2. Systèmes synchrones

Nous avons vu qu'un système synchrone est composé de phases qui s'exécutent simultanément. Nous étudions tout d'abord les mécanismes de base des phases de communication avant d'étudier des systèmes synchrones particuliers.

2.1 Accès synchrone ou asynchrone à des ressources partagées

Considérons des processus P^i qui ont accès à des variables partagées stockées dans une mémoire M. Le code des processus est schématiquement le suivant

Processus P^i
Répéter
Calculer R(i)
Lire M[R(i)]
Calculer R(i)
Ecrire M[R(i)]

Chaque processus est composé de quatre phases. Les phases de lecture et d'écriture sont les celles pendant lesquelles les processus interagissent. Cette interaction peut poser des difficultés si les processus veulent lire ou écrire simultanément la même variable.

Dans le cas synchrone, on distinguera deux politiques d'accès : **exclusive ou concurrente**. La politique d'accès exclusif doit garantir que deux processus n'accèderons pas simultanément à une variable. Une telle garantie, en général, ne pourra être assurée que par le programmeur. Dans le cas d'accès concurrent, la politique d'accès doit offrir une garantie de résultat. Par exemple, si deux processeurs veulent écrire des valeurs différentes dans la même variable, on peut garantir que l'une des deux sera écrite (mais on ne sait pas laquelle) ou bien qu'aucune ne le sera.

Dans le cas asynchrone, on doit garantir que les accès n'ont pas lieu simultanément. Dans ce cas, il ne faut autoriser qu'un seul processeur à accéder à la variable. Le problème correspondant est appelé **exclusion mutuelle**. Ce problème sera traité dans un prochain chapitre

2.2 Communication synchrone ou asynchrone

Considérons tout d'abord le cas de deux processus qui interagissent par échanges de données (ou de messages). Les primitives Lire et Ecrire sont remplacées par Recevoir et Envoyer. L'envoi consiste à placer une donnée dans une mémoire de communication (buffer), la réception à récupérer la donnée dans cette mémoire.

Dans le cas synchrone, l'envoi d'un message interrompt l'activité du processus émetteur jusqu'à la réception par le processus destinataire. On parle de **communication synchrone ou par rendez-vous**. Un exemple est la communication téléphonique (sans messagerie).

Dans le cas asynchrone, l'émetteur envoie le message et entame une autre phase (de calcul ou de communication) indépendamment de l'action du destinataire. Un exemple est la communication par courrier postal.

2.3 Parallel Random Access Machine (PRAM)

Une PRAM est un modèle de système parallèle synchrone communiquant par une mémoire partagée.

Définition 2.1 : Une PRAM est un ensemble de **n** processus identiques, numérotés de 0 à n-1, qui accèdent de manière synchrone à une mémoire partagée composée de n cases, numérotées de 0 à n-1.

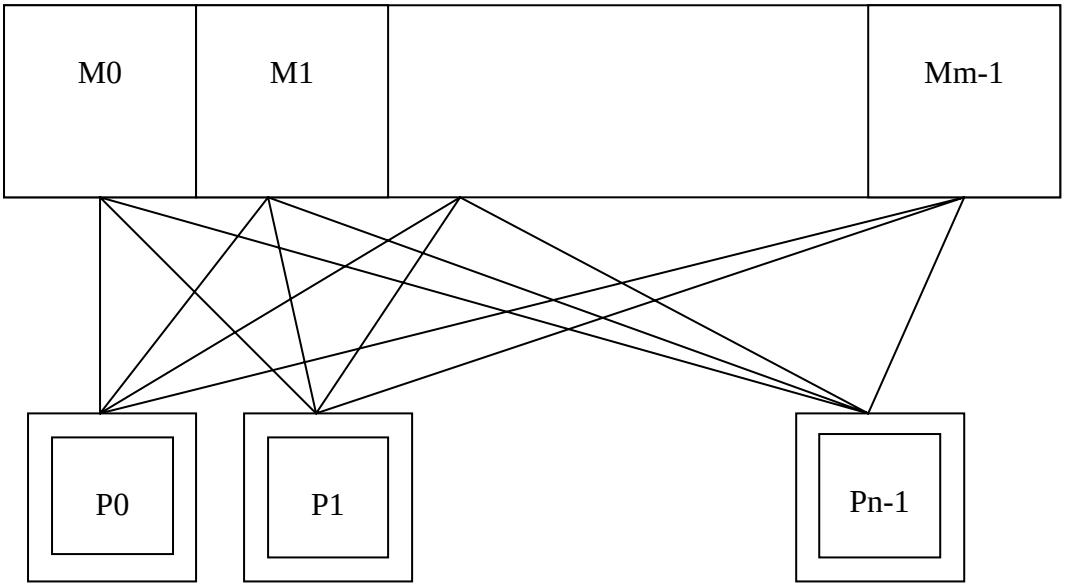


Figure 2.1 : PRAM

Selon les politiques d'accès à la mémoire, on peut obtenir quatre types de PRAM :

- EREW (Excluvive Read Exclusive Write) : deux processus ne peuvent pas lire ou écrire simultanément dans la même case mémoire
- CRCW (Concurrent Read Concurrent Write) : deux processus peuvent lire ou écrire simultanément dans la même case mémoire
- CREW : deux processus peuvent lire mais ne écrire simultanément dans la même case mémoire
- ERCW

Les modèles les plus couramment utilisés dont EREW et CREW.

Définition 2.2 : La complexité d'un programme PRAM est donnée par la valeur de l'horloge lors de l'exécution de la dernière phase des processus.

2.4 Réseaux de processus

Un réseau de processus est un modèle de système parallèle synchrone communiquant par échanges de messages. On suppose donc que les processus sont les sommets d'un graphe **G = (V,E)**. Si **(i,j)** est un arc entre les sommets i et j, alors le processus **P^i peut envoyer un message à P^j** . Le système fonctionne de manière synchrone en enchaînant les opérations Envoi-Réception de message (phase de communication) et Calcul (phase de calcul). Si les buffers d'envoi ou de réception sont vides, aucune donnée n'est envoyée ou reçue (nul). Sinon un message est un élément non nul de l'ensemble des messages possibles, noté M.

La phase de calcul est définie par la fonction de transition qui, étant donné l'état du processus et les messages reçus au temps t-1, associe l'état au temps t. L'ensemble des états de $\mathbf{P^i}$ sera noté $\mathbf{S_i}$. La fonction de transition se modélise donc par :

$$\mathbf{\Delta_i : S_i \times M^{in(i)} \rightarrow S_i}$$

où in(i) est le degré entrant de $\mathbf{V_i}$.

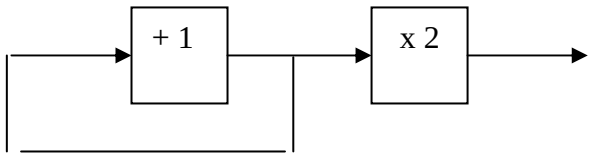
La phase de communication est définie par la fonction de transition qui, étant donnés l'état du processus au temps t-1 et un nœud successeur, construit le message envoyé au temps t à ce successeur. L'ensemble des successeurs de $\mathbf{P^i}$ sera noté $\mathbf{Succ_i}$. La fonction de transition se modélise donc par :

$$\mathbf{\Gamma_i : S_i \times Succ_i \rightarrow M}$$

Une exécution du réseau de processus sera donc caractérisée par la suite de l'ensemble des états des processus et de l'ensemble des messages. Si nous notons **S(t), Env(t) et Rec(t)** ces ensembles au temps **t**, une exécution sera donc définie par :

S(0), Env(0), Rec(0), S(1), Env(1), Rec(1), …, S(t), Env(t), Rec(t)

Exemple : Générateur des nombres entiers pairs.



Les processus sont :

- P^1 : **début** z := 0 ; **répéter** z := z+1 ; (**envoyer** z ; **recevoir** z) **fin**.
- P^2 : **début** t := 0 ; **répéter** t := 2 t ; (**envoyer** t ; **recevoir** t) **fin**.

Les fonctions de transition correspondantes sont :

- $V = \{1,2,Out\}$, $E = \{(1,1), (1,2), (2,Out)\}$, $Succ_1 = \{1,2\}$, $Succ_2 = \{Out\}$
- $S_1 = S_2 = N$, $M = N \cup \{nul\}$
- $\Delta_1(s,m) = m$; $\Gamma_1(s,1) = \Gamma_1(s,2) = s+1$; $\Delta_2(s,m) = m$; $\Gamma_2(s,1) = 2s$;

Définition 2.3 : La complexité en temps d'un réseau de processus est donnée par la valeur de l'horloge lors de l'exécution de la dernière phase des processus. La complexité en espace est le nombre de processus.

Dans l'exemple précédent, on voit qu'un réseau de processus peut avoir une durée d'exécution non bornée. Pour ce générateur, on constate facilement que la génération des n premiers entiers pairs prend un temps proportionnel à n et un nombre de processus constant (2).

2.5 Election d'un dirigeant (leader) sur un anneau synchrone

L'élection d'un dirigeant dans un réseau synchrone de processus consiste à distinguer un processus parmi n. Un tel problème se rencontre dans de nombreuses applications, la plus connue étant celle de reconstruction du jeton dans le cas d'une perte de jeton dans un système faisant circuler un tel jeton pour avoir accès à des ressources. Le processus distingué crée un nouveau jeton de manière unique.

Supposons que les processus de notre réseau soient connectés sous la forme d'un anneau, c'est-à-dire que G(V,E) est défini par :

Cas unidirectionnel : $\mathbf{E = \{(V_i,V_{i+1}), i = 0, \ldots, n-1\}}$

Cas bidirectionnel : $\mathbf{E = \{(V_i,V_{i+1})}$ **et** $\mathbf{(V_i,V_{i-1}), i = 0, \ldots, n-1\}}$

où les calculs d'indice sont faits modulo n.

Supposons que les processus n'ont aucune information globale sur leur environnement : ils ne connaissent pas n, ni la topologie du graphe, mais uniquement leur(s) voisin(s) par exemple en envoyant (ou en recevant) un message à gauche ou à droite. Pour se distinguer les uns des autres, les processus ne peuvent donc utiliser que leur **Identité Unique (IU)**. Nous supposons que ces IU sont choisies parmi un ensemble de très grands nombres entiers et que deux processus ne possèdent pas la même UI. Les processus possèdent une **variable d'état**, nommée **Etat**, qui peut prendre deux valeurs **d** et **nd** et est initialisée à **nd** (non dirigeant). Il possède une deuxième variable nommée **Dir**, prenant des valeurs entières et initialisée à **0**.

L'état initial du réseau est donc caractérisé par :

$$\mathbf{(EI)\ \forall i = 0, \ldots, n-1, Etat_i\ (0) = nd \text{ et } Dir_i\ (0) = 0.}$$

L'état final du réseau est caractérisé par :

$$\mathbf{(EF)\ \exists k, 0 \leq k \leq n-1, Etat_k\ (T) = d \text{ et } Dir_k\ (T) = IU_k \text{ et } \forall i \neq k, Etat_i\ (T) = nd \text{ et } Dir_i\ (T) = IU_k.}$$

Avant d'aborder la résolution de ce problème, nous allons constater qu'il est impossible à résoudre dans sa plus grande généralité.

Théorème 2.1 : Soit PROG un réseau de n processus connectés selon un anneau bidirectionnel. Si tous les processus sont complètement identiques, en particulier ne possèdent pas d'identités uniques, alors PROG ne résoud pas le problème de l'élection du dirigeant.

Démonstration : Supposons que PROG résolve le problème et montrons que c'est impossible. Soit donc l'exécution de PROG à partir de (EI) et se terminant en (EF). Par induction sur t, on peut montrer que les variables Etat et Dir de tous les processus sont identiques :

$$\mathbf{\forall t, \forall i,j = 0, \ldots, n-1, Etat_i\ (t) = Etat_j\ (t) \text{ et } Dir_i\ (t) = Dir_j\ (t).}$$

Par conséquent, si un processus est tel que $\mathbf{Etat_k\ (T) = d}$, alors $\mathbf{\forall i \neq k, Etat_i\ (T) = d}$, ce qui viole la condition d'unicité de (EF). %

Nous déduisons qu'il faut introduire une dissymétrie dans les processus. Le plus simple est de demander à ce que les IU soient différentes. Nous montrons dans la suite qu'il suffit alors de pouvoir comparer les UI deux à deux pour résoudre le problème sans aucune autre connaissance sur le réseau.

Le réseau LCR (Le Lann, Chang et Roberts) résoud le problème de l'élection d'un dirigeant sur un anneau en calculant le nœud d'identité maximum : ce nœud sera le dirigeant. Informellement, l'algorithme est le suivant : chaque processus envoie son IU et reçoit celle de son voisin; il compare cette IU à la sienne ; si l'IU reçue est supérieure à la sienne, il l'envoie au processeur suivant ; si elle est inférieure, il ne fait rien ; si elle est égale à la sienne, alors il se déclare dirigeant.

Formellement, chaque processus exécute de manière synchrone l'algorithme LCR suivant :

Algorithme LCR :

$\mathbf{P^i}$ **:** **début**
choisir IU ; /H := 0/
Etat := nd ;
Dir := IU ;
envoyer Dir **à droite** ;

recevoir m **de gauche** ;
répéter : /H := H+1/
si m > Dir **alors** Dir := m ; **envoyer** Dir **à droite** ; **recevoir** m **de gauche** ;
si m < Dir **alors envoyer** Nul **à droite** ; **recevoir** m **de gauche** ;
si m = Dir **alors** Etat := d **; envoyer** Nul **à droite** ; **recevoir** m **de gauche** ;
fin.

Théorème 2.2 : Après H(LCR) = n étapes de communication, l'algorithme LCR résoud le problème de l'élection d'un dirigeant pour n processus connectés par un anneau unidirectionnel avec connaissance réduite aux voisins. Le nombre maximum de messages non nuls échangés est $O(n^2)$.

Démonstration : Appelons $Pred_i(t)$ l'ensemble des processus qui peuvent atteindre le processus P^i en au plus t étapes de communication. On peut définir par récurrence $Pred_i(t)$ par $Pred_i(0) = \{i\}$ et $Pred_i(t) = Pred_{i-1}(t-1) \cup Pred_i(t-1)$. Après t étapes de communication de l'algorithme LCR, on a l'invariant suivant

$$\mathbf{Dir_i(t) = max \{IU_k , \forall k \in Pred_i(t)\}}$$
$$\mathbf{m_i(t) = Dir_i(t) \text{ ou } Nul.}$$

La propriété est vraie pour t=0. Supposons là vraie en t-1. Si à l'étape t, le processus P^i reçoit un message m alors, par construction,

$$\mathbf{m = m_{i-1}(t-1) = Dir_{i-1}(t-1) = max \{IU_k , \forall k \in Pred_{i-1}(t-1)\}}$$

De l'algorithme, on déduit :

$$\mathbf{Dir_i(t) = max\{m, Dir_i(t-1)\} = max\{ Dir_{i-1}(t-1), Dir_i(t-1)\}}$$
$$\mathbf{= max\{ max \{IU_k , \forall k \in Pred_{i-1}(t-1)\}, max \{IU_k , \forall k \in Pred_i(t-1)\}\} = max \{IU_k , \forall k \in Pred_i(t)\}}$$

Si à l'étape t, le processus P^i reçoit un message Nul alors, par construction,

$$\mathbf{Dir_i(t) = Dir_i(t-1) = Dir_{i-1}(t-1) = \{ Dir_{i-1}(t-1), Dir_i(t-1)\}}$$

Ce qui conclut la récurrence.
Plaçons nous maintenant à l'étape n. On a alors :

$$\mathbf{Dir_i(n) = max \{IU_k , \forall k \in Pred_i(n)\} = IU_{max}}$$

Soit imax le numéro du processus possédant l'IU maximum. On a alors

$$\mathbf{Dir_{imax-1}(n-1) = max \{IU_k , \forall k \in Pred_i(n-1)\} = IU_{max}}$$
$$\mathbf{m_{i-1}(n-1) = IU_{max}}$$

A l'étape n, le processus imax reçoit le message $m_{i-1}(n-1) = IU_{max}$ et donc affecte d à sa variable Etat.

Le nombre de message ne peut pas dépasser $O(n^2)$ puisqu'il y a au maximum n étapes de communication. Si les nœuds sont ordonnés par IU décroissante de gauche à droite alors le nombre de messages non nuls est de $O(n^2)$. %

2.6 Election d'un dirigeant sur un réseau synchrone quelconque

Supposons que G soit maintenant un graphe orienté fortement connexe :

$$\mathbf{G = (V,E) ; \forall\ i,j\ \exists \text{ chemin de } V_i \text{ à } V_j}$$

L'élection d'un dirigeant dans un réseau synchrone de processus reprend le principe de l'algorithme LCR en inondant les successeurs avec les identités maximales. Pour chaque V_i on note $Succ_i$ l'ensemble des successeurs et $Pred_i$ l'ensemble des prédécesseurs. Chaque processus envoie l'identité du dirigeant courant (ou nul) et reçoit de chacun de ses prédécesseurs l'identité de leurs dirigeants courants (ou nul), l'ensemble correspondant est noté M. On obtient l'algorithme IM (inondation maximale) :

P^i : début
choisir IU ; /H := 0/

Etat := nd ;
Dir := IU ;
envoyer Dir **à** Succ ;
recevoir M **de** Pred;
répéter : /H := H+1/
m := max M ;
si m > Dir **alors** Dir := m ; **envoyer** Dir **à** Succ ; **recevoir** M **de** Pred;
si m < Dir **alors envoyer** Nul **à** Succ ; **recevoir** M **de** Pred;
si m = Dir **alors** Etat := d **; envoyer** Nul **à** Succ ; **recevoir** M **de** Pred;
fin.

Malheureusement cet algorithme est faux. En effet, il est possible que m = Dir sans être le maximum global. Pour avoir le maximum global, il faut être sûr que l'information soit passée par tous les sommets. Il y a différentes façons de résoudre ce problème, l'une des plus simples étant de répéter la boucle autant de fois que le diamètre du réseau. Rappelons que le diamètre d'un graphe G est :

$$\mathbf{Diam(G) = max_{i,j}\ min\ (\ |\ chemin\ de\ V_i\ à\ V_j\ |\ )}$$

Algorithme IM :

P^i : **début**
choisir IU ; /H := 0/
Etat := nd ;
Dir := IU ;
envoyer Dir **à** Succ ;
recevoir M **de** Pred;
répéter diam(G) **fois:** /H := H+1/
m := max M ;
si m > Dir **alors** Etat := nd **;** Dir := m ; **envoyer** Dir **à** Succ ; **recevoir** M **de** $Pred_i$;
si m < Dir **alors envoyer** Nul **à** $Succ_i$; **recevoir** M **de** $Pred_i$;
si m = Dir **alors** Etat := d **; envoyer** Nul **à** Succ ; **recevoir** M **de** Pred;
fin.

Théorème 2.2 : L'algorithme IM résoud le problème de l'élection d'un dirigeant pour n processus connectés par un réseau fortement connexe quelconque avec connaissance réduite aux voisins. Le nombre maximum de messages non nuls échangés est $O(diam(G).n^2)$.

Démonstration : La démonstration est proche de la précédente. Elle se base sur un invariant : après r tours, le maximum a atteint tous les sommets à distance r. %