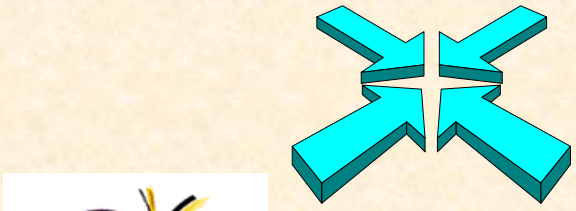
PROGRAMMATION CONCURRENTE



Your goal as programmer is not to write code, your goal is to solve problems.

Itamar Turner-Trauring

PROGRAMMATION CONCURRENTE : CONTEXTE

ARCHITECTURE MATERIELLE

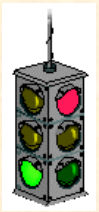
- Systèmes monoprocesseur
- Systèmes multiprocesseur
  - à mémoire commune (fortement couplés)

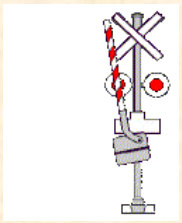
    multi cœur ou multi processeur
  - répartis (faiblement couplés)

    domaine du réseau et de la programmation répartie

PROGRAMMATION CONCURRENTE

PROGRAMMATION SÉQUENTIELLE OU PROGRAMMATION CONCURRENTE ?



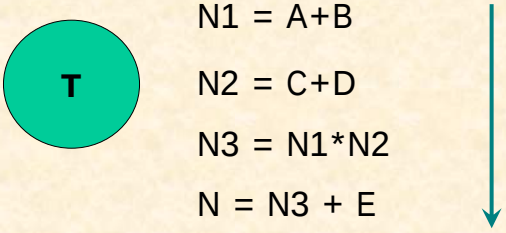


EXEMPLE 1

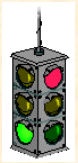
Soit le calcul:

N = (A+B) * (C+D) + E

➢ **Programmation séquentielle**



EXEMPLE 1

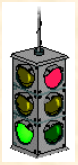


➢ **Programmation concurrente**

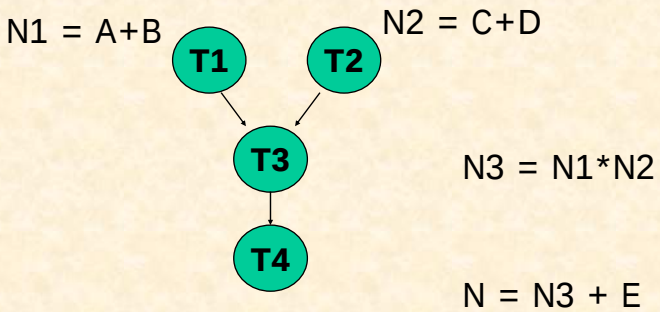
On peut décomposer le calcul en 4 tâches :

T1 : calcul de N1 = A+B
T2 : calcul de N2 = C+D
T3 : calcul de N3 = N1*N2
T4 : calcul de N = N3 + E

EXEMPLE 1



- **Programmation concurrente**



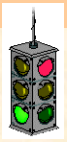
EXEMPLE 2

- Traitement de messages : programmation séquentielle

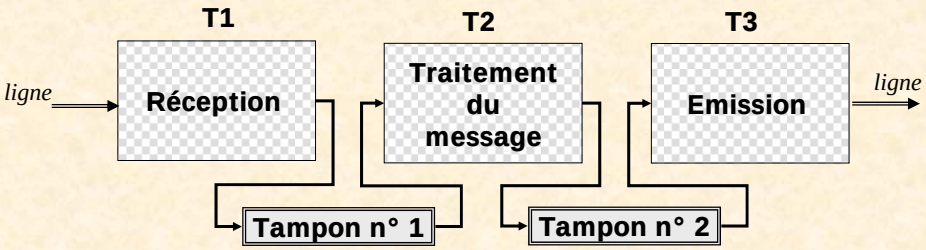
- temps total = t1 + t2 + t3
- réception impossible pendant le traitement et l'émission

PROGRAMMATION CONCURRENTE

EXEMPLE 2



- Traitement de messages : programmation concurrente



- temps total < t1 + t2 + t3
- réception pendant le traitement et l'émission

PROGRAMMATION CONCURRENTE : DEFINITIONS

UN PEU DE VOCABULAIRE ...

- Multitâche

  exécution de plusieurs tâches, concurrentes ou non, sur un seul processeur, en leur attribuant le processeur à tour de rôle

- Multiprocessing

  exécution de tâches concurrentes sur des processeurs séparés pouvant accéder à la même mémoire vive

PROGRAMMATION CONCURRENTE

INTERÊT

- Réalisation d'applications fonctionnellement multitâches
- Gain de temps par parallélisation des calculs
- Réactivité aux événements extérieurs
- Exploitation des architectures matérielles actuelles (processeurs multi cœurs)

PROGRAMMATION CONCURRENTE

INTERÊT

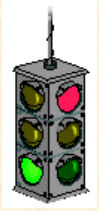
- Optimisation de l'utilisation du processeur dans les applications interactives ou effectuant beaucoup d'entrées-sorties (multitâche)
- Modularité des applications
- ...

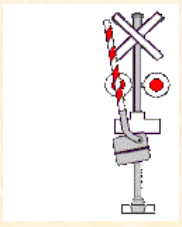
TYPES D'APPLICATIONS

- Calcul parallèle
  domaine scientifique, jeux vidéo, ...
- Systèmes réactifs temps réel
  conduite de procédés industriels, robotique, aéronautique, automobile, télécommunications, ..
- Systèmes multiutilisateurs
- Applications interactives, IHM
  meilleure réactivité

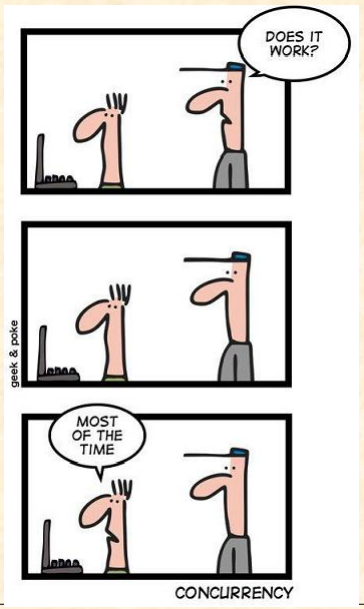
PROGRAMMATION SÉQUENTIELLE ET PROGRAMMATION CONCURRENTE :

QUELLES DIFFERENCES ?





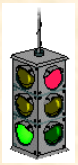
PROGRAMMATION CONCURRENTE



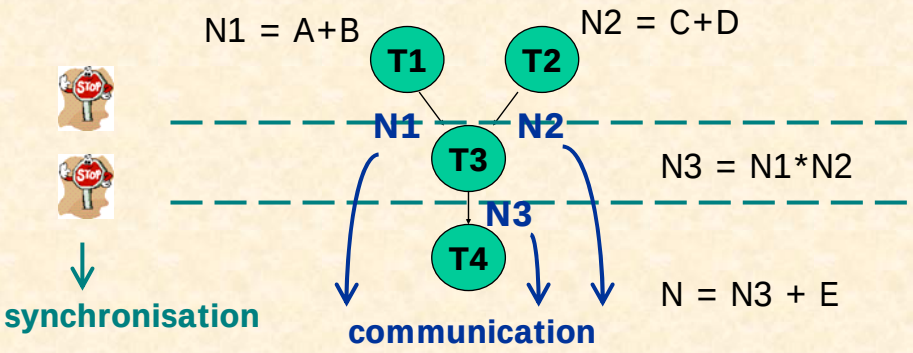
Source : Geek&Poke

PROGRAMMATION CONCURRENTE

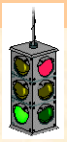
EXEMPLE 1



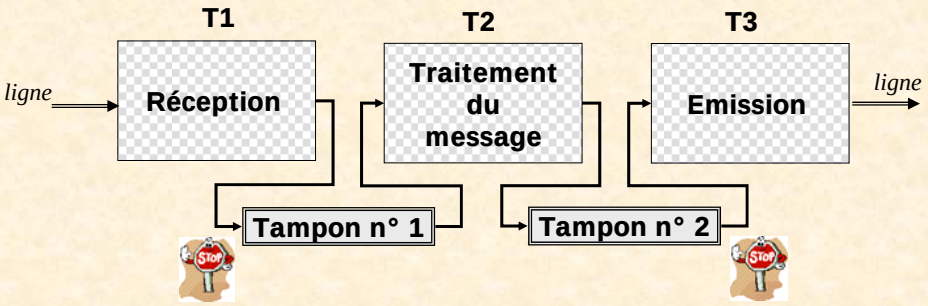
➢ **Programmation concurrente**



EXEMPLE 2



- Traitement de messages : programmation concurrente



- temps total < t1 + t2 + t3
- réception pendant le traitement et l'émission

EN RESUME ...

- Synchronisation : blocage d'une tâche tant qu'une autre tâche ne la réveille pas
- Communication : échange de données entre tâches

Des contraintes ...

Des moyens à mettre en œuvre ...

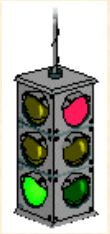
CONTRAINTES

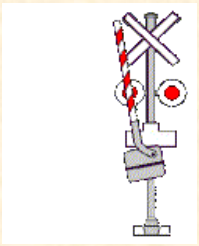
- On ne peut pas prévoir a priori l'ordonnancement ...
- Une solution ne doit pas dépendre de la vitesse d'exécution, ni de l'ordonnancement des tâches
- Le couplage entre tâches doit être le plus faible possible
  L'objectif est d'atteindre un parallélisme maximal
- Les solutions doivent être équitables (attention à la famine !)

PARALLELISME ET DETERMINISME

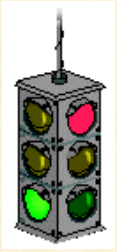
- Un système monotâche est toujours déterminé
- Un système composé de plusieurs tâches est déterminé si toutes ses tâches sont deux à deux non interférentes
- Le non déterminisme peut survenir par les zones mémoire partagées
- Nécessité de retrouver les conditions de l'exécution monotâche pour l'accès aux ressources partagées

SYNCHRONISATION





PROBLEMES TYPES



- Exclusion mutuelle
- Cohorte
- Passage de témoin
- Producteur – consommateur
- Lecteurs - rédacteurs

Problèmes de **synchronisation** pour lesquels il existe une solution type (pattern)

SYNCHRONISATION : PROBLEMES TYPE

EXCLUSION MUTUELLE

- Protection d'une ressource commune (mémoire, fichier, ressource logicielle, matériel) accédée par plusieurs tâches :

Ressource critique

- Exemple :
  - 2 tâches chargées de la réservation de places d'avion s'exécutent concurremment
  - Elles demandent la réservation de la place i

SYNCHRONISATION : PROBLEMES TYPE

EXCLUSION MUTUELLE

- Exemple

	Tâche 1	Etat de la place i	Tâche 2	
1	place i libre ?	libre		
		libre		
2 préemption →		libre	place i libre ?	3
		libre	occupation place i	4
		occupée	Edition billet place i	5
6	occupation place i	occupée		
7	Edition billet place i			

La place i est occupée 2 fois
2 billets sont édités pour la même place ...

SYNCHRONISATION : PROBLEMES TYPE

EXCLUSION MUTUELLE

René Descartes booking a flight ticket

Source : Geek&Poke

SYNCHRONISATION : PROBLEMES TYPE

EXCLUSION MUTUELLE

- **Section critique** = ensemble d'instructions qui accèdent à une ressource critique
- Une seule tâche à la fois doit exécuter la section critique relative à une ressource donnée

=> exécution monotâche de la section critique

SYNCHRONISATION : PROBLEMES TYPE

EXCLUSION MUTUELLE

- Avant d'entrer dans la section critique
  - est-ce qu'une autre tâche est en train d'exécuter la sienne ?
    - Si oui : attente
    - Si non : signaler l'entrée en section critique
- En sortant de la section critique:
  - signaler ce fait aux autres tâches
  - une autre tâche en attente peut alors être libérée

SYNCHRONISATION : PROBLEMES TYPE

EXCLUSION MUTUELLE

Ne pas monopoliser le processeur !

Eviter l'attente active, l'objectif reste la parallélisation maximale

Hypothèse de base : toute tâche sort de la section critique au bout d'un temps fini

EXCLUSION MUTUELLE : SOLUTIONS

- Quel type de solution ?
  - Naïve : utiliser une variable booléenne partagée

```
si libre == TRUE
        libre = FALSE
sinon
        attente
```

Ne fonctionne pas !!!

- Autres solutions, matérielles ou logicielles ...

EXCLUSION MUTUELLE : SOLUTIONS MATERIELLES

- Masquage des interruptions
  - Uniquement valable en monoprocesseur
  - Risque de perte d'IT dans le cas d'une section critique longue
  - Risque de retard de l'horloge système
  - Que faire si une E/S est englobée dans la section critique (cas fréquent) ?
  - Ne peut être effectuée qu'en mode noyau (-> nécessite un appel système spécifique)

SYNCHRONISATION : PROBLEMES TYPE

EXCLUSION MUTUELLE :
SOLUTIONS MATERIELLES

- Instructions spéciales
  - Instructions du processeur qui offrent une solution d'exclusion mutuelle
  - Garantissent l'accès atomique à une cellule mémoire partagée
  - Test and Set (TAS)
  - Echange du contenu de 2 variables (SWAP)
  - risque de congestion du bus mémoire, peu équitable, attente active, non standard

SYNCHRONISATION : PROBLEMES TYPE

EXCLUSION MUTUELLE :
SOLUTIONS MATERIELLES

- En résumé
  - Pas des solutions pour les applications
  - Utilisées par les systèmes pour la mise en œuvre de solutions de plus haut niveau, avec des sections critiques de très courte durée

SYNCHRONISATION : PROBLEMES TYPE

EXCLUSION MUTUELLE :
SOLUTIONS LOGICIELLES

- Attente active
  - algorithme de Dekker, pour 2 tâches, 1965
  - algorithme de Peterson, optimal pour 2 tâches, 1981
  - algorithme de Lamport, pour n tâches, 1974
  - solutions lourdes, complexes et qui immobilisent le processeur
  - A éviter dans les applications
  - Utilisées dans les systèmes pour la mise en œuvre de solutions de plus haut niveau, avec des sections critiques de très courte durée

SYNCHRONISATION : PROBLEMES TYPE

EXCLUSION MUTUELLE :
SOLUTIONS LOGICIELLES

- Sémaphores d'exclusion mutuelle (mutex)
  - Dijkstra (1965)
  - Variable globale partagée, gérée par le système, associée à une file d'attente
  - 3 opérations atomiques :

    I(S, x) : Initialisation de S à la valeur x

    P(S) : si (S > 0) S--; sinon attente

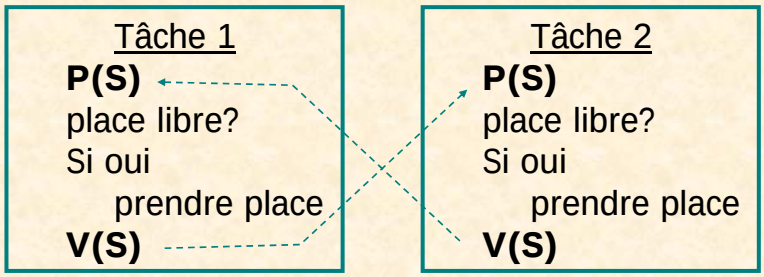
    V(S) : S++; réveil d'une tâche en attente

EXCLUSION MUTUELLE : LE MUTEX

- Exemple

  2 tâches chargées de la réservation de places d'avion s'exécutent concurremment

I(S, 1)



COHORTE

- Utilisation d'un groupe de ressources banalisées de taille bornée

  Chaque ressource est offerte ou utilisée par une tâche (N serveurs ou N clients)

- Exemple

  L'entrée d'un parking de N places est contrôlée par une barrière qui reste fermée quand le parking est plein

COHORTE : SOLUTION

- Utilisation d'un sémaphore comme compteur de la ressource « place de parking disponible »

I(S, N)

Entrée du parking	**Sortie du parking**
P(S) ouverture barrière	V(S)

PASSAGE DE TEMOINS

- Coopération par division du travail entre tâches
- 3 types

  Envoi d'un signal : avertir de la disponibilité d'une information ou d'une autorisation

  Rendez-vous : établir un point de synchronisation

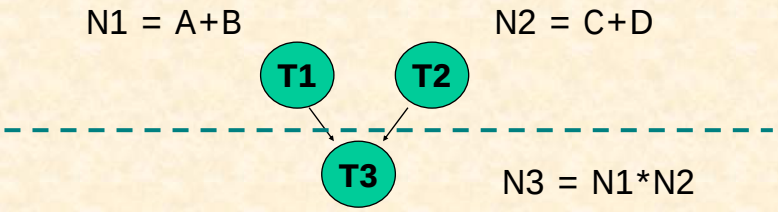
  Appel procédural

SYNCHRONISATION : PROBLEMES TYPE

PASSAGE DE TEMOINS

Envoi d'un signal

- Exemple



SYNCHRONISATION : PROBLEMES TYPE

PASSAGE DE TEMOINS

Envoi d'un signal

- Solution

I(S1, 0) I(S2, 0)

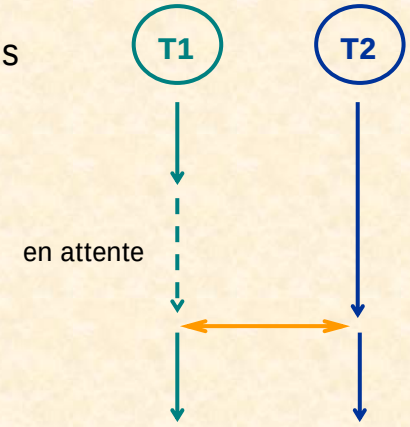
T1	**T2**	**T3**
N1 = A+B	N2 = C+D	P(S1)
V(S1)	V(S2)	P(S2)
		N3 = N1*N2

SYNCHRONISATION : PROBLEMES TYPE

PASSAGE DE TEMOINS

Rendez-vous

- Permet à 2 processus d'établir un point de synchronisation



SYNCHRONISATION : PROBLEMES TYPE

PASSAGE DE TEMOINS

Rendez-vous

- Solution : barrière de synchronisation

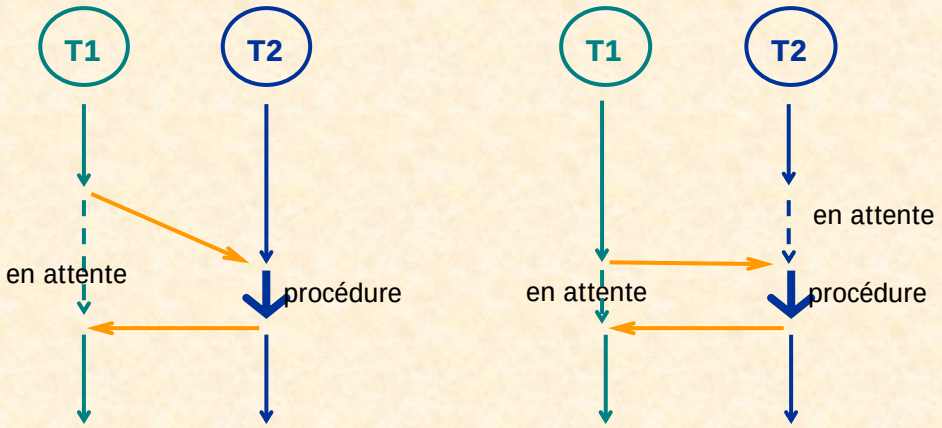
I(S1, 0) I(S2, 0)

T1	T2
V(S1)	P(S1)
P(S2)	V(S2)

SYNCHRONISATION : PROBLEMES TYPE

PASSAGE DE TEMOINS

Appel de procédure Exemple



SYNCHRONISATION : PROBLEMES TYPE

PASSAGE DE TEMOINS

Appel de procédure

- Solution

I(S1, 0) I(S2, 0)

T1	T2
V(S1)	P(S1)
P(S2)	...
	V(S2)

SYNCHRONISATION : PROBLEMES TYPE

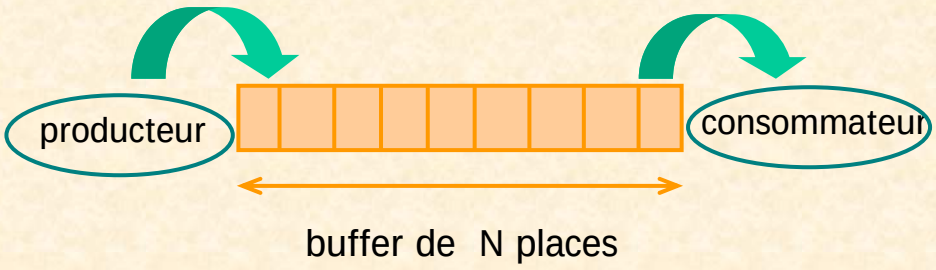
PRODUCTEUR - CONSOMMATEUR

- Une ou plusieurs tâches productrices produisent des ressources à destination d'une ou plusieurs tâches consommatrices
- Les ressources doivent être consommées dans l'ordre de production
- S'il n'y a pas de ressource disponible, le consommateur est mis en attente
- Si la file de transfert des ressources est pleine, le producteur est mis en attente

SYNCHRONISATION : PROBLEMES TYPE

PRODUCTEUR - CONSOMMATEUR

Gestion d'une file d'attente en mémoire



- Problème sous-jacent à toute relation client/serveur

PRODUCTEUR - CONSOMMATEUR

- Solution

I(placeLibre, N) I(infoDispo, 0)

Producteur	**Consommateur**
P(placeLibre) écriture V(infoDispo)	P(infoDispo) lecture V(placeLibre)

LECTEURS - REDACTEURS

- Problème type de la gestion de bases de données
- Plusieurs tâches accèdent à la base pour y lire (**lecteurs**) ou pour y écrire (**rédacteurs**)
- La lecture est interdite pendant l'écriture
- Deux écritures simultanées sont interdites
- Deux lectures simultanées sont autorisées
- Pas d'exclusion systématique des accès à la base

SYNCHRONISATION : PROBLEMES TYPE

LECTEURS - REDACTEURS

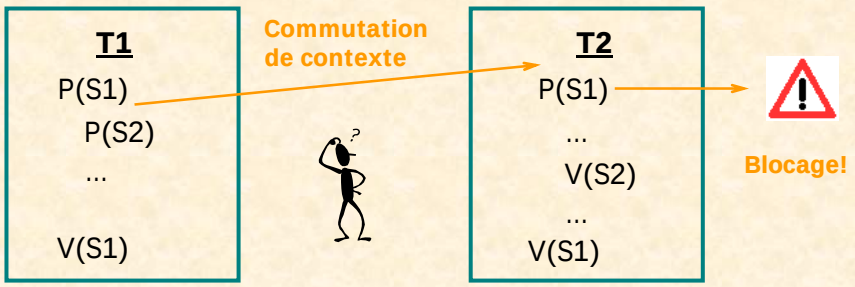
- Stratégies possibles
  - priorité aux lecteurs (risque de famine des rédacteurs)
  - priorité aux rédacteurs (risque de famine des lecteurs)
  - FIFO (pas de famine, mais pas optimal)
  - optimiste : contrôle effectué une fois l'opération réalisée. Annulation si conflit détecté. (concurrence maximale et pas d'interblocage, mais risque d'abandons multiples)

SYNCHRONISATION

INTERBLOCAGE (DEADLOCK)

- Cas limite où 2 tâches en concurrence peuvent être bloquées

I(S1, 1) I(S2, 0)



INTERBLOCAGE (DEADLOCK)

- Causes possibles
  - Erreur de programmation
  - Utilisation d'outils de synchronisation inadaptés
  - Crash d'une tâche qui détient une ressource
  - ...

LES SEMAPHORES : LIMITES

- Une seule condition de blocage/réveil à la fois : risque d'interblocage
- Ne répondent pas à tous les problèmes de la programmation concurrente
- Programmation de bas niveau, non portable
- Erreurs de programmation faciles et fatales !!!
- Une solution : les moniteurs de Hoare

SYNCHRONISATION

LES MONITEURS

- Définition : données et fonctions qui s'exécutent en exclusion mutuelle
- Région critique conditionnelle
- Hoare, 1974 et Brinch-Hansen, 1975
- Exclusion mutuelle + mécanismes explicites de blocage et de réveil
- Permet d'associer une expression booléenne (condition de blocage multiple) et une file d'attente

SYNCHRONISATION

LES MONITEURS

- Principes

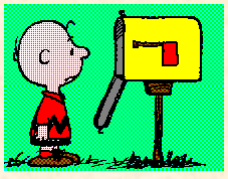
  - Une seule tâche peut exécuter du code inclus dans le moniteur à un instant donné. Les autres sont mises en attente
  - Une tâche en attente doit libérer le moniteur
  - Exclusion mutuelle + condition de blocage multiple + file d'attente + mécanismes explicites de blocage et de réveil

LES MONITEURS

- Implémentations
  - Méthodes « synchronized » en Java
  - Objets « protected » en Ada
  - Variables conditionnelles Posix

COMMUNICATION

= Échange de données entre tâches



COMMUNICATION

- De nombreux mécanismes possibles
- Critères de choix
  - volume des données
  - lecture destructrice
  - vitesse
  - synchronisation
  - structure de l'information
  - existence indépendante

MISE EN OEUVRE

- En utilisant directement les services offerts par le système d'exploitation (appels système)
- En utilisant les mécanismes offerts par certains langages
  -  ADA : tasks
  - JAVA : threads
- En utilisant des compilateurs et des bibliothèques générant du code parallélisable