIX pouvant être utilisé sur une architecture de type 80386. Le premier clone d'UNIX fonctionnant sur PC a été Minix, écrit par Andrew Tanenbaum, un système d'exploitation minimal pouvant être utilisé sur PC. Linus Torvalds décida donc d'étendre les possibilités de Minix, en créant ce qui allait devenir Linux.
- L'originalité de ce système réside dans le fait que Linux n'a pas été développé dans un but commercial.
- Il existe plusieurs distributeurs : RedHat, Debian, SuSe, Mandrake, …

Linux : caractéristiques

- Système fiable, robuste et puissant. capable de fonctionner avec très peu de ressources.
- Le support des standards de l'Internet, c'est-à-dire des protocoles TCP/IP, la famille de protocoles utilisée sur Internet. Linux est donc un moyen gratuit de créer un réseau local, de se connecter à Internet et de mettre en place un serveur.
- Une sécurité accrue due à la transparence de son code source et de la réactivité de la communauté lors des annonces de vulnérabilités.
- Un cloisonnement des espaces mémoire et de l'espace disque couplé à une gestion pointue des droits permettant de gérer un grand nombre d'utilisateurs avec un niveau de risque minimal.
- Un noyau entièrement configurable en fonction du matériel de la machine sur laquelle le système est installé afin de maximiser les performances.

Systèmes d'exploitation

- Le système d'exploitation est un ensemble de programmes qui réalise l'interface entre le matériel de l'ordinateur et les utilisateurs. Il a deux objectifs principaux:
  - Construction au dessus du matériel d'une machine virtuelle plus facile d'emploi et plus conviviale;
  - Prise en charge de la gestion de plus en plus complexe des ressources et partage de celles-ci.



Fonction d'un système d'exploitation

- Gestion du processus
- Gestion de la mémoire
- Gestion des entrées-sorties
- Gestions des objets externes
- Gestion de la concurrence
- Gestion de la protection

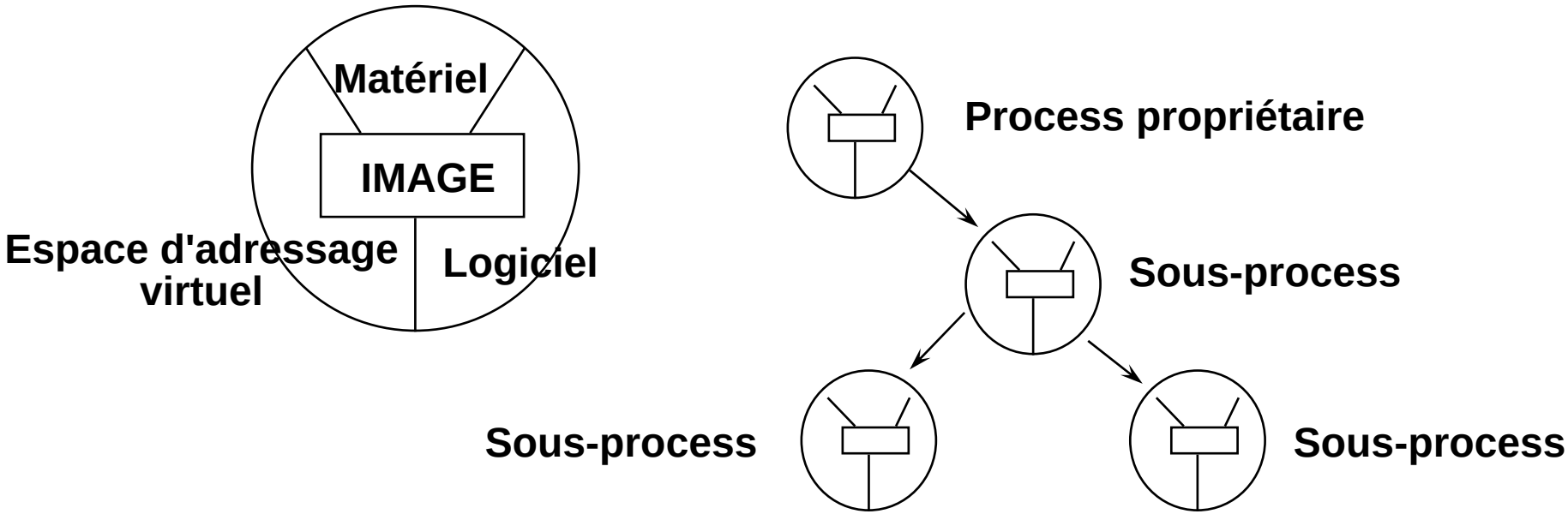
- Types de système d'exploitation
  - Les systèmes à traitement par lots
  - Les systèmes interactifs
  - Les systèmes temps réel

Plan

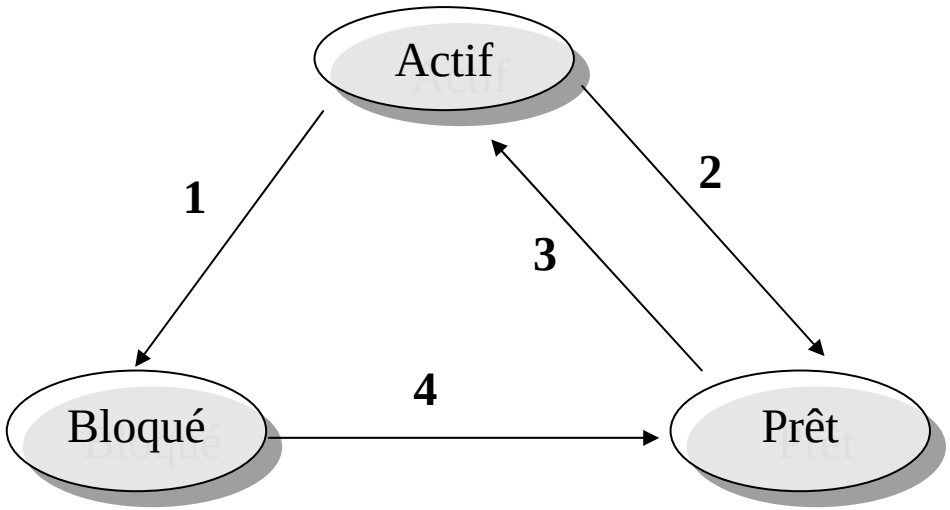
- Les processus
- L’ ordonnancement
- Synchronisation
- Exclusion mutuelle
- TAS (*Test And Set*)
- Sémaphores
- *Problème des philosophes*
- Producteur / consommateur

Processus

- Ensemble de contexte :
  - Matériel, Logiciel, Espace d'adressage virtuel

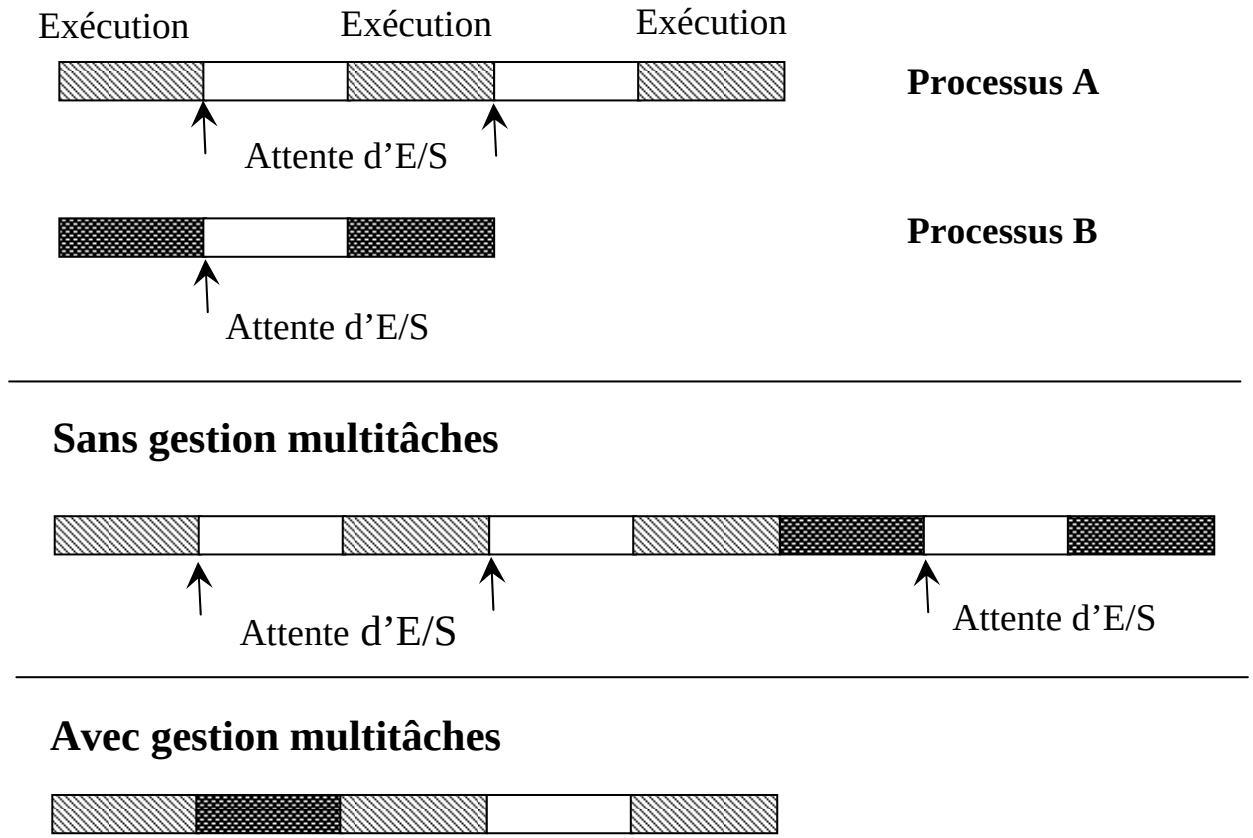


État d'un processus



1. Le processus se bloque en attente de donnés.
2. Désactivation du processus par L'Ordonnanceur.
3. Activation du processus.
4. Réveil du processus bloqué

Multiprogrammation



Critère d'évaluation de performances

- **Rendement**

  Nombre de travaux exécuter par unité de temps.

- **Temps de service**

  Temps qui s'écoule entre le moment où un travail est soumis et où il est exécuté (temps d'accès mémoire + temps d'attente dans la file des processus éligibles + temps d'exécution dans l'unité centrale + temps d'attente + temps d'exécution dans les périphériques d'entrée/sortie).

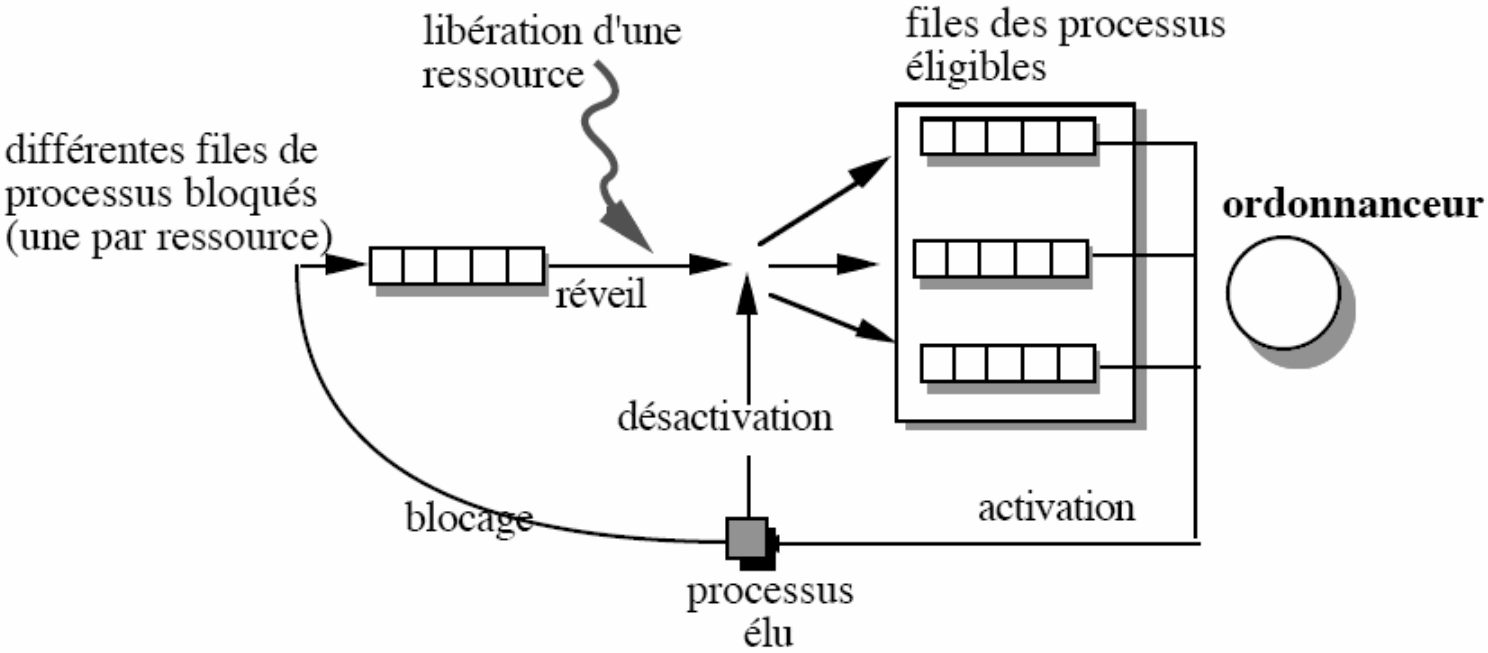
- **Temps d'attente**

  Temps passé dans la file des processus éligibles.

- **Temps de réponse**

  Temps qui s'écoule entre la soumission d'une requête et la première réponse obtenue.

Ordonnancement



But de l'ordonnancement :

- Equitabilité
- Efficacité
- Temps de réponse
- Temps d'exécution
- Rendement

Ordonnancement préemptif et non préemptif

- Si l'ordonnancement est non préemptif, la transition de l'état élu vers l'état prêt est interdite : un processus quitte le processeur s'il a terminé son exécution ou s'il se bloque ;

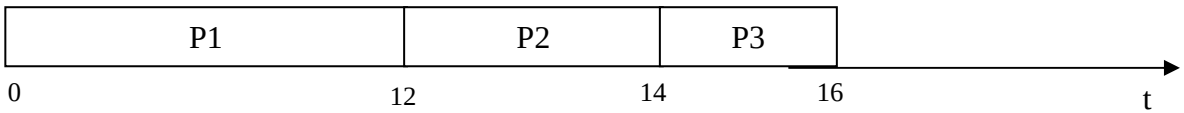
- Si l'ordonnancement est préemptif, la transition de l'état élu vers l'état prêt est autorisée : un processus quitte le processeur s'il a terminé son exécution, s'il se bloque ou si le processeur est réquisitionné.

Politiques d'ordonnancement

Politique Premier Arrivé, Premier Servi (First come First Served)

	durée	Ordre d'arrivée 1	Ordre d'arrivée 2
P1	12	1	3
P2	2	2	1
P3	2	3	2

Diagramme de Gantt selon le 1er ordre d'arrivée



Temps de réponse moyen = (12 + 14 + 16) / 3 = 14 ut
Temps d'attente moyen = (0 + 12 + 14) / 3 = 8,66 ut

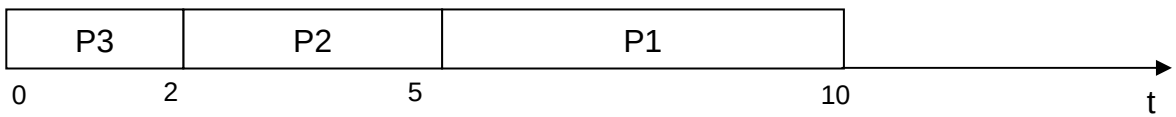
Pour le 2ème ordre d'arrivée :
Temps de réponse moyen = (2 + 4 + 16) / 3 = 7,6 ut
Temps d'attente moyen = (0 + 2 + 4) / 3 = 2 ut

Politiques d'ordonnancement

Politique plus court d'abord (Shortest Job first)

	Durée
P1	5
P2	3
P3	2

Diagramme de Gantt



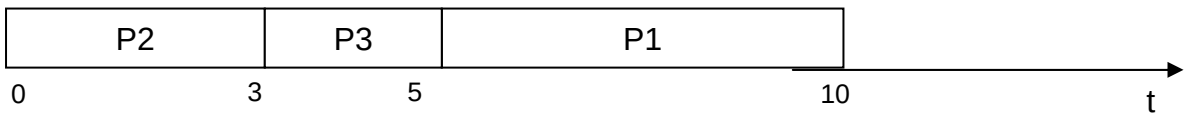
Temps de réponse moyen = (2 + 5 + 10) / 3 = 5,67 ut
Temps d'attente moyen = (0 + 2 + 5) / 3 = 2,33 ut

Politiques d'ordonnancement

Politique par priorité

	durée	priorité
P1	5	3
P2	3	2
P3	2	1

Diagramme de Gantt



Temps de réponse moyen = (3+ 5 + 10) / 3 = 6ut
Temps d'attente moyen = (0 + 3+ 5) / 3 = 2,67

Politiques d'ordonnancement

	Durée	Ordre d'arrivée
P1	8	1
P2	2	2
P3	4	3

Quantum = 2

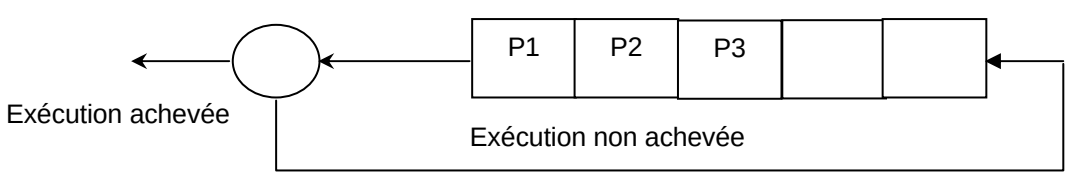
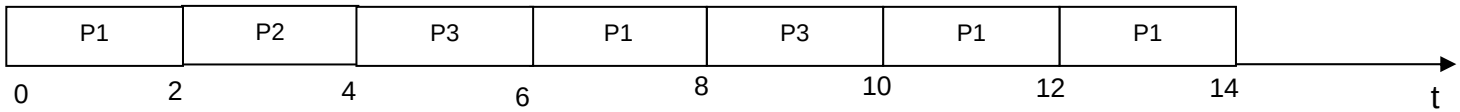
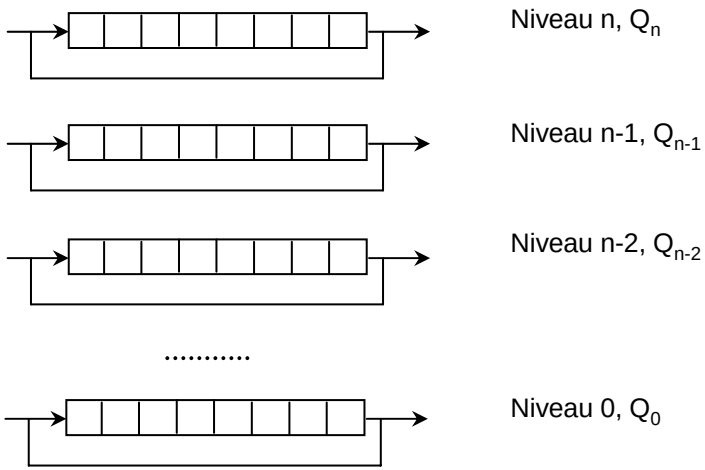


Diagramme de Gantt

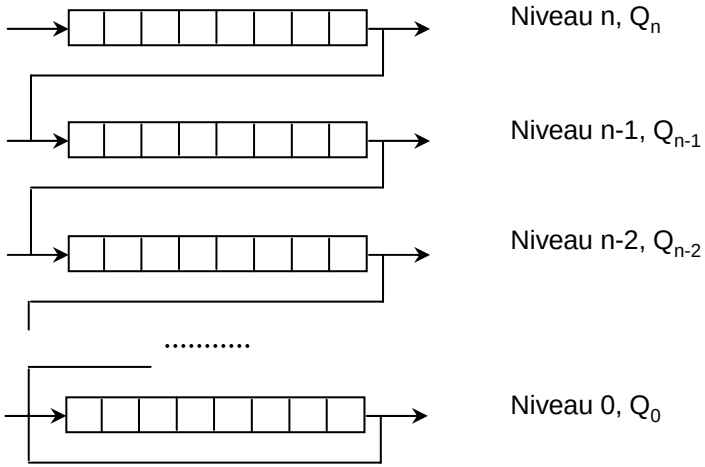


Temps de réponse moyen = (14 + 4 +10) / 3 = 9,33

Exemple d'ordonnancement

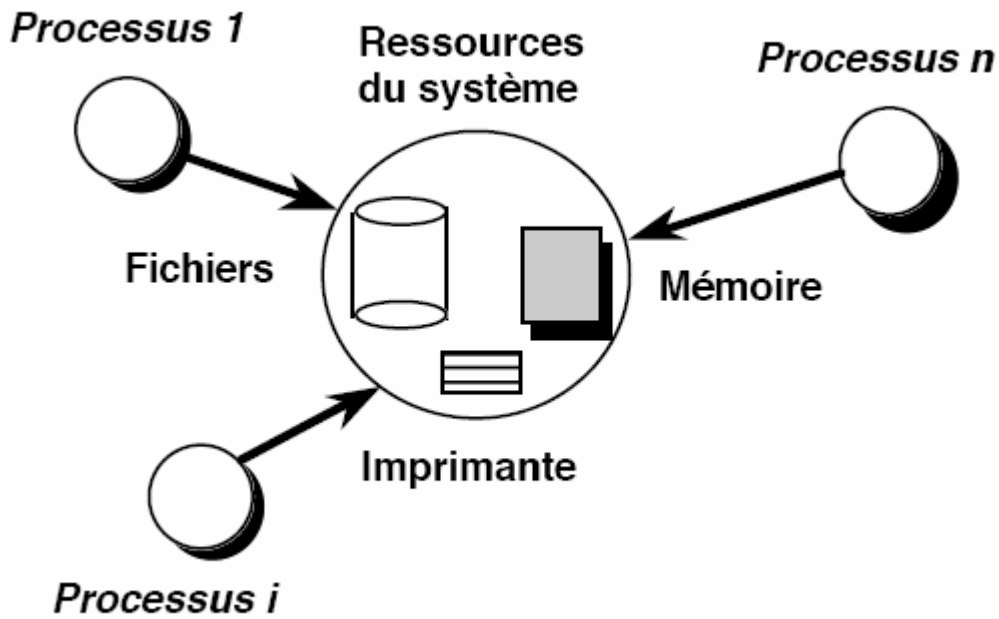


Files multiniveaux sans extinction de priorité



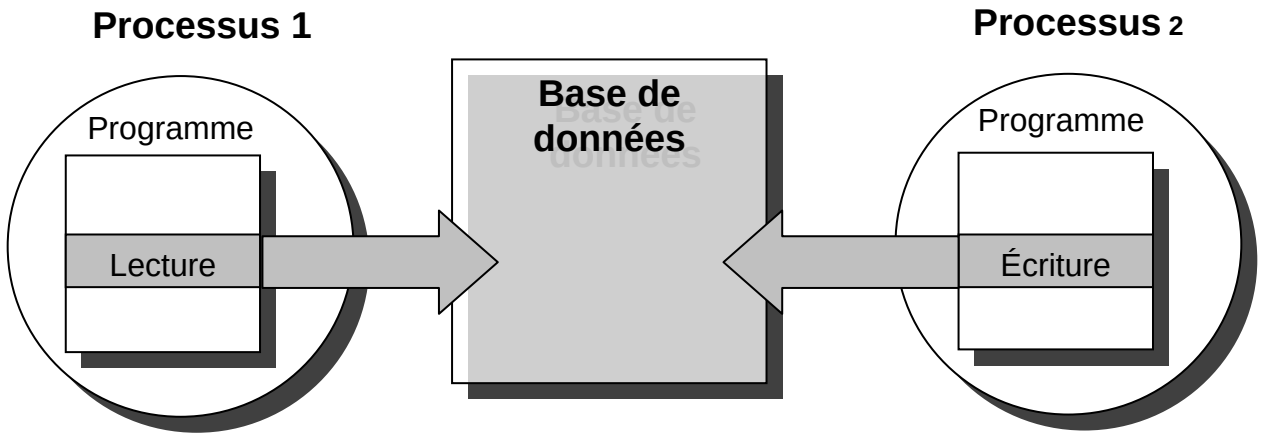
Files multiniveaux avec extinction de priorité

Synchronisation des processus



Notion de ressource critique qui correspond à une ressource ne pouvant être utilisée que par un seul processus à la fois

Exclusion mutuelle




```
Réservation :
Si nbplace > 0
Alors
        Réserver une place ;
        nbplace = nbplace – 1 ;
fsi
```

Exclusion mutuelle

- Pour réaliser l'exclusion mutuelle, il y a plusieurs propriétés à respecter :
  - *à tout instant un processus au plus peut se trouver en section critique*
  - *si plusieurs processus sont bloqués en attente de la ressource critique, l'un d'eux doit pouvoir y entrer au bout d'un temps fini*
  - *si un processus est bloqué hors d'une section critique, ce blocage ne doit pas empêcher l'entrée d'un autre processus en sa section critique*
  - *la solution doit être la même pour tous les processus*

Hypothèse : tout processus sort de section critique au bout d'un temps fini

Problème de DEKKER

- Les seules instructions indivisibles sont :
  - l'affectation d'une valeur à une variable
  - le test de la valeur d'une variable

- On dispose de deux processus P1, P2 et d'une section critique

DEKKER (1)

- On veut protéger la section critique avec un seul booléen c
  - c = faux <=> libre
  - c = vrai <=> occupé

```
Booléen c; c := faux
       Process Pi :
              Début
                     Ai :   si c alors aller à Ai
                     c := vrai
                            ...
                     c := faux
                            Reste du programme
                     Aller à Ai
              Fin;
```

DEKKER (1)

```
Réservation :
Masquer_it ; - prélude de section critique
Si nbplace > 0
Alors
        Réserver une place ;
        nbplace = nbplace – 1 ;
fsi
Démasquer_it ; - postule de section critique
```

DEKKER (2)

- On veut protéger la section critique avec une variable commune t
  - t = i ssi le processus Pi est autorisé à s'engager dans la section critique

```
Entier t; t := 1
       process Pi :
              Début
                     Ai :   si t = j alors aller à Ai
                            ...
                     t := j
                            Reste du programme
                     Aller à Ai
              Fin;
```

DEKKER (3)

- On introduit une variable booléenne par processus
  - c(i) vrai si Pi est dans sa section critique ou demande à y entrer
  - Pi peut seulement lire c(j)

```
Booléen tableau c(1:2); c(1) = c(2) := faux;
       Process Pi :
              Début
                     Ai :   si c(j) alors aller à Ai
                     c(i) := vrai
                            ...
                     c(i) := faux
                            Reste du programme
                     Aller à Ai
              Fin;
```

DEKKER (4)

- Nouvelle solution :

```
Ai :   c(i) := vrai
       si c(j) alors aller à Ai
```

- Exclusion mutuelle est garantie
  - mais pas la condition b) :
  - les deux processus peuvent s'engager dans une boucle infinie
- Nouvelle solution :

```
Ai :   c(i) := vrai
       Si c(j) alors
              Début
                     c(i) := faux
                     Aller à Ai
              Fin
```

DEKKER (5)

- Utilisation de trois variables communes

```
Entier t; booléen tableau c(1:2);
t := 1; c(1) = c(2) := faux;
Process Pi :
       Début
       Ai : c(i) := vrai
       Li : si c(j) alors
              Début
                si t = i alors aller à Li
                c(i) := faux
       Bi :     si t = j alors aller à Bi
                Aller à Ai
              Fin;
```

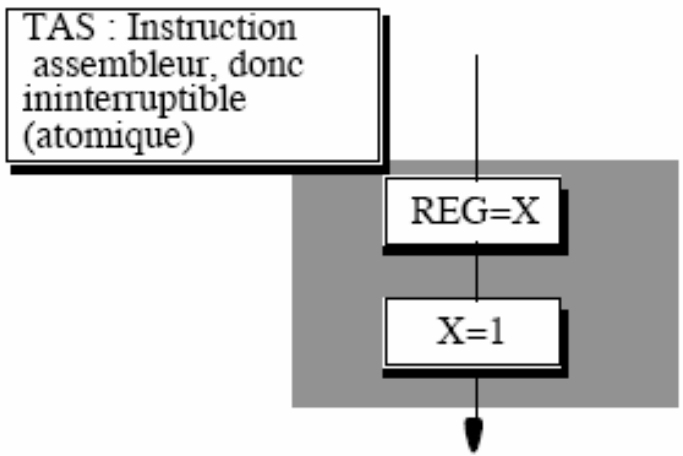
Entrée

Sortie

```
t := j; c(i) := faux
       Reste du programme
Aller à Ai
Fin;
```

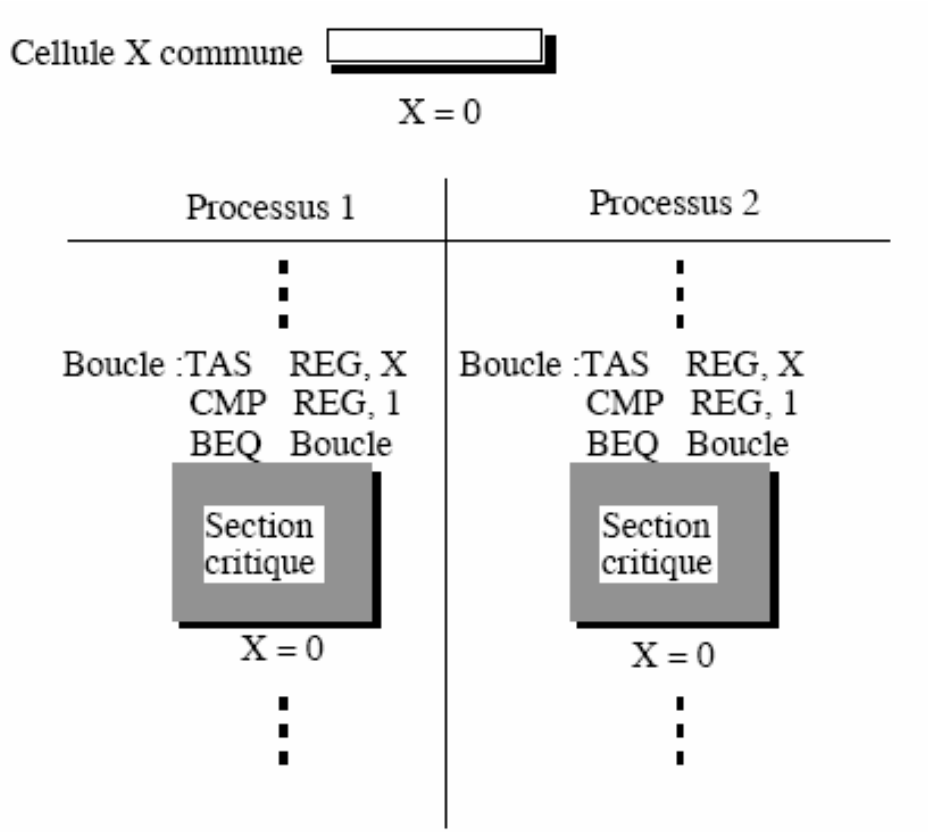
TAS (*Test And Set*)

- Conclusion : programmation très complexe
  - utilisation en dernier ressort d'un mécanisme câblé qui réalise une forme élémentaire d'exclusion
- Exemple : instruction élémentaire TAS (Test And Set)




```
Boucle :TAS   REG, X
        CMP   REG, 1
        BEQ   Boucle
        Section critique
        X = 0
```

TAS (*Test And Set*) : *Exemple*



Les sémaphores

- Définition:

  Un sémaphore *sem* est une structure système composée d'une file d'attente f(s) de processus et d'un compteur e(s), appelé niveau du sémaphore et contenant initialement une valeur e0(s). Cette structure ne peut être manipulée que par trois opérations P(sem), V(sem) et init (sem, e0(s)). Ces trois opérations sont des opérations indivisibles c'est-à-dire que l'exécution de ces opérations s'effectue interruptions masquées et ne peut être interrompue.

Les sémaphores

- *L'opération init*

  elle a pour but d'initialiser le sémaphore, c'est-à-dire qu'elle met à vide la file d'attente f(s) et initialise avec la valeur e0(s) le compteur e(s). on définit ainsi le nombre de jetons initiaux dans le sémaphore.

```
init (sémaphore sem, entier e0(s))
début
masquer_it;
       sem.e(s):=e0(s)
       sem.f(s):=0
démasquer_it;
fin
```

Les sémaphores

- *L'opération P(sem)*

  L'opération P(sem) attribue un jeton au processus appelant s'il en reste au moins un et sinon bloque le processus dans la file f(s). L'opération P est donc une opération éventuellement bloquante pour le processus élu qui l'effectue. Dans le cas du blocage, il y a réordonnancement et un nouveau processus prêt est élu. Concrètement, le compteur e(s) du sémaphore est décrémenté d'une unité. Si la valeur du compteur devient négative, le processus est bloqué.

```
P(s) :
début
Masquer_it
        e(s) := e(s) - 1
        si e(s) < 0 alors
                début
                        état(r) := bloqué
                        mettre r dans f(s)
                fin
démasquer_it
fin;
```

Les sémaphores

- *L'opération V(sem)*

  L'opération V(sem) a pour but de rendre un jeton au sémaphore. De plus, si il y a au moins un processus bloqué dans la file d'attente f(s) du sémaphore, un processus est réveillé. La gestion des réveils s'effectue généralement en mode FIFO. L'opération V est une opération qui n'est jamais bloquante pour le processus qui l'effectue.

```
V(s) :
début
masquer_it
        e(s) := e(s) + 1
        si e(s) <= 0 alors
                début
                        sortir v de f(s)
                        état(v) := actif
                fin
démasquer_it
fin;
```

Les sémaphores

- Problèmes d'interblocage:

  Considérons à présent la situation suivante pour laquelle deux processus P1 et P2 utilisent tous les deux deux ressources critiques R1 et R2 selon le code suivant.

```
Processus P1                          Processus P2
Début                                 Début
P(R2)                                 P(R1)
P(R1)                                 P(R2)
Utilisation de R1 et R2               Utilisation de R1 et R2
V(R2)                                 V(R1)
V(R1)                                 V(R2)
Fin                                   Fin
```

Synchronisation

- Hypothèse :
  - Les processus sont complètement asynchrones
- Objectifs :
  - Bloquer un processus ou soi même
  - Activer un processus avec ou sans mémorisation
- Réalisation : Deux techniques
  - Synchronisation directe : on nomme explicitement le processus
  - Synchronisation indirecte : on actionne un mécanisme qui agit sur d'autres processus

Synchronisation directe

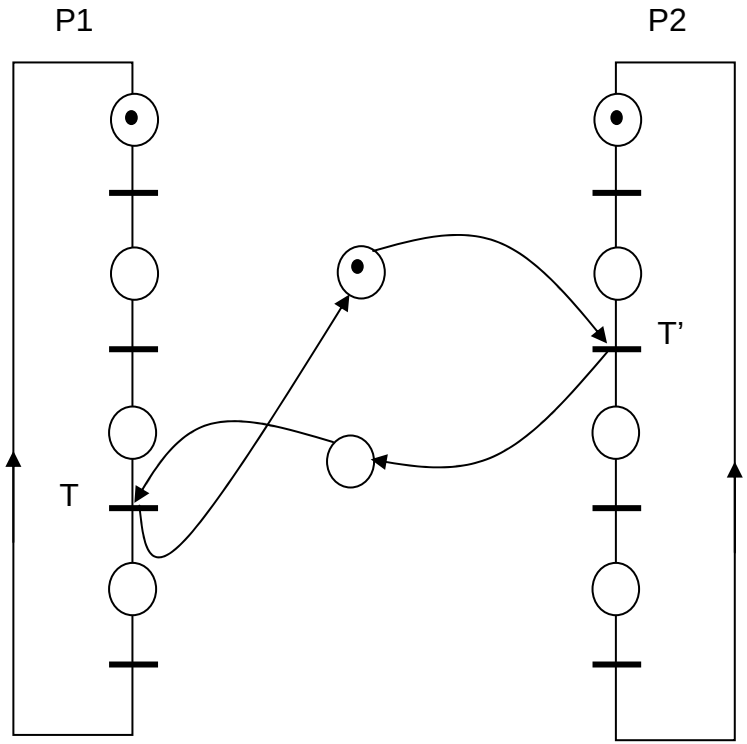
- On utilise deux fonctions : activer (q) et bloquer (q) avec un niveau de mémorisation.

```
Bloquer (q)
        Si témoin (q) alors
                Etat (q) ← bloqué
                Sinon témoin (q) ← faux
        Fsi
```

```
Activer (q)
        Si état (q)= actif alors
                Témoin (q) ← vrai
                Sinon état (q) ← actif
        Fsi
```

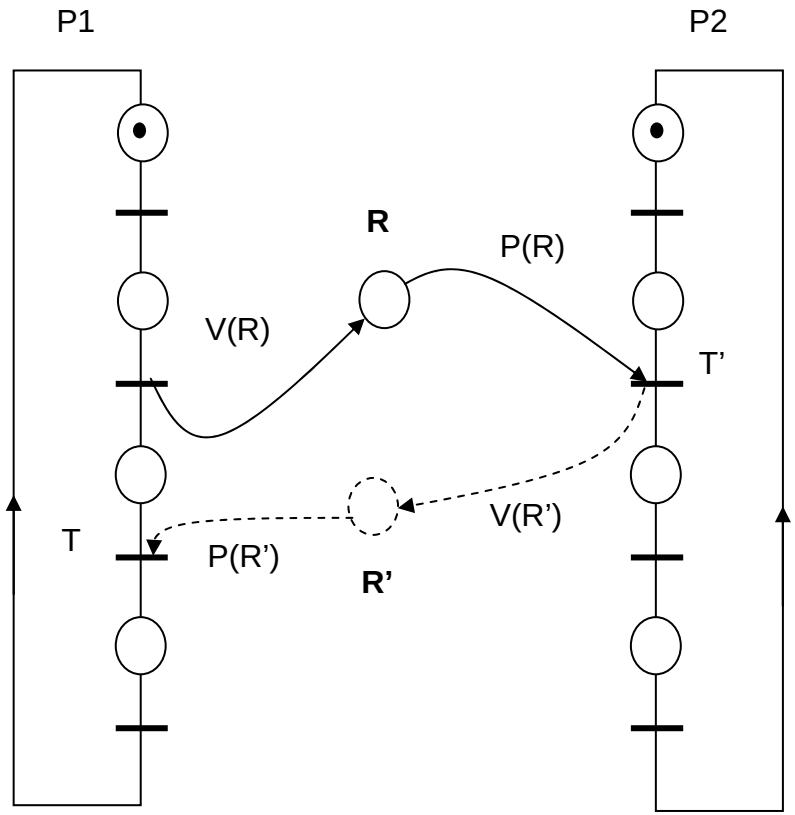
Synchronisation indirecte

- On actionne un mécanisme qui agit sur d'autres processus

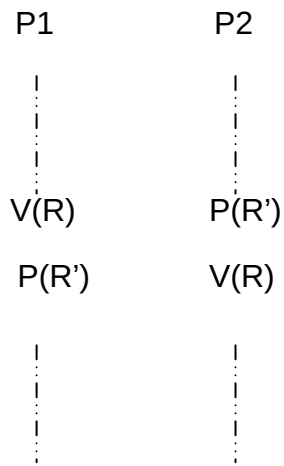


La transition T' ne sera pas franchise avant T.
Les processus P1 et P2 ne sont pas connus, seulement le mécanisme de synchronisation est connu.
Si P1 est plus rapide que P2, il n'y a plus de synchronisation.

Synchronisation indirecte



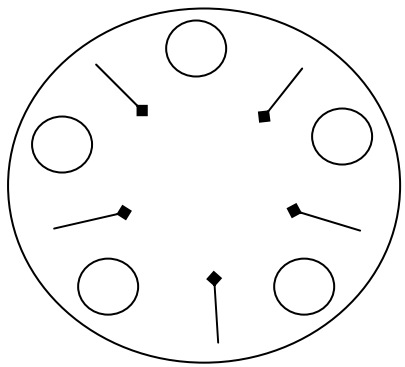
Quelque soit le processus le plus rapide, ils sont obligé de s'attendre l'un et l'autre.
Agencement de deux processus :



Valeur initiale de R et R' est zéro

Les philosophes

- Un philosophe passe son temps à manger et à penser. Lorsqu'un philosophe a faim, il tente de s'emparer des deux fourchettes qui sont à sa droite et à sa gauche, une après l'autre. L'ordre n'importe pas. S'il obtient les deux fourchettes, il mange pendant un certain temps, puis repose les fourchettes et se remet à penser. La question est la suivante : pouvez vous écrire un programme qui permette à chaque philosophe de se livrer à ses activités sans jamais être bloqué ?



Les philosophes

- 1ère solution

```
Philosophe (i)
Début
        Penser ;                          (le philosophe pense)
        Prendre_fourchette (i) ;          (prend fourchette gauche)
        Prendre_fourchette (i+1) ;        (prend fourchette droite)
        Manger ;                          (miam, miam, spaghettis)
        Poser_fourchette (i) ;            (poser fourchette gauche)
        Poser_fourchette (i+1) ;          (poser fourchette droite)
Fin
```

Les philosophes

- 2ème solution

```
Action pour demander de manger du philosophe i
                P(mutex)
                        si c(i-1) <> 2 et c(i+1) <> 2 alors
                                c(i) := 2; V(sempriv(i))
                        sinon c(i) := 1
                V(mutex)
                P(sempriv(i))
```

```
Action pour arrêter de manger
                P(mutex)
                        c(i) := 0
                        si c(i+1) = 1 et c(i+2) <> 2 alors
                                c(i+1) := 2; V(sempriv(i+1))
                        si c(i-1) = 1 et c(i-2) <> 2 alors
                                c(i-1) := 2; V(sempriv(i-1))
                V(mutex)
```

Les philosophes

- Solution générale (1/2) :

```
Entier tableau c(0:4); (v.i. = 0)
Sémaphore tableau sempriv(0:4); (v.i. = 0)
Sémaphore mutex; (v.i. = 1)

        Procédure test(k);
           si (c(k) = 1) et (c(k+1) <> 2) et (c(k-1) <>2)
               alors
                      c(k) := 2
                      V(sempriv(k))
           fin;
```

Les philosophes

- Solution générale (2/2):

```
Procédure philosophe(i)

        Li :   penser
                        P(mutex)
                        c(i) := 1
                        test(i)
                        V(mutex)
                        P(sempriv(i))
```

```
        manger
                        P(mutex)
                        c(i) := 0
                        test(i-1)
                        test(i+1)
                        V(mutex)
        aller à Li
```

Producteur / Consommateur

Prod : *{produire un message}*

```
nvide := nvide - 1
si nvide = -1 alors attendre
```

{déposer un message}

```
nplein := nplein + 1
si consommateur en attente
alors réveiller consommateur
```

aller à Prod

Cons :

```
nplein := nplein - 1
si nplein = -1 alors attendre
```

{prélever un message}

```
nvide := nvide + 1
si producteur en attente
alors réveiller consommateur
```

{consommer le message}

aller à cons