Chapitre 2 :

La programmation concurrente

1- Définition

Un programme concurrent est un programme non séquentiel,

- c'est la situation du parallélisme, avec un seul processeur (au départ),
- puis avec plusieurs processeurs, appelée *programmation parallèle*.

Son utilité est qu'elle permet d'exploiter pleinement la possibilité qu'ont le processeur et les périphériques, de travailler simultanément.

==> Considérons l'expression arithmétique
(a + b) * (c + d)

Une évaluation séquentielle de cette expression consiste à calculer

- d'abord (a + b),
- puis (c + d)

Et finalement
- le produit (a + b) * (c + d)

Une évaluation concurrente de cette expression consiste à calculer

- simultanément (a + b) et (c + d)
- puis de calculer leur produit.

L'avantage de cette évaluation concurrente est évident si nous disposons de **2** processeurs :

Un 1er peut évaluer (a + b) en même temps que l'autre évalue (c + d)
→ le temps total nécessaire à l'évaluation de l'expression
(a + b) * (c + d) se trouve ainsi diminué.

Que se passe-t-il si nous ne disposons que d'un seul processeur ? :

Évaluer de façon concurrente (a + b) et (c + d) doit être interprété dans ce cas de la manière suivante :
- le processeur est utilisé en une fraction de seconde pour évaluer (a +b)
- puis une fraction de seconde pour évaluer (c + d),
- puis une fraction de seconde pour continuer d'évaluer (a + b)
- et ainsi de suite.

Est-il avantageux de procéder ainsi ?

Certainement pas dans ce cas, et pourtant, c'est une technique importante, même dans le cas d'un seul processeur :
par exemple, le problème de la gestion des entrées/sorties :
la programmation concurrente permet d'exploiter pleinement la possibilité qu'ont le processeur et les périphériques de travailler en parallèle :

pendant que le processeur attend que l'interface soit prêtre à accepter un caractère (l'écran, l'imprimante), il est oisif, et ceci peut être évité en lui fournissant du travail, qu'il accomplira apparemment simultanément (ou qu'il attend de recevoir un caractère).

Dès que le périphérique est prêt, il le signale au processus (par une interruption),

et le processeur accomplit la tâche qui était en attente, en laissant l'autre tâche de coté.

2. Exemple concret :

L'atelier de coupe de bois (de menuisier)

Dans cet atelier, se trouve un menuisier qui exécute des commandes transmises par des clients

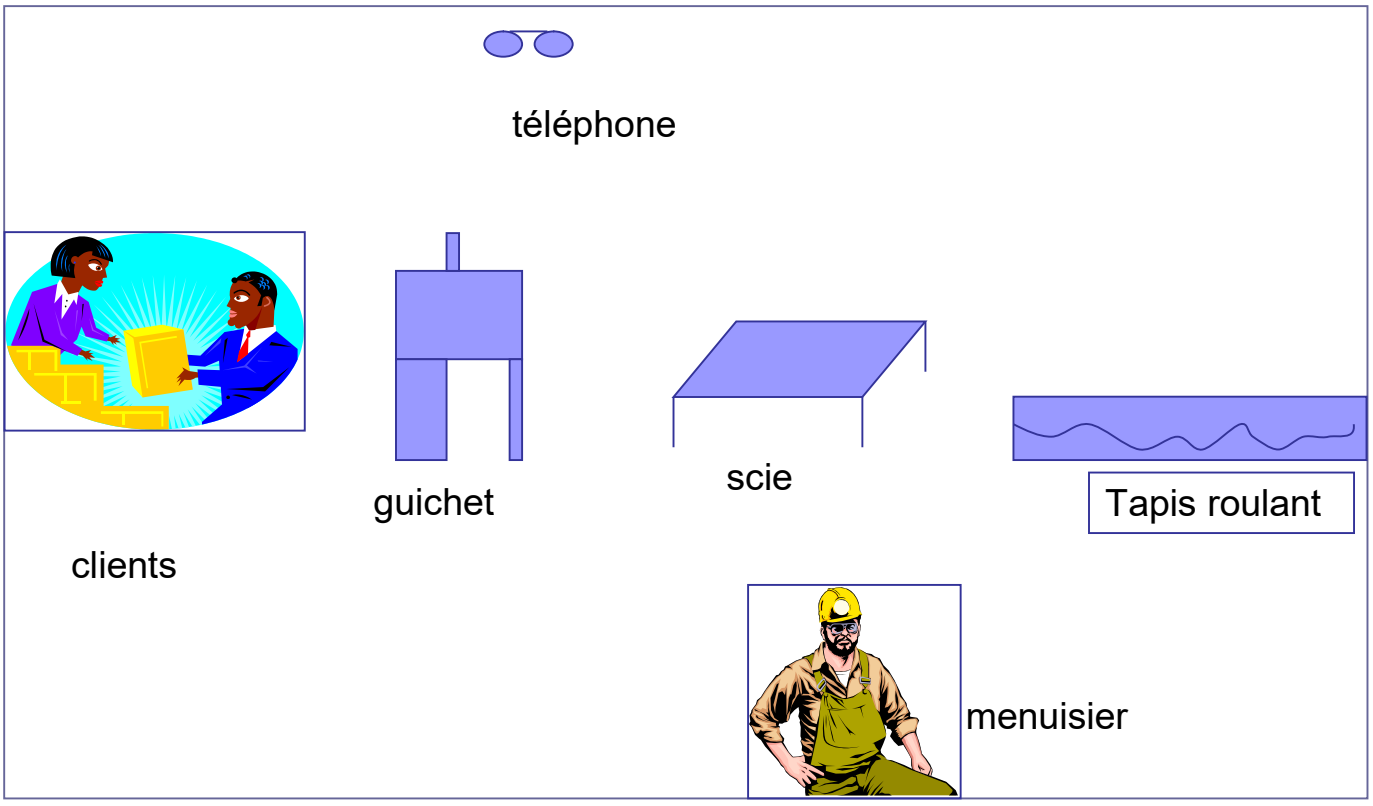
L'activité du menuisier peut être assimilée à l'exécution d'un programme sur un processeur.

L'atelier comporte :

* 2 périphériques d'entrées :
  - un *téléphone*, utilisé par les clients pour transmettre une commande
  - un *guichet*, derrière lequel les clients font la queue, pour transmettre également une commande

- et un périphérique de sortie:
  - un tapis roulant sur lequel le menuisier dépose le bois scié, selon les mesures indiquées par les clients.

On ne s'intéresse pas à la façon dont le bois est acheminé au client destinataire.
Une seule commande sur le tapis à la fois.

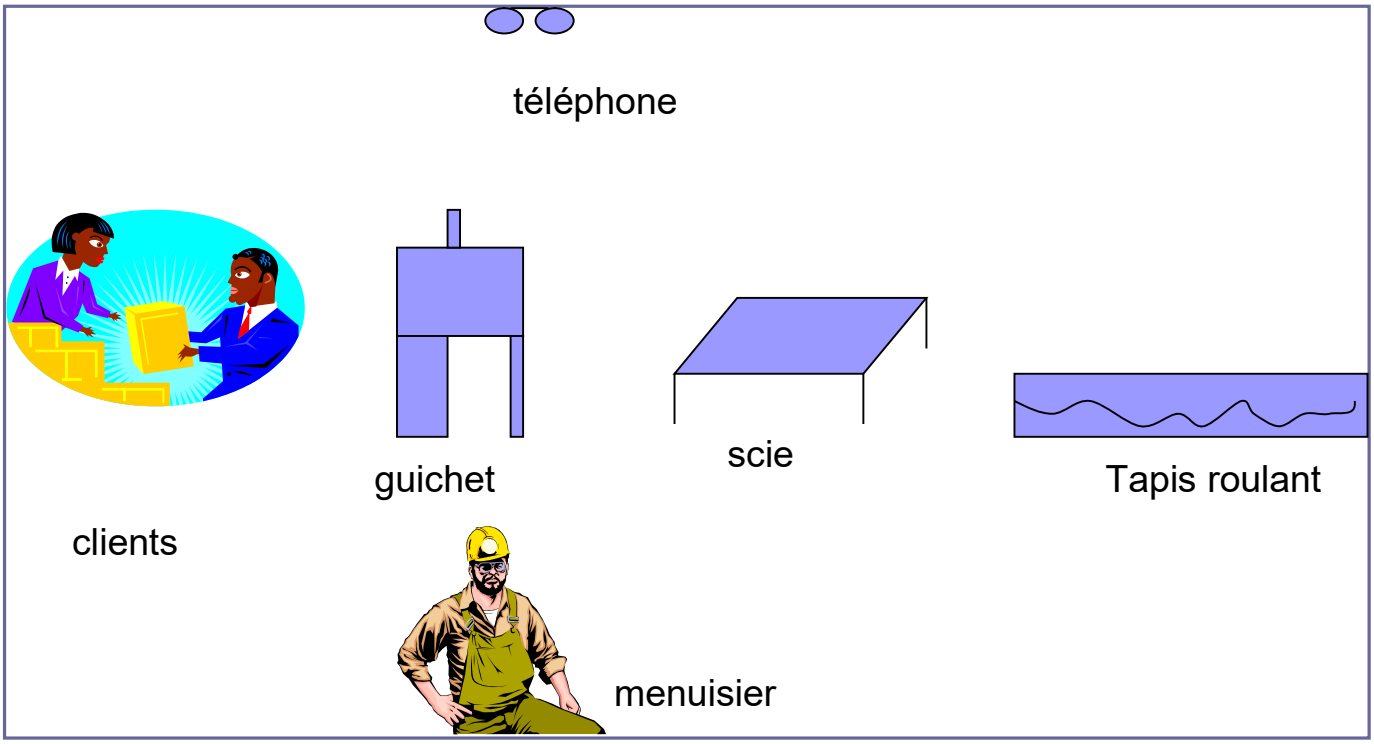


Une programmation séquentielle de l'atelier correspond au traitement suivant, en supposant que *téléphone* et *guichet* sont dépourvus (pas) de sonnerie

1. le menuisier attend une commande en allant alternativement
   - décrocher le téléphone, au cas où un client se trouverait au bout de fil,
   - ou ouvrir le guichet au cas où un client serait arrivé.

2. Dès que le menuisier obtient une commande, il l'exécute :
   il scie le bois selon les dimensions exigées par le client.

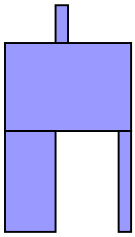
3. Lorsque la coupe est terminée :
   Le menuisier se rend vers le tapis roulant pour y déposer le bois coupé.
   S'il n'y a pas de place à l'extrémité du tapis roulant, (la commande précédente s'y trouve encore),
   alors le menuisier attend avant de pouvoir déposer sa livraison.
   Dans ce cas, il est à noter une perte de temps du menuisier qui reste inactif alors que éventuellement, il y a des clients qui attendent au téléphone ou au guichet.



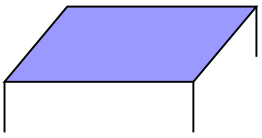
téléphone

menuisier

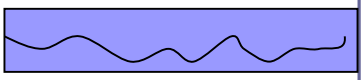
clients



guichet



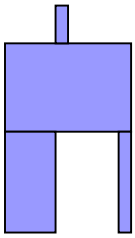
scie



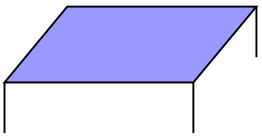
Tapis roulant

téléphone

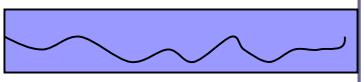
clients



guichet



scie

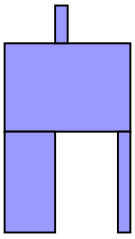


Tapis roulant

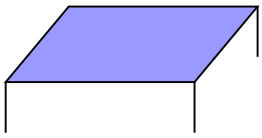
menuisier

téléphone

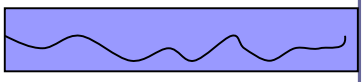
clients



guichet



scie



Tapis roulant

menuisier

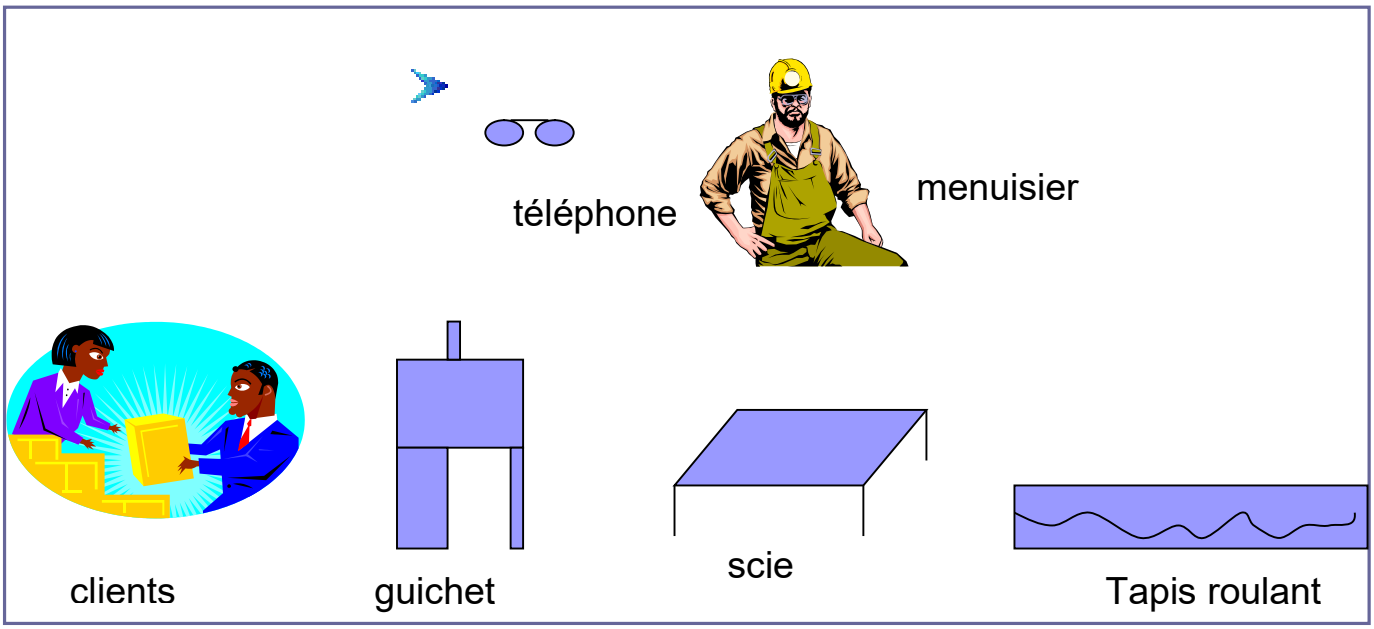
On passe d'une programmation séquentielle à une programmation concurrente de l'atelier dès que l'on introduit des sonneries

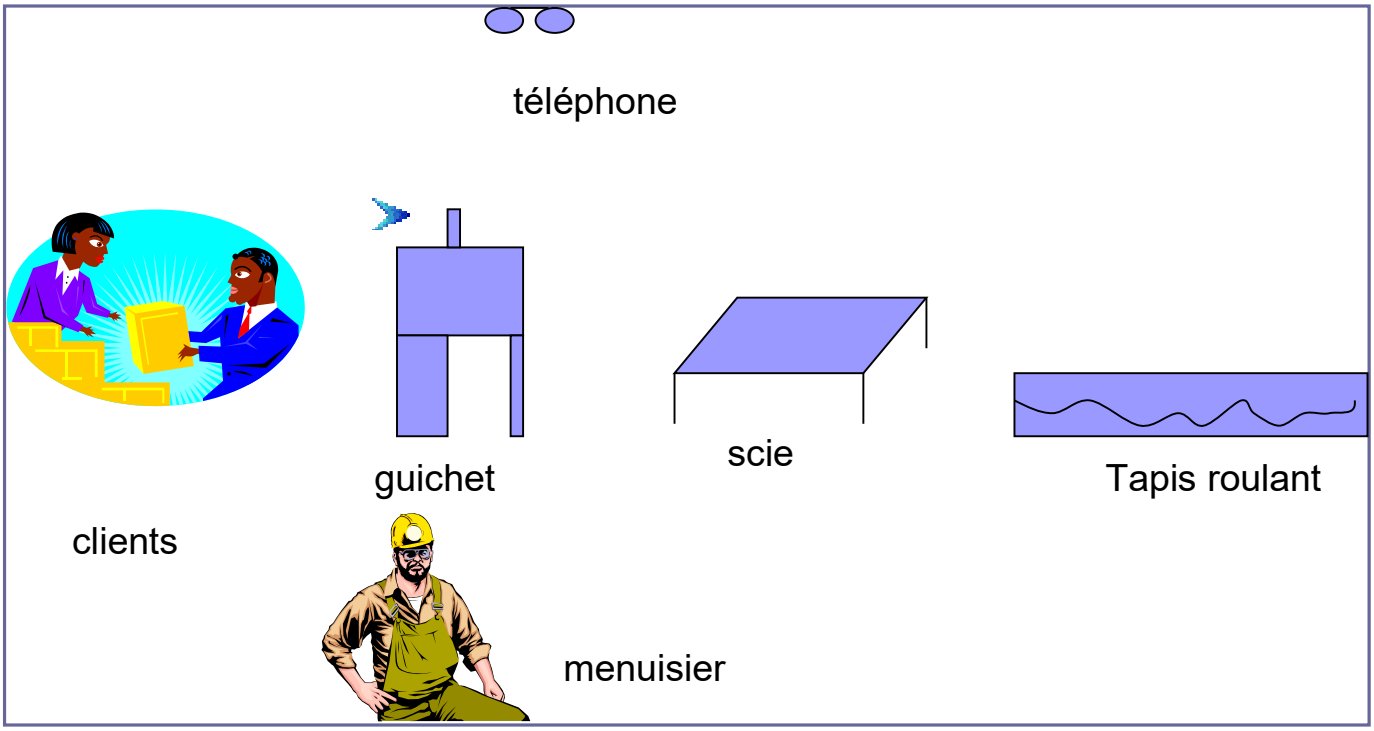
Les sonneries sont celles :

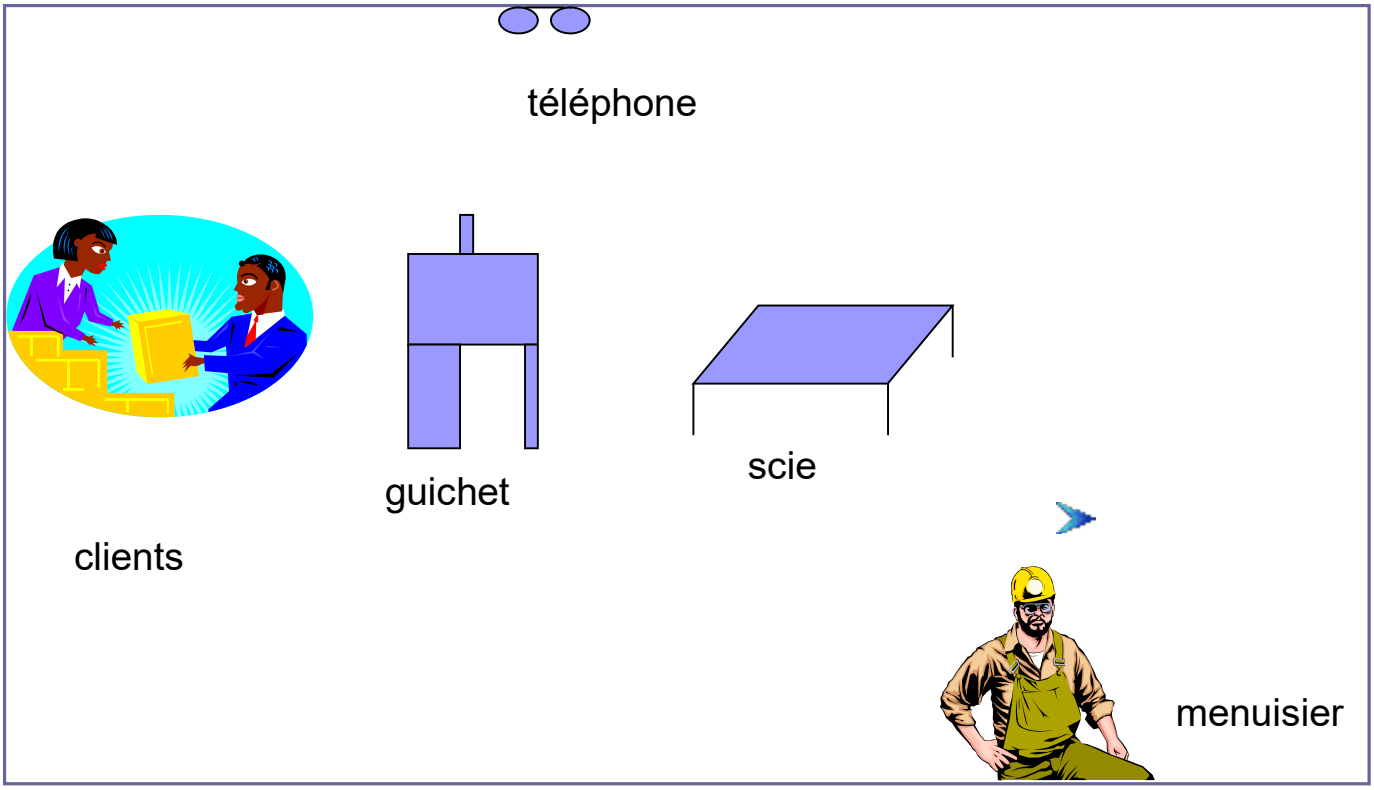
- du téléphone
- du guichet : un client arrive au guichet et sonne
- du tapis roulant : dès qu'il y a une place disponible dans le tapis

L'existence de sonnerie entraîne une organisation différente du travail de menuisier :
cette nouvelle organisation se caractérise par l'introduction :

- d'un carnet de commandes
- d'une étagère sur laquelle est déposée le bois coupé







L'activité principale du menuisier se déroulera alors comme suit :

1. il consulte son carnet de commandes
2. il exécute la commande la plus ancienne
3. il dépose le bois scié sur l'étagère prévue à cet effet ou directement sur le tapis roulant s'il est libre.

Mais, le menuisier est interpellé régulièrement par des sonneries : lorsqu'une sonnerie retentit, le menuisier interrompt le travail en cours, et le reprend ultérieurement

- dans le cas du téléphone ou guichet, le menuisier note la commande dans son carnet
- dans le cas du tapis roulant, il prend le bois de l'étagère et le dépose sur le tapis.

Il est évident que ce mode de fonctionnement concurrent, plus organisé, est plus efficace que le séquentiel où aucune organisation ne s'impose à l'exception de suivre les 3 actions citées successivement.

Les avantages de ce fonctionnement concurrent sont les suivants :

- le menuisier n'est plus oisif en attendant de déposer le bois scié sur le tapis roulant

- les clients ne s'impatientent plus derrière le guichet ou au téléphone : les commandes sont prises au fur et à mesure de l'arrivée des clients, et non plus au rythme où elles peuvent être traitées.

Néanmoins

Ce fonctionnement n'est pas aussi judicieux qu'on le croit, l'inconvénient est que la complexité du programme va croître considérablement si on ajoute des périphériques (un autre téléphone par exemple).

Une programmation concurrente judicieuse de l'atelier consistera à introduire **3** personnes supplémentaires

(P1) – 1 au guichet
(P2) – 1 au téléphone
(P3) – 1 au tapis roulant

(P1) - ouvre le guichet dès que la sonnerie retentit
- inscrit la commande dans le carnet
- referme le guichet
- attend la sonnerie suivante

(P2) fait de même pour les commandes par téléphone

(P3) transporte à chaque sonnerie, le bois scié de l'étagère au tapis roulant

Le menuisier travaille alors de manière identique à celle d'avant l'introduction des sonneries (cas séquentiel) sauf que :

- il cherche les commandes directement du carnet, et ne va plus au guichet ni au téléphone
- le bois scié est déposé sur l'étagère, et non pas sur le tapis

Ainsi, le menuisier et les personnes qui sont au guichet, au téléphone, au tapis sont appelés des processus.

S'il existe un seul processeur, à tout instant, un seul processus à la fois peut être en exécution.

Dans notre exemple, une seule personne des 4 peut être active à tout instant.

La sonnerie implique un changement d'activité.

Dès qu'elle retentit, la personne active P1 s'immobilise et la personne P2 concernée par la sonnerie se met en mouvement. Lorsque le travail de P2 est terminé, P2 s'immobilise et l'activité de P1 peut reprendre.

Ceci ne change en rien l'activité de l'atelier si on le film et qu'on le passe en accéléré, on a l'impression que toutes les personnes sont actives simultanément.

C'est exactement ce qui se passe dans un ordinateur, qui fonctionne à l'ordre de la microseconde, et donne l'impression, que des processus concurrent sont exécutés simultanément.

Figure 1 : travail séquentiel du menuisier

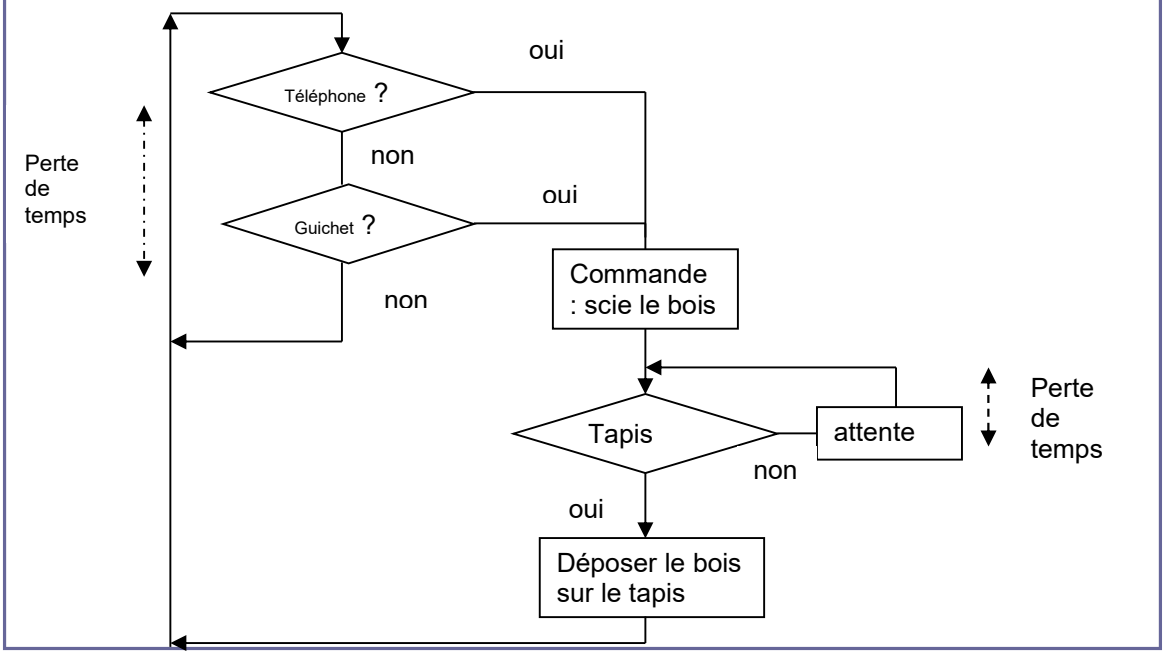
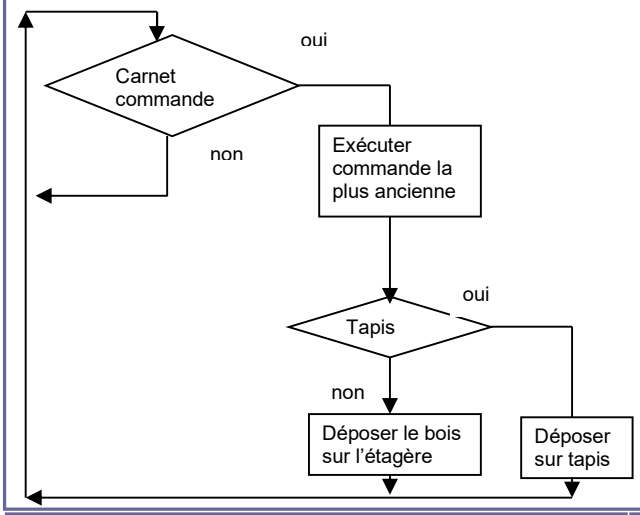


Figure 2 : travail concurrent du menuisier



Dans le cas de sonnerie

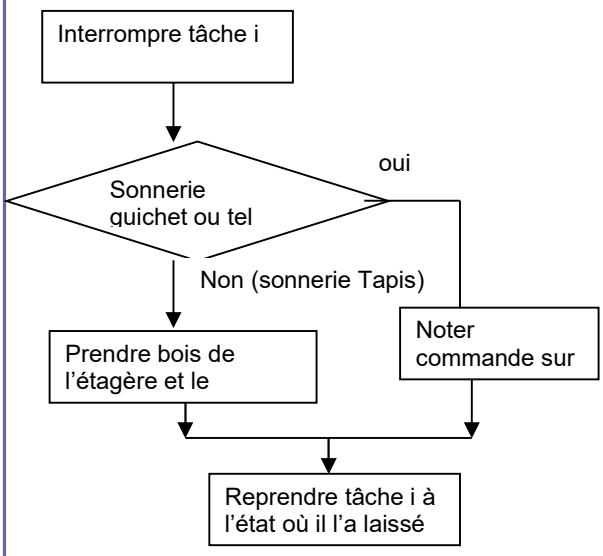
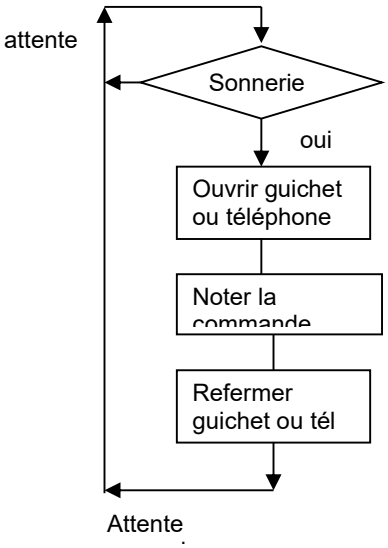
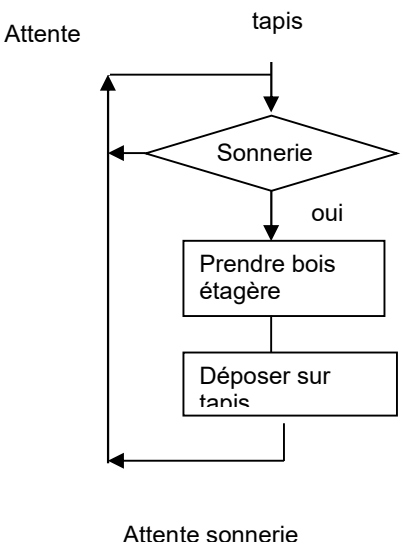


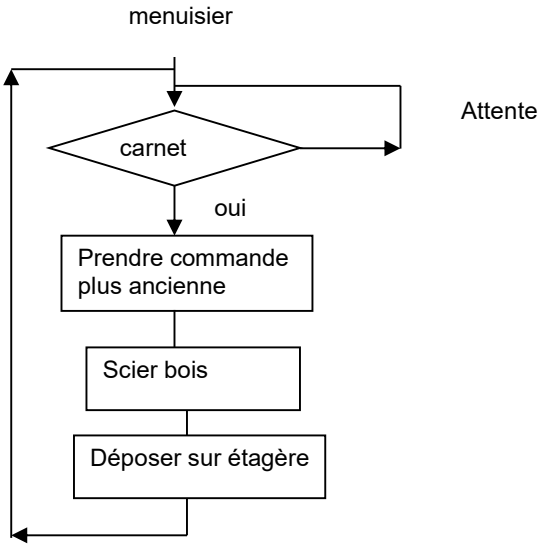
Figure 3 : programmation judicieuse concurrente Guichet ou téléphone



Tapis



Menuisier



3-

3. Exclusion mutuelle et synchronisation

Deux problèmes majeurs sont impliqués par l'organisation concurrente de l'atelier.

- le problème de **l'exclusion mutuelle**
- le problème de la **synchronisation**

Ces problèmes sont communs à tous les programmes concurrents

a) Exclusion mutuelle :

Considérons le carnet de commande du menuisier, et imaginons que la personne responsable du guichet soit entrain d'inscrire une commande.

Si le téléphone sonne au même moment, la personne responsable au téléphone a besoin du même carnet de commandes pour y inscrire la nouvelle commande.

Ceci risque alors d'entraîner des inscriptions incohérentes sur le carnet si les 2 y inscrivent en même temps

Il faut donc empêcher que le carnet soit subtilisé pendant qu'une commande est entrain d'y être inscrite

En d'autres termes, les activités de 1) et 2) s'excluent mutuellement, c'est-à-dire, ne peuvent avoir lieu en même temps.

b) Synchronisation

Considérons l'activité du menuisier : celui ci doit arrêter de travailler lorsque son **carnet de commandes** est vide, ou que **l'étagère** est pleine ;

il doit reprendre son travail dès qu'une commande est reçue, ou qu'il y a de la place sur l'étagère.

De manière générale le problème de synchronisation se pose chaque fois que plusieurs processus interviennent dans une chaîne de traitement

Pour l'atelier, la chaîne de traitement est constituée par les personnes du guichet et du téléphone, le menuisier, et la personne responsable du tapis.

Les problèmes de synchronisation et d'exclusion mutuelle se résolvent par des outils ou des mécanismes tels que :

- ***verrous***,
- ***événements***,
- ***sémaphores***,
- ***moniteurs***,
- ***rendez-vous***,
- etc…

4. Liens avec la programmation en temps réel

Les deux programmations sont proches, celle en temps réel comporte une dimension supplémentaire : celle de contrôle d'un système externe, et donc, des contraintes de temps

Exemples de systèmes contrôlés par un programme en temps réel :

- robots industriels,
- système de commandes de chauffage d'un immeuble,
- centraux téléphoniques,
- systèmes de contrôle d'expériences en laboratoire,
- système de contrôle de navigation aérienne, etc…

Exemple concret de commande de chauffage
(système automatique d'immeuble):

a. les périphériques correspondent à
   - entrée : capteurs de température
   - sortie : actionneurs, permettent d'enclencher ou de déclencher une ou des chaudières
   -

b. le rôle du programme : sera de maintenir la température à l'intérieur de l'immeuble, à 20 °C la journée, et à 16°C la nuit. Pour ceci, le programme doit réagir en temps réel, c-à-d dans un délai maximum à ne pas dépasser, déterminé par le système externe à contrôler, à toute modification de température, par l'envoi de commandes à la chaudière.

Cette caractéristique est connue à tous les programmes en temps réel.

- le programme reçoit des informations du système externe à contrôler
- le programme traite ces données
- le programme réagit par l'envoi de commandes au système à contrôler.

5- Les langages de la programmation concurrente

Ce sont des langages qui ont des instructions et des moyens d'écrire entièrement un programme concurrent, ou en temps réel (où le temps intervient comme une composante à part).

Ceci n'a pas toujours été ainsi.

L'écriture de programmes en temps réel (ont été plus demandés que des programmes concurrents utilisés pour système), a longtemps passé par l'utilisation de systèmes d'exploitations multi-tâches mettant à disposition des programmes d'application, les outils adéquats (notamment les outils de synchronisation.

Un programme en temps réel était écrit en langage d'assemblage ou dans un langage évolué, séquentiel, mais incluant des appels au système d'exploitation

Exemple des langages de ce type :
CORAL 66,
WOODWARD 70
et RTL/2

Les langages incluant des notions concurrentes sont apparus dans la première moitié des années 70 par exemple : PEARL et PASCAL CONCURRENT, les langages PORTAL et MODULA-2, et ADA sont plus récents.

1- le langage **PORTAL**, par l'entreprise Landès & Gyr, en 1978, l'aspect concurrent se caractérise par la notion de moniteur.

2- Le langage **MODULA-2** a été proposé par N-Wirth, l'auteur de Pascal, en 1980 (ne pas confondre avec MODULA datant de 1976). Ce langage comprenait aussi la notion de moniteur, dans sa version MODULA, qui disparaît dans MODULA-2 après réflexion de l'auteur, ce dernier ne contenant aucun mécanisme de synchronisation, offrant la possibilité à l'utilisateur de construire le mécanisme de synchronisation de son choix.

3- Le langage **ADA** est l'aboutissement d'un projet lancé en 1975 par le département de la défense des USA, dont le but étant de trouver un langage unique pour toutes les applications. C'est donc ADA qui a été retenu, en 1979, par une équipe dirigée par J. Ichbiak, développé chez CII Honewell/Bull
L'aspect concurrent de ADA se caractérise par la notion de rendez-vous.