13.4 File d'attente répartie

Dans cette section, nous étudions un algorithme d'exclusion mutuelle basé sur le principe des sémaphores. Nous commençons par le cas d'un système centralisé, avant de généraliser l'algorithme à un système complètement réparti. La technique de transformation de l'algorithme centralisé en distribué, basée sur les estampilles (*cf.* chapitre 5) est un principe général qui peut être appliqué à d'autres problèmes que celui de l'exclusion mutuelle (et notamment à la gestion des données réparties).

13.4.1 Solution centralisée

La solution centralisée repose sur l'existence d'un site particulier, appelé *coordonnateur*. Il est connu de tous les autres sites, qui peuvent communiquer directement avec lui. Si le réseau physique n'est pas complet, le système doit donc disposer d'un routage.

Le coordonnateur gère les accès à la ressource, en autorisant ou non les sites qui le sollicitent à rentrer en section critique. Il maintient une liste des demandeurs non encore satisfaits de manière à traiter les demandes dans l'ordre de leurs arrivées. Le traitement d'une demande, l'accord de la ressource ou la mise en attente du site demandeur se fait de façon atomique : une seule demande est traitée à la fois.

Cette solution centralisée correspond en fait à l'adaptation directe de la gestion d'un sémaphore dans un système concurrent. Ici, les actions P et V se traduisent par des communications vers le coordonnateur.

13.4.2 Solution répartie

Cette solution consiste à dupliquer les données de la file d'attente sur chaque site. Les messages sont acquittés avec des accusés de réception. Les multiples copies de la file sont maintenues à jour grâce aux estampilles. Comme précédemment, le réseau est supposé complet, mais on suppose maintenant que les liens de communication sont FIFO.

Exercice 302 : Quel(s) protocole(s) de communication satisfont ces hypothèses ?

Chaque site S_i maintient un tableau Tab_i de N cases (autant que de sites dans le réseau), lui permettant de retenir le dernier message échangé avec chacun des autres sites. Pour entrer en section critique, le site S_i diffuse une requête estampillée à l'ensemble des autres sites, et attend les accusés de réception, eux-mêmes estampillés. Il stocke dans $\text{Tab}_i[i]$ sa requête, ainsi que l'estampille correspondante.

Un site S_j recevant la requête de S_i la stocke dans $\text{Tab}_j[i]$ avec son estampille, et renvoie à S_i un accusé de réception. Puisque les canaux sont FIFO, cet accusé de réception a pour effet de «pousser» les messages sur le canal : lors de la réception de l'accusé de réception de S_j, le site S_i sait qu'il ne reste pas de messages en transit envoyés par S_j avant sa propre requête d'entrée en section critique.

Le site S_i entre effectivement en section critique après avoir diffusé sa requête d'entrée, et uniquement lorsque l'estampille de la case $\text{Tab}_i[i]$ est la plus petite de toutes celles du tableau. Si ce n'est pas le cas, il attend que cela le devienne. Cela ne pourra se produire que lorsqu'il aura reçu des messages de chacun des autres sites postérieurs à sa propre requête. Ces messages peuvent être les accusés de réception attendus. Mais ils peuvent également être des requêtes concurrentes. Si l'une de ces requêtes est plus ancienne (au sens de la causalité), alors S_i doit attendre. Si la requête de S_i est la plus ancienne, alors il peut entrer en section critique. Afin de ne pas perdre l'information concernant

une requête en attente et ainsi de gérer les requêtes concurrentes, la réception par S_i de l'acquittement de S_j et de son estampille ne peut écraser une requête plus ancienne de S_j dans le tableau de S_i.

Dans ce cas, pour tout site S_j, la case $\text{Tab}_j[i]$ contient la plus petite estampille du tableau Tab_j, et cela assure que S_i sera le seul site à entrer en section critique. Lorsque S_i sort de la section critique, il diffuse un message de sortie estampillé, qu'il stocke lui-même dans la case $\text{Tab}_i[i]$. L'arrivée d'un tel message sur un site S_j, forcément d'estampille supérieure à celle d'une requête survenue après celle de S_i mais avant sa sortie de section critique, permet d'écraser l'estampille dans la case $\text{Tab}_j[i]$, et ainsi de débloquer le site ayant fait la requête d'entrée en section critique la plus ancienne après S_i.

On obtient l'algorithme ci-dessous, écrit sous la forme d'un algorithme de contrôle, qui intercepte les messages d'une application de base (quelconque) souhaitant entrer en section critique pour une ressource (quelconque).

Algorithme 28 : **Répartition d'une file d'attente, site** S_i

Initialisation :
  $\text{Tab}_i[k] \leftarrow$ (libération, 0) pour tout $k \in \{1, \ldots, N\}$
  ▷ *Tab$_i$ est un tableau de N couples $(type, date)$, où type prend les valeurs* requête*,* libération *et* accusé*, et où date est un entier (date logique).*
  $h_i \leftarrow 0$

Réception d'une demande de section critique de l'application de base :
  recevoir([demandeSC] **)** de l'application de base
  $h_i \leftarrow h_i + 1$
  $\text{Tab}_i[i] \leftarrow$ (requête, h_i)
  envoyer([requête] h_i **)** à tous les autres sites

Réception fin de section critique de l'application de base :
  recevoir([finSC] **)** de l'application de base
  $h_i \leftarrow h_i + 1$
  $\text{Tab}_i[i] \leftarrow$ (libération, h_i)
  envoyer([libération] h_i **)** à tous les autres sites.

Réception d'un message de type requête :
  recevoir([requête] h **)** de S_j
  $h_i \leftarrow \max(h_i, h) + 1$
  $\text{Tab}_i[j] \leftarrow$ (requête, h)
  envoyer([accusé] h_i **)** à S_j
  ▷ *L'arrivée du message pourrait permettre de satisfaire une éventuelle demande de S_i.*
  si $\text{Tab}_i[i]$.type == requête **et** $(\text{Tab}_i[i].\text{date}, i) <_2 (\text{Tab}_i[k].\text{date}, k)$ pour tout $k \neq i$ **alors**
    ▷ *S_i est demandeur et sa requête est la plus ancienne.*
    envoyer([débutSC] **)** à l'application de base
  fin si

Réception d'un message de type libération :
  recevoir([libération] h **)** de S_j
  $h_i \leftarrow \max(h_i, h) + 1$
  $\text{Tab}_i[j] \leftarrow$ (libération, h)

▷ *L'arrivée du message pourrait permettre de satisfaire une éventuelle requête de S_i.*
si Tab$_i[i]$.type == requête **et** $(\text{Tab}_i[i].\text{date}, i) <_2 (\text{Tab}_i[k].\text{date}, k)$ pour tout $k \neq i$ **alors**
▷ *S_i est demandeur et sa requête est la plus ancienne.*
envoyer([débutSC] **)** à l'application de base
fin si

Réception d'un message de type accusé :
recevoir([accusé] h **)** de S_j
$h_i \leftarrow \max(h_i, h) + 1$
si Tab$_i[j]$.type $\neq$ requête **alors**
▷ *On n'écrase pas la date d'une requête par celle d'un accusé.*
Tab$_i[j] \leftarrow$ (accusé, h)
fin si
▷ *L'arrivée du message pourrait permettre de satisfaire une éventuelle demande de S_i.*
si Tab$_i[i]$.type == requête **et** $(\text{Tab}_i[i].\text{date}, i) <_2 (\text{Tab}_i[k].\text{date}, k)$ pour tout $k \neq i$ **alors**
▷ *S_i est demandeur et sa requête est la plus ancienne.*
envoyer([débutSC] **)** à l'application de base
fin si

Le principe mis en œuvre ici peut être appliqué pour maintenir la cohérence de données réparties quelconques, et ainsi décentraliser des algorithmes qui ont été conçus de façon centralisée. Néanmoins la complexité en messages est assez mauvaise. En outre, il est nécessaire de connaître le nombre de sites.

Exercice 303 : Combien de messages doivent être émis au minimum avant qu'un site S_i puisse entrer en section critique ?