CHAPITRE IV

LA CONCURRENCE

version du 1er décembre 2003

1 Problématique

La concurrence des processus lors de l'accès à des ressources partagées peut provoquer l'incohérence de celles-ci. Gérer la concurrence revient à contrôler que des accès concurrents s'exécutent de manière cohérente. La manière la plus simple de parvenir à cette cohérence consiste à garantir que pour chaque ressource au plus une section de code qui la manipule soit en cours d'exécution. Une telle section de code est appelée *section critique*. Bien que d'autres manières de gérer la concurrence soient possibles (e.g. la séquentialisation des transactions de bases de données), nous nous limiterons dans ce chapitre à étudier la mise en oeuvre de l'exclusion mutuelle entre sections critiques.

De manière plus précise, lorsqu'un processus applicatif désirera exécuter une section critique, son programme se présentera comme suit :

```
Prologue(); Section critique; Epilogue()
```

`Prologue` et `Epilogue` sont des primitives du service qui devront garantir les deux propriétés caractéristiques de l'exclusion mutuelle :

- A tout instant, au plus un processus est en cours d'exécution d'une section critique. Ce type de propriété est appelée propriété de *sûreté* et traduit le fait que "quelque chose de mal n'arrivera jamais".
- Tout processus demandant à exécuter une section critique (par appel à `Prologue`), pourra l'exécuter (par retour de `Prologue`) au bout d'un temps fini. Ce type de propriété est appelée propriété de *vivacité* et traduit le fait que "quelque chose de bien finira par arriver".

De nombreux algorithmes ont été proposés dans la littérature. Dans un souci pédagogique nous présenterons trois d'entre eux (parmi les plus anciens). Ces algorithmes sont tous basés sur les horloges logiques présentées au chapitre précédent. Chaque nouvel algorithme se construit par une modification conceptuelle du précédent et diminue la complexité (mesurée en nombre de messages échangés par exécution d'une section critique).

Avant d'étudier ces algorithmes, examinons pourquoi la technique de circulation d'un jeton sur un anneau pour assurer l'exclusion mutuelle n'est pas satisfaisante. Nous mesurons la complexité en nombre de messages échangés pour une section critique. Or il apparaît que même si les sites ne sont pas désireux d'exécuter une section critique, le jeton doit circuler. Ce qui signifie que le nombre de messages échangés peut s'accroître indéfiniment sans une seule entrée en section critique !

2 Algorithme de Lamport [Lam78]

Comme les deux algorithmes suivants, cet algorithme repose sur le mécanisme des horloges logiques. Aussi pour éviter l'écriture d'un code répétitif, nous supposerons que **la mise à jour de l'horloge logique est faite dans la couche réseau** à l'émission et à la réception. Ceci impose cependant à la couche service de fournir une valeur d'horloge dans tous les messages de l'algorithme (le lecteur pourra vérifier que cette contrainte est respectée).

2.1 Principe et réalisation de l'algorithme

Le protocole échange trois types de message :

- des requêtes d'exécution de section critique diffusées à tous les autres sites par le site demandeur,
- des acquittements de requête pour indiquer que le site a "enregistré" la requête,
- des libérations diffusées à tous les autres sites par un site qui a terminé l'exécution d'une section critique.

Sans entrer maintenant dans les détails de l'algorithme et en posant `n` le nombre de sites, on voit immédiatement qu'une section critique s'accompagne de `3.(n-1)` messages : `(n-1)` requêtes, `(n-1)` acquittements et `(n-1)`libérations.

Le point clef de l'algorithme est la condition d'entrée en section critique. Pour ce faire, chaque site maintient un tableau de messages indicé par les identités des sites. Chaque cellule de ce tableau contient le type du message et son heure. Lorsqu'un site désire exécuter sa section critique, **il met à jour sa propre cellule avec sa requête**. La condition d'entrée en section critique est alors **d'avoir dans sa cellule le message le plus âgé du tableau**. Rappelons qu'au sens des horloges logiques, la "date de naissance" d'un message est constitué du doublet (horloge, identité du site) ce qui évite le cas d'égalité des âges de deux messages.

Toutes les cellules sont initialisées avec un message de libération daté de l'heure `–1` de telle sorte qu'une requête initiale doit donner lieu à des acquittements pour exécuter la section critique. Il reste à préciser la règle de mise à jour des autres cellules du tableau. On pourrait penser que tout message reçu de j provoque la mise à jour de la cellule j du tableau mais l'exemple introductif décrit ci-dessous (figure 4.1) montre que cette règle doit être modulée.

Dans ce scénario, les sites `1` et `2` désirent entrer en section critique. Ils diffusent donc leur requête et mettent respectivement à jour la cellule de leur tableau. Lorsque le site `2` reçoit l'acquittement du site `3` puis la requête du site `1`, il met à jour « leur » cellule de son tableau. Il ne peut entrer en section critique car la requête du site `1` est plus âgée - `(0,1)<(0,2)` - . Lorsque le site `1` reçoit la requête du site `2`, il met à jour la cellule correspondante de son tableau. Il ne peut entrer en section critique car le message de libération du site `3` est plus âgé - `(-1,3)<(0,1)` - . Que doivent faire les sites `1` et `2` lorsqu'ils reçoivent l'acquittement de l'autre site ? S'ils mettent à jour leur tableau, le site `2` rentre immédiatement en section critique, tandis que le site `1` entrera en section critique à la réception de l'acquittement du site `3`. Il n'y aura donc pas exclusion mutuelle entre les sections critiques. Le problème vient du fait que la mise à jour du tableau par l'acquittement provoque l'oubli de la requête qui pourtant n'est pas encore traitée.

Nous sommes maintenant en mesure de définir la règle de mise à jour du tableau lors d'une réception. Tout message arrivant est enregistré dans le tableau **à l'exception d'un acquittement qui remplacerait une requête**.

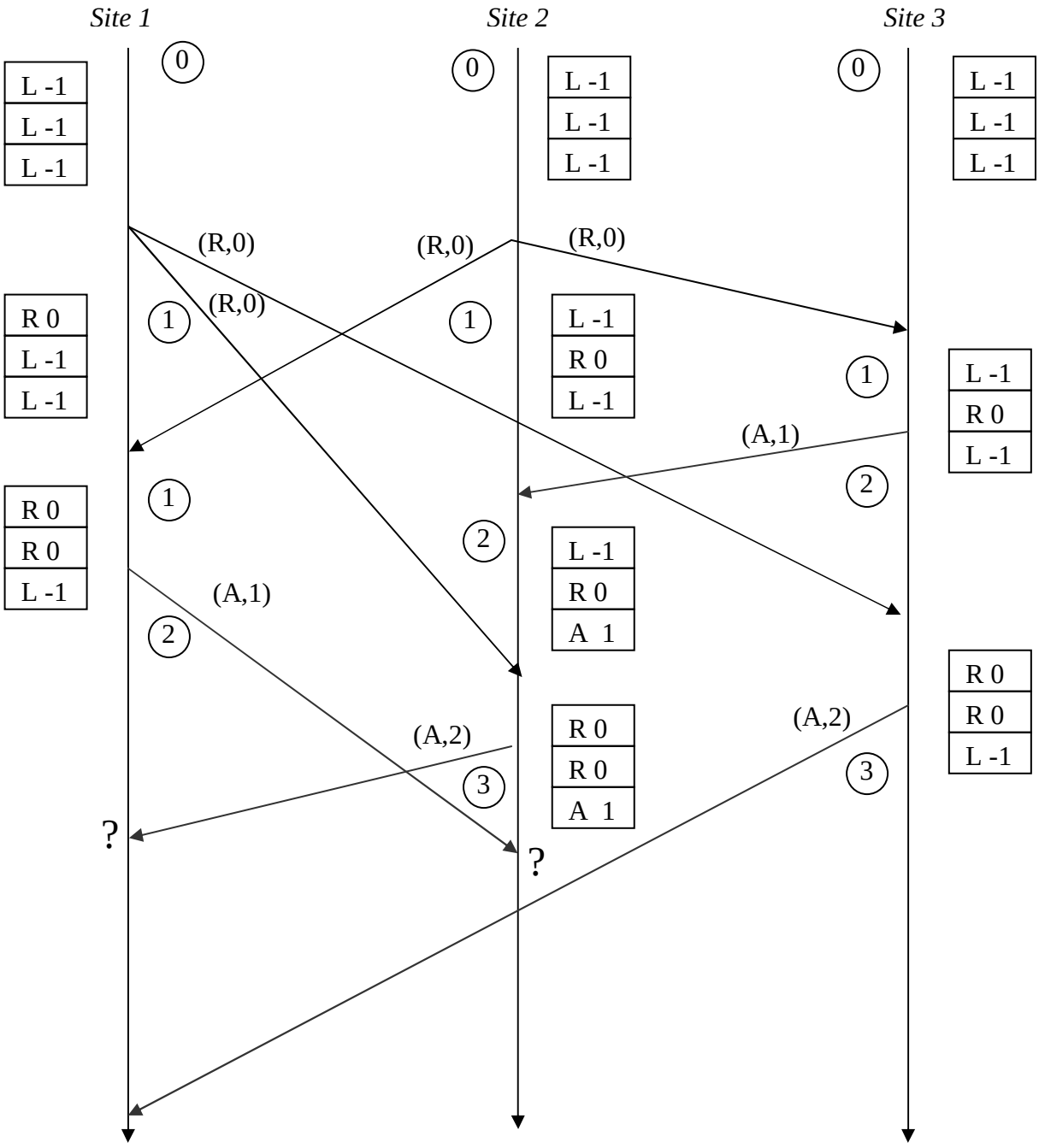

Figure 4.1 : Un scénario d'entrée en section critique (début)

La suite du scénario est présenté à la figure 4.2. Le site `1` entre en section critique à la réception de l'acquittement du site `3`. Lorsque l'application du site `1` termine l'exécution de sa section critique et appelle `Epilogue()`, la diffusion des messages de libération provoque à la réception par le site `2`, son entrée en section critique. On remarque que la diffusion des messages de libération par le site `2` est sans influence sur le comportement des autres sites. Nous reviendrons sur ce point au prochain algorithme.

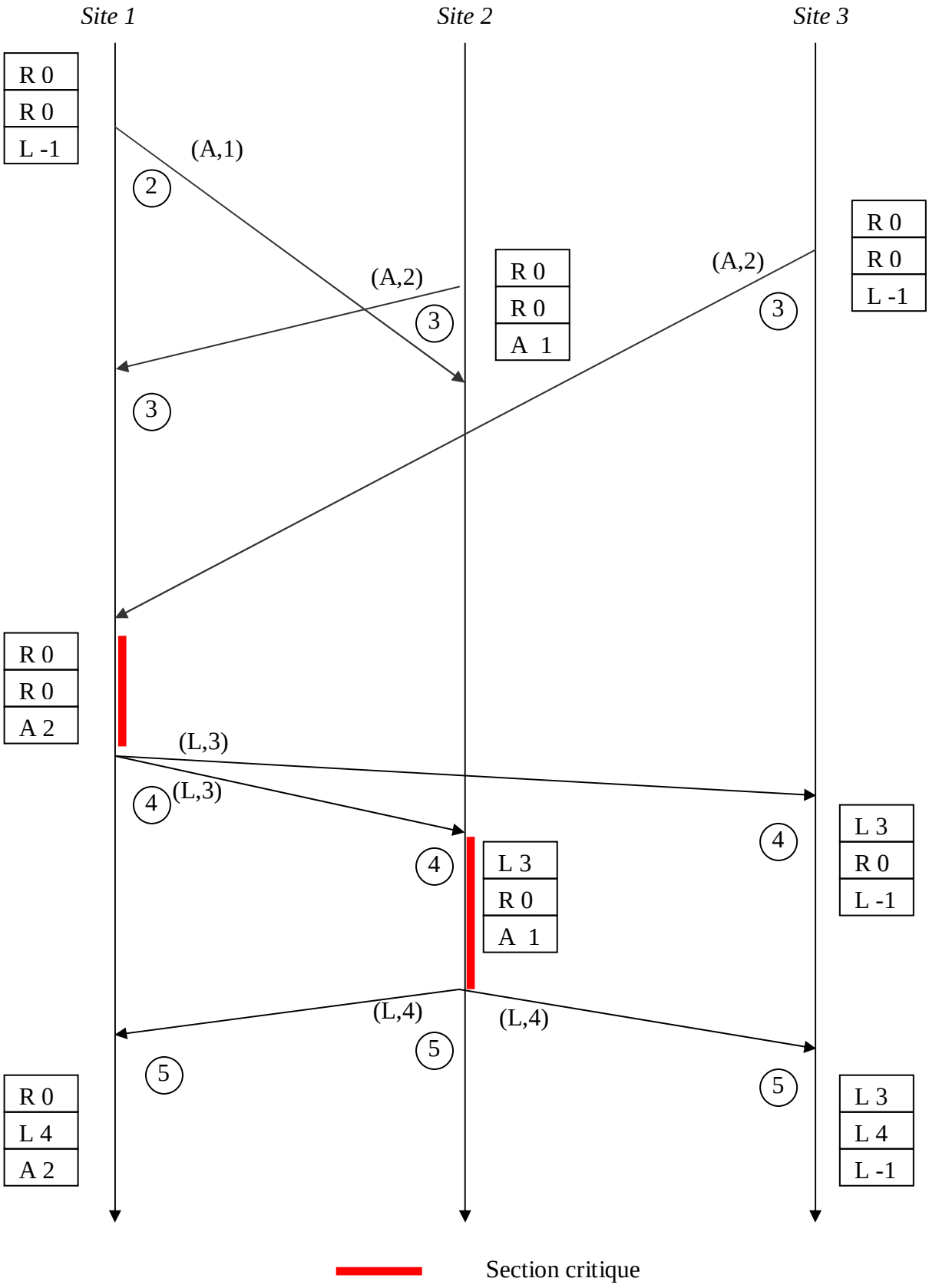

Figure 4.2 : Un scénario d'entrée en section critique (suite)

Variables du site `i`

- h_i : variable contenant la valeur de l'horloge locale du site i. Elle est initialisée à `0`.
- $Mess_i[1..N]$: tableau des messages. Chaque cellule de ce tableau est composée des deux champs `<type,heure>`. `type` prend ses valeurs parmi {`req`, `acq`, `lib`} et est initialisé à `lib`. `heure` est initialisé à `-1`.

Algorithme du site `i`

Avant d'entrer en section critique, un site enregistre sa requête courante, la diffuse et attend que la requête soit le message le plus âgé du tableau.

```
prologue()
Début
     Mess_i[i].heure = h_i;
     Mess_i[i].type = req;
     Diffuser(req,h_i); // h_i est incrémenté (par la couche réseau)
     Attendre(∀j≠i,(Mess_i[i].heure,i)<(Mess_i[j].heure,j);
Fin
```

Lorsqu'un site quitte la section critique, il diffuse un message de libération à tous les autres sites.

```
épilogue()
Début
     Diffuser(lib,h_i);
Fin
```

La réception d'une requête provoque l'envoi d'un accusé de réception. Le message est enregistré dans le tableau excepté dans le cas discuté plus haut.

```
sur_réception_de(j,(tp,h)) // h_i=max(h_i,h+1) (par la couche réseau)
Début
     Si tp == req Alors
          envoyer_à(j,(acq,h_i));
     Finsi
     Si (tp != acq) || (Mess_i[j].type != req) Alors
          Mess_i[j].heure = h;
          Mess_i[j].type = tp;
     Finsi
Fin
```

2.2 Preuve de l'algorithme

A titre d'exemple, nous allons établir la propriété de vivacité et de sûreté de l'algorithme de Lamport. Nous invitons le lecteur à établir des preuves similaires pour les deux autres algorithmes.

2.2.1 Propriété de vivacité

Nous raisonnons par l'absurde : supposons qu'il existe une exécution (infinie) au cours de laquelle un ensemble de sites E_0 reste indéfiniment bloqué au cours d'un appel à `prologue()`. Les autres sites peuvent être partitionnés en deux catégories : E_1 l'ensemble des sites qui n'accèdent qu'un nombre fini de fois à une section critique et E_2 l'ensemble des sites qui accèdent un nombre infini de fois à une section critique.

Nous nous nous plaçons en un instant t_0 où :

1. Tous les messages des requêtes associées aux appels à `prologue()` non terminés des sites de E_0 ont été reçus par les autres sites et donc enregistrés dans leur tableau. Ces messages restent indéfiniment dans les tableaux.
2. Tous les acquittements associés à ces requêtes ont été reçus.
3. Tous les messages de libération associés aux derniers appels à `épilogue()`des sites de E_1 ont été reçus par les autres sites et donc enregistrés dans leur tableau.

Montrons d'abord que E_2 est vide. En effet, supposons qu'il existe un site $i \in E_2$, alors (par définition de E_2) l'application de `i` appelle `prologue()` après l'instant t_0. D'après le mécanisme des horloges logiques, cette requête a nécessairement une heure plus grande que celle d'une requête non traitée d'un quelconque site de E_0. D'après le point 1, i reste bloqué indéfiniment. Ce qui est contradictoire avec la définition de E_2.

Soit maintenant $i \in E_0$ et $j \in E_1$. Examinons le contenu de la cellule $Mess_i[j]$ après l'instant t_0. L'acquittement par `j` de la requête non traitée de `i` a été reçu.

- S'il a été enregistré, il ne peut bloquer le site `i` puisque son heure est plus grande que celle de la requête. Les messages, qui éventuellement lui succèdent dans cette cellule, ayant des heures plus grandes ne peuvent non plus bloquer le site `i`.
- S'il n'a pas été enregistré, cela signifie qu'une requête était présente dans $Mess_i[j]$. Dans ce cas, (par définition de E_1) le site `i` recevra au pire à l'instant t_0 le message de libération correspondant qui ne pourra bloquer le site `i` puisque son heure est plus grande que celle de l'acquittement donc de la requête non traitée de `i`. Les messages, qui éventuellement lui succèdent dans cette cellule, ayant des heures plus grandes ne peuvent non plus bloquer le site `i`.

Donc un site de E_1 ne peut bloquer un site de E_0 à partir de t_0.

Soit maintenant le site $i \in E_0$ dont la requête non traitée est la plus âgée. A l'instant t_0, il ne peut être bloqué par un message des autres sites de E_0 ni par les messages des sites de E_1. En conséquence son appel à `prologue()` devrait se terminer. D'où la contradiction.

2.3.2 Propriété de sûreté

Nous raisonnons par l'absurde : supposons que deux sites soient à un même instant en cours d'exécution d'une section critique.

Appelons i_1 le site dont la requête correspondante est la plus « jeune » et i_2 l'autre site. Le site i_1 diffuse la requête correspondante r_1 à l'heure logique h_1 et le site i_2 diffuse la sienne (r_2) à l'heure logique h_2. Appelons m le message émis par le site i_2 à destination du site i_1, présent dans le tableau au moment où i_1 pénètre en section critique.

Remarquons tout d'abord que r_2 ne précède pas m car au moment où i_1 pénètre en section critique, i_2 n'a pas terminé la section critique correspondant à r_2. Il n'a donc pas envoyé de message de libération et les messages d'acquittement ne peuvent remplacer r_2 dans le tableau de i_1.

D'autre part r_2 et m ne sont pas confondus car la requête du site i_2 étant plus âgée, elle empêcherait i_1 de pénétrer en section critique.

Le message m précède donc r_2 (cf. figure 4.3). Soit h l'heure logique de m. Puisque i_1 pénètre en section critique $(h_1,i_1) < (h,i_2)$. Puisque m précède r_2, $(h,i_2) < (h_2,i_2)$. Par transitivité, $(h_1,i_1) < (h_2,i_2)$. Ce qui est contradictoire avec nos hypothèses.

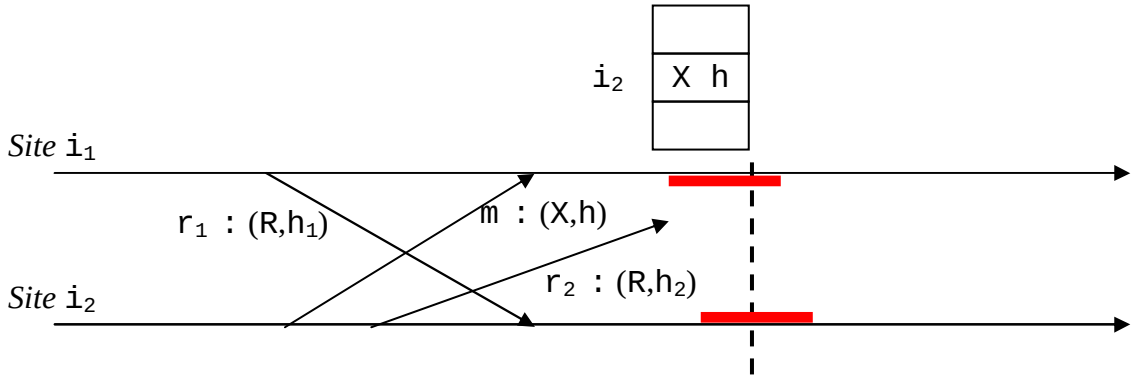

Figure 4.3 : Deux sites simultanément en section critique

3 Algorithme de Ricart et Agrawala [Ric81]

3.1 Principe de l'algorithme

Dans l'algorithme de Lamport, un acquittement s'interprète comme le fait d'enregistrer la requête. Le principal apport de ce nouvel algorithme consiste à modifier la sémantique de l'acquittement.

A présent, un acquittement du site `j` à une requête du site `i` signifie que `j` autorise `i` à pénétrer en section critique. La condition d'entrée en section en section critique devient maintenant **la réception d'un acquittement venant de chacun des autres sites**. Comme chaque site n'envoie qu'un seul acquittement relatif à une requête, il suffit de compter le nombre d'acquittements reçus.

Il reste maintenant à définir la politique du site `j` lorsqu'il reçoit cette requête. Si son application n'est pas en attente ou en cours d'exécution d'une section critique, l'acquittement sera délivré immédiatement. Dans le cas contraire, `j` compare l'âge de sa requête avec celui de la requête de `i`. Si cette dernière est plus âgée, l'acquittement est délivré immédiatement. Dans le cas contraire, `j` diffère l'acquittement jusqu'à la fin de sa section critique. Les messages de libération de l'algorithme de Lamport s'interprètent alors comme des acquittements différés. Bien entendu, il s'agit pour un site de mémoriser, les sites qu'il a retardés pour réaliser cet envoi différé.

Il est immédiat que pour une entrée en section critique, `2.(n-1)` messages sont échangés : `(n-1)` requêtes et `(n-1)` acquittements. La complexité de cet algorithme est donc meilleure que celle de l'algorithme précédent.

Dans le scénario de la figure 4.4, les sites `1` et `2` désirent entrer en section critique. Ils diffusent donc leur requête et modifient leur état, mémorisent l'heure de leur requête et attendent les acquittements. Le site `3` acquitte les deux requêtes car son application n'est pas intéressée (au moment de la réception des requêtes) par une section critique. Lorsque le site `1` reçoit la requête du site `2`, il retarde sa réponse car sa propre requête est plus âgée. Par contre le site `2` lui renvoie immédiatement un acquittement. A la réception de l'acquittement du site `3`, le site `1` entre en section critique. Lorsque cette section critique est terminée, le site `1` envoie un acquittement aux sites qu'il a retardés (dans le cas présent au site `2`). A la réception de cet acquittement, le site `2` pénètre à son tour en section critique.

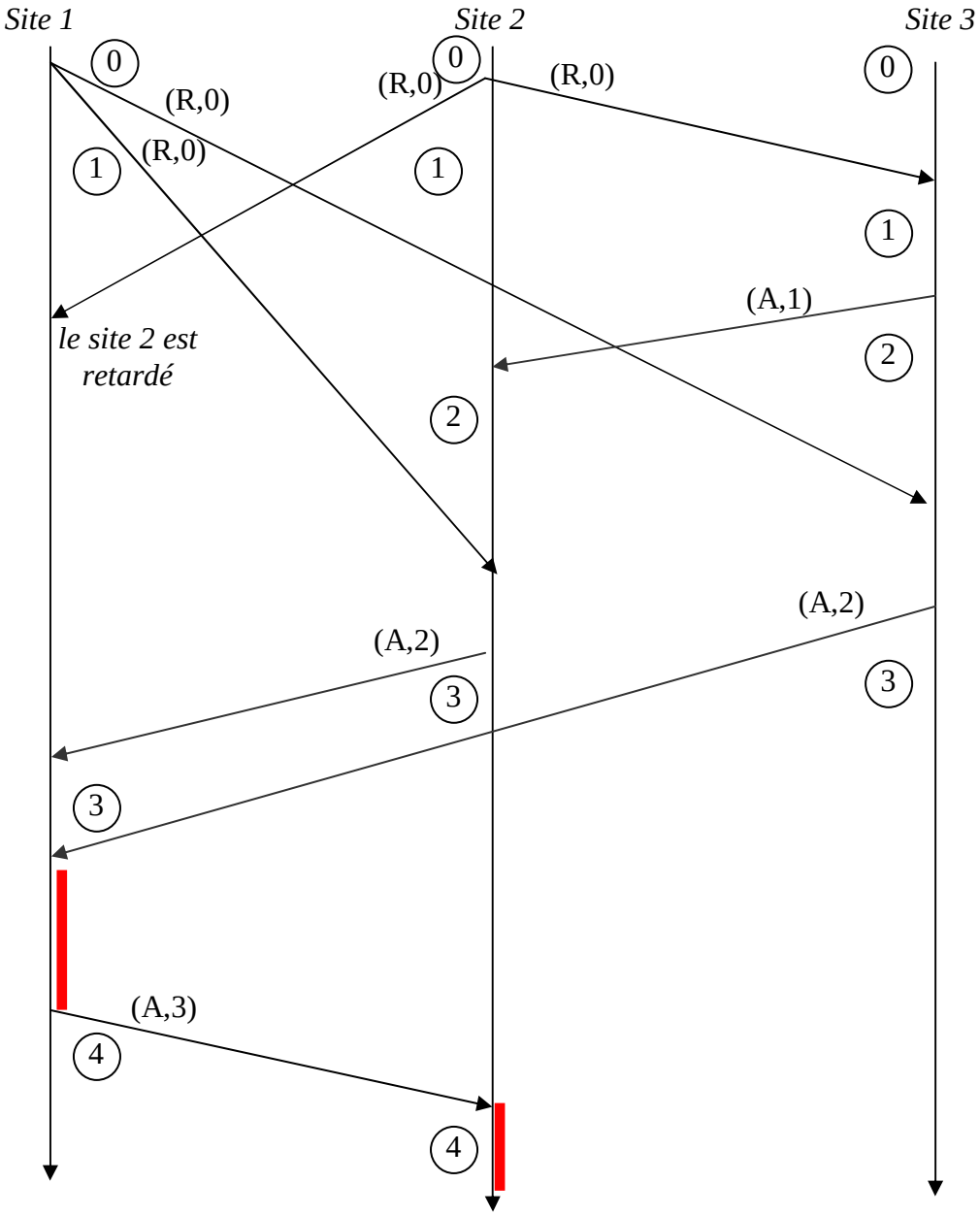

Figure 4.4 : Un scénario d'entrée en section critique

3.2 Description

Variables du site `i`

- h_i : valeur de l'horloge logique. Elle est initialisée à `0`.
- $état_i$: état du site. Cette variable peut prendre l'une des deux valeurs {`repos`, `en_cours`}. Initialement, tous les sites sont dans l'état `repos`.
- $hreq_i$: heure de la requête courante `i`
- $nbacq_i$: compteur des acquittements reçus.
- $Retardé_i[1..N]$: tableau de booléens. $Retardé_i[j]$ est vrai lorsque `i` retarde l'acquittement d'une requête `j`.
- $temp_i$: variable temporaire.

Algorithme du site i

Lorsqu'un site désire entrer en section critique, il diffuse une requête, puis se met en attente de tous les acquittements.

```
prologue()
Début
     hreq_i = h_i;
     nbacq_i = 0;
     Pour temp_i de 1 à n faire // n est le nombre de sites
          Retardé_i[temp_i] = Faux;
     Finpour
     état_i = en_cours;
     Diffuser(req,h_i);
     Attendre(nbacq_i == (n-1));
Fin
```

A la sortie de la section critique, les site envoie des acquittements aux sites qu'il a retardés.

```
épilogue()
Début
     Pour tout temp_i de 1 à n faire
          Si Retardé_i[temp_i] Alors
               envoyer_à(temp_i,(acq,h_i));
          Fin si
     Finpour
état_i = repos;
Fin
```

La réception de la requête suit la politique décrite plus haut.

```
sur_réception_de(j,(req,h))
Début
     Si (état_i == en_cours)&&((hreq_i,i)<(h,j)) Alors
          Retardé_i[j] = Vrai;
     Sinon
          envoyer_à(j,(acq,h_i));
     Finsi
Fin
```

La réception d'un acquittement incrémente le compteur des acquittements reçus.

```
sur_réception_de(j,(acq,h))
Début
     nbacq_i++;
Fin
```

4 Algorithme de Carvalho et Roucairol [Car83]

4.1 Principe de l'algorithme

Dans l'algorithme de Ricart et Agrawala, un acquittement s'interprète comme le fait d'autoriser l'entrée en section critique. Ici aussi l'apport de ce nouvel algorithme consiste à modifier la sémantique de l'acquittement.

A présent, un acquittement du site `j` à une requête du site `i` signifie que `j` autorise `i` à pénétrer en section critique pour la section critique courante **mais aussi pour les sections critiques suivantes**. L'acquittement s'interprète alors comme l'envoi d'une permission partagée entre `i` et `j`. Si `j` veut par la suite pénétrer en section critique, il doit réclamer la permission qu'il a concédée à `i`. La condition d'entrée en section critique devient maintenant **la possession des permissions partagées avec chacun des autres sites**. On remarque que les permissions sont similaires aux jetons multiples associés à l'algorithme du rendez-vous (vu au chapitre 2).

Pour aboutir au développement de l'algorithme, il faut cependant résoudre quelques problèmes liés à cette idée. Tout d'abord il s'agit de répartir initialement les jetons entre les sites. Une idée simple à réaliser consiste à ce que le jeton du couple de sites `{i,j}` soit possédé par le site d'identité `max(i,j).`

Une première modification importante est que lors du `prologue()` seules les permissions manquantes seront réclamées. Ce qui signifie qu'un site peut **entrer immédiatement en section critique sans échanger un seul message** !

Illustrons maintenant les spécificités de cet algorithme par deux scénarios d'exécution. Examinons la figure 4.5. Les sites `1` et `2` désirent exécuter une section critique. Il manque au site `2` le jeton `(2,3)` qu'il réclame au site `3`. Il manque au site `1`, les jetons `(1,3)` et `(2,3)` qu'il réclame aux sites correspondants. Notons que la requête du site `1` est plus âgée que celle du site `2`. Le site `2` ayant obtenu le jeton qui lui manquait entre en section critique. En cours de section critique, il reçoit la requête du site `1`. Bien que celle-ci soit plus âgée que sa propre requête, il ne peut donner le jeton `(1,2)` car l'exclusion mutuelle ne serait plus garantie (la preuve de la correction de l'algorithme de Lamport implique que ceci ne peut arriver dans les algorithmes précédents). Il faut donc que l'algorithme distingue trois états pour mettre en oeuvre sa politique de choix. Soit l'application n'est pas intéressée par une section critique (`repos`), soit elle attend dans le prologue (`attente`), soit elle exécute la section critique (`en_SC`).

La figure 4.6 est une variation du scénario précédent dans lequel la requête du site `1` arrive au site avant l'entrée en section critique. Dans ce cas, la priorité donnée à la requête la plus âgée s'applique et le site `2` donne son jeton au site `1`. Il doit cependant le lui réclamer aussitôt car il ne pourrait entrer en section critique. Il est évident que sa requête sera retardée (excepté si entre l'arrivée des deux messages le site `1` a eu le temps d'exécuter sa section critique) mais l'important est ici de récupérer le jeton perdu. Attention, toutes les requêtes sont datées de la même heure qu'elles soient émises immédiatement, ou sur une perte ultérieure du jeton. Ceci permet d'éviter la famine d'un site.

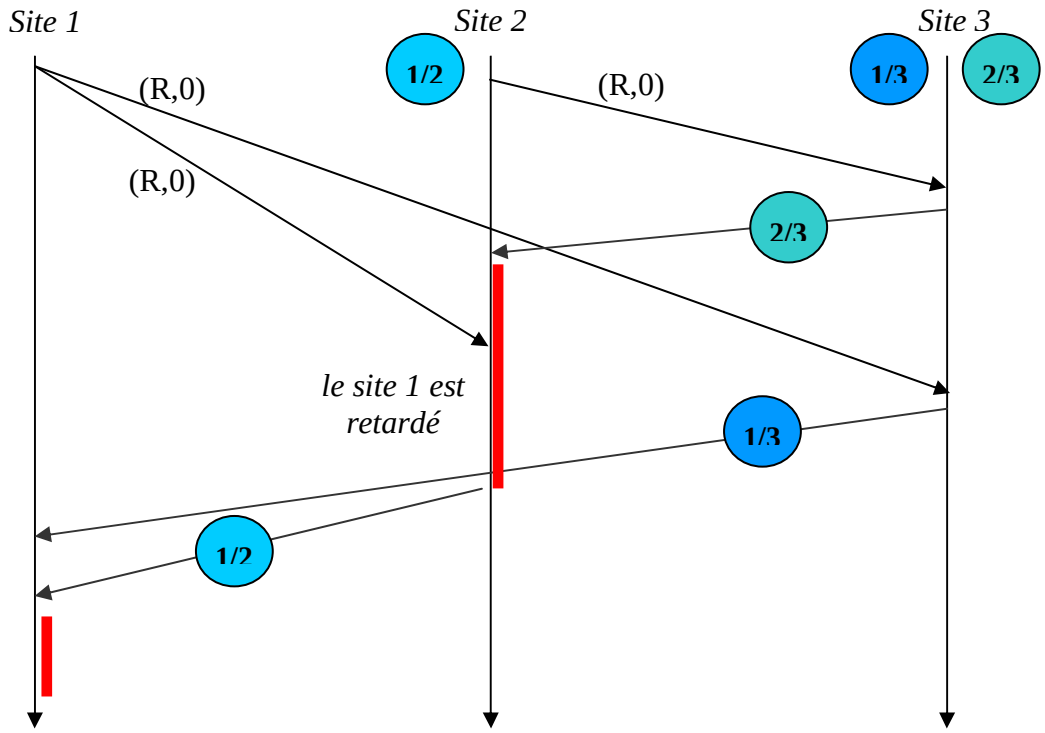

Figure 4.5 : Une requête retardée par une requête plus jeune

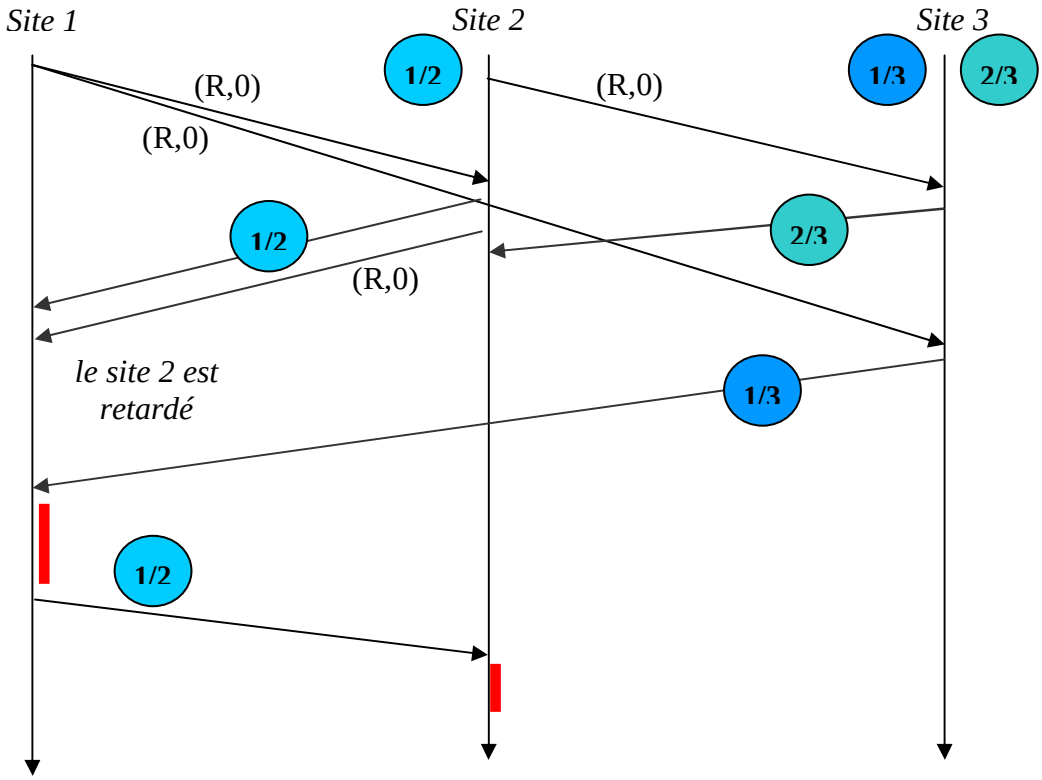

Figure 4.6 : Un site qui donne un jeton et le réclame aussitôt

4.2 Description

Variables du site `i`

- h_i : valeur de l'horloge logique. Elle est initialisée à `0`.
- $état_i$: état du site. Cette variable peut prendre l'une des deux valeurs {`repos`, `attente`, `en_SC`}. Initialement, tous les sites sont dans l'état `repos`.
- $hreq_i$: heure de la requête courante `i`
- $Jeton_i[1..N]$: tableau de booléens indiquant la présence des jetons. $Jeton_i[j]$ est initialisé à la valeur $(i \geq j)$.
- $Retardé_i[1..N]$: tableau de booléens. $Retardé_i[j]$ est vrai lorsque `i` retarde l'acquittement d'une requête `j`.
- $temp_i$: variable temporaire.

Algorithme du site `i`

Le site réclame les jetons qui lui manquent et attend de posséder tous les jetons pour entrer en section critique.

```
prologue()
Début
     hreq_i = h_i;
     Pour temp_i de 1 à n faire // n est le nombre de sites
          Retardé_i[temp_i] = Faux ;
          Si Jeton_i[temp_i] == Faux Alors
               envoyer_à(j,(req,hreq_i));
          Finsi
     Finpour
     état_i = attente;
     Attendre(∀ j,  Jeton_i[j] == Vrai);
     état_i = en_SC;
Fin
```

A la sortie de la section critique, le site envoie les jetons aux sites qu'il a retardés.

```
épilogue()
Début
     Pour temp_i de 1 à n faire
          Si Retardé_i[j] Alors
               envoyer_à(j,(acq,h_i));
               Jeton_i[j] = Faux;
          Finsi
     Finpour
     état_i = repos;
Fin
```

Lors de la réception par le site `i` d'une requête émise par le site `j`, la décision prise par `i` dépend de son état :

- Si le site `i` n'est pas intéressé par une section critique, il envoie le jeton à `j`.
- Si le site `i` exécute sa section critique ou si sa requête est plus âgée que la requête de `j`, il retarde l'envoi du jeton jusqu'à la fin de sa section critique.
- Sinon il envoie le jeton à `j` et le lui réclame aussitôt.

```
sur_réception_de(j,(req,h))
Début
     Si état_i == repos Alors
          envoyer_à(j,(acq,h_i));
          Jeton_i[j] = Faux;
     Sinon si  (état_i==en_SC)||((hreq,i)<(h,j)) Alors
          Retardé_i[j] = Vrai;
     Sinon
          envoyer_à(j,(acq,h_i));
          Jeton_i[j] = Faux;
          envoyer_à(j,(req,hreq_i));
     Finsi
Fin
```

Un jeton arrive ...

```
sur_réception_de(j,(acq,h))
Début
     Jeton_i[j] = Vrai;
Fin
```

4.3 Complexité de l'algorithme

Si un site possède tous ses jetons, il entre immédiatement en section critique. Dans le meilleur des cas, il n'y a pas de messages échangés pour entrer en section critique. Démontrons maintenant qu'un site ne réclame un jeton particulier qu'**au plus une fois**.

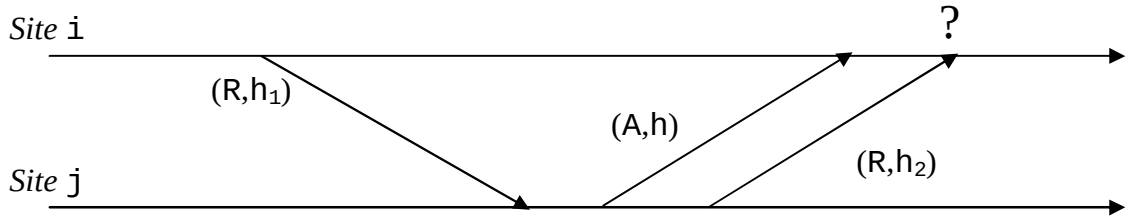

Figure 4.7 : Un jeton récupéré et réclamé une nouvelle fois

Sur la figure 4.7, le site `i` a récupéré le jeton `(i,j)` et le site `j` le lui réclame avant que `i` ne soit entré en section critique. Deux cas se présentent :

- Soit l'émission de la requête correspond à l'appel à `prologue()` et donc $h_2>h>h_1$ auquel cas le site `j` sera retardé.
- Soit l'émission de la requête correspond à la perte du jeton lors de la réception de la requête de `i` et donc (voir la réception d'une requête) $(h_2,j)>(h_1,i)$ auquel cas le site `j` sera ici aussi retardé.

Par conséquent, le nombre de messages nécessaire à une section critique varie entre `0` et `2.(n-1)`puisqu'il y a au plus `n-1` requêtes et donc `n-1` acquittements.

En fonctionnement normal durant une période relativement longue, un petit nombre `p` parmi `n` de sites est actif (comportement sporadique du réseau local). Passée la première entrée en section critique pour chacun des `p` sites, les jetons échangés ne le seront qu'entre ces sites conduisant à une complexité de `2.(p-1)<< 2.(n-1)`. Autrement dit, cet algorithme est très efficace dans la pratique même si sa complexité au pire des cas n'est pas meilleure que celle de l'algorithme précédent.

5 Exercices

Sujet 1
On se propose de définir un nouvel algorithme d'accès à une section critique n'utilisant pas les horloges logiques. Cet algorithme est une adaptation de l'algorithme de Carvahlo et Roucairol. Deux sites quelconques partagent une permission. A la différence de l'algorithme précedent, chaque site tient à jour pour une permission qu'il possède le fait qu'il est prioritaire ou non pour cette permission. Initialement la permission `(i,j)` est possédée par le site d'identité `max(i,j)` et ce site est prioritaire pour l'utilisation de cette permission.

Pour entrer en section critique un site doit posséder toutes les permissions dont il est copropriétaire : il demande donc celles qui lui manquent.

Lorsqu'un site `i` reçoit une requête de `j` , il réagit selon les règles suivantes :
• `i` est au repos,
    il renvoie immédiatement la permission
• `i` est en attente de section critique,
    si `i` a la permission et qu'il n'est pas prioritaire pour cette permission
        il renvoie immédiatement la permission
        il la réclame aussitôt
    sinon
        il diffère sa réponse
• `i` est en section critique,
    il diffère sa réponse

Lorsqu'un site sort de sa section critique, il devient non prioritaire pour toutes les permissions qu'il possède et il renvoie les permissions à tous les sites pour lesquels il avait différé sa réponse.

L'indication du site prioritaire d'une permission est envoyée avec la permission.

Question 2 Décrire les variables de chaque site nécessaires à cet algorithme et leur initialisation éventuelle.

Question 3 Décrire les réceptions de requête et de permission ainsi que la l'entrée et la sortie de section critique.

Question 4 Avec trois sites, déroulez le scénario où le site `1` et le site `2` demandent simultanément la section critique. Vous devrez représenter les variables de chaque site et l'échange des messages.

Question 5 Expliquez **informellement** comment cet algorithme assure qu'un site qui demande la section critique finira par l'obtenir.

Sujet 2

On se propose de définir un algorithme d'accès à un fichier partagé par des sites. Ce fichier peut être accédé en lecture et en écriture. Les deux contraintes à respecter sont les suivantes :

Les lectures et les écritures sont exclusives

Les écritures sont mutuellement exclusives

Cet algorithme est une adaptation de l'algorithme de Ricart et Agrawala. Lorsqu'un site veut effectuer une opération (de lecture ou d'écriture) sur le fichier il envoie sa demande d'opération à tous les autres sites. Sur réception d'une demande d'opération, un site accorde sa permission si :
- soit il n'est pas intéressé par le fichier
- soit sa propre demande d'opération est compatible avec celle du site distant
- soit sa demande est plus récente que la demande du site distant

Dans le cas contraire, il diffère son accord. A la fin d'une opération, un site accorde toutes les permissions aux sites qu'il a retardé.

Question 1 Décrire les variables de chaque site nécessaires à cet algorithme.

Question 2 Décrire les réceptions de requête et de permission, les demandes de lecture et d'écriture et les fins de lecture et d'écriture.

Question 3 Avec trois sites, déroulez le scénario où le site `1` , le site `2` et le site `3` demandent simultanément un accès - le site `1` et le site `3` en lecture, le site `2` en écriture - . Vous devrez représenter les variables de chaque site et l'échange des messages.

Question 4 Expliquez **informellement** pourquoi les contraintes sont satisfaites.

Sujet 3

La diffusion atomique `Diffuser_Atomique(m)` est une primitive de diffusion utilisée par des applications qui a la propriété suivante :
"Si `m` et `m'` sont deux messages diffusés par cette primitive (pas nécessairement par le même émetteur) alors ils seront reçus par l'application de chaque site dans un même ordre"

Cette propriété permet d'ordonner les messages diffusés selon leur instant de traitement par l'application (et ceci indépendamment du site).
Attention, il ne faut pas confondre cette primitive avec la primitive de la couche réseau diffuser(m) utilisée par les algorithmes du cours !

Question 1 Donnez un exemple d'application informatique où il y a necessité de diffusion atomique.

Une manière simple de gérer la diffusion atomique consiste à séquentialiser les diffusions atomiques. Autrement dit, lorsqu'un site veut effectuer une diffusion atomique :
• son service demande la section critique,
• lorsqu'il l'a obtenue, il diffuse avec la primitive de bas niveau le message de l'application,
• il libère la section critique, une fois reçus les acquittements de la diffusion envoyés par les autres services.

On vous demande d'adapter l'algorithme de Carvahlo et Roucairol pour cette gestion.

Question 2 Décrire les variables de chaque site nécessaires à cet algorithme (elles incluent les variables de l'algorithme originel).

Question 3 Décrire l'implémentation de `Diffuser_Atomique(m)` et des réceptions de message.

Question 4 Avec trois sites, déroulez le scénario où le site `1` et le site `2` décident simultanément de diffuser un message. Vous devrez représenter les variables de chaque site et l'échange des messages.

Question 5 Quels sont les inconvénients d'une telle gestion ?

Sujet 4 : Implémentation d'un sémaphore sur un anneau virtuel

Dans ce problème, tous les sites sont interconnectés (réseau totalement maillé). Ils sont de plus organisés en un anneau logique (sites numérotés `0`,…,`n-1`) sur lequel circule un jeton.

On désire maintenant fournir une gestion d'un sémaphore à une application répartie. L'implémentation que l'on désire doit être équitable : tout site qui appelle `P()` ne restera pas bloqué indéfiniment. Tous les sites connaissent la valeur initiale du sémaphore cpt_0. La couche application n'émet qu'une requête à la fois (`P()` ou `V()` sans paramètre car il n'y a qu'un sémaphore).

On rappelle que pour qu'un `P()` soit passant, il faut que la condition suivante soit réalisée :

`(C)` nombre de `V()` effectués + cpt_0 > nombre de `P()` satisfaits

Le principe de la solution est le suivant :

(1) Chaque site tient à jour le membre gauche de `(C)` initialement égal à cpt_0

(2) Le jeton contient le membre droit de `(C)`

(3) Un `V()` sur le sémaphore par un site entraine une diffusion de cette information aux autres sites.

(4) Un `P()` sur le sémaphore est bloquant jusqu'à l'arrivée du jeton. Lorsque le jeton arrive sur le site, deux cas sont possibles :
• la condition `(C)` est satisfaite, le jeton est alors relaché après une mise à jour et le `P()` devient passant.
• la condition `(C)` n'est pas satisfaite, le jeton est alors conservé jusqu'à la satisfaction de la condition.

(5) Si le jeton arrive sur un site qui n'est pas en cours de `P()`, le jeton est aussitôt retransmis.

Attention ! Toutes les actions ne sont décrites ci-dessus.

Question 1 Définir les variables de la couche service.

Question 2 Définir les procédures `P()` , `V()` ainsi que la réception du jeton et du message "V".

Question 3 Décrire le scénario suivant :
• Il y a quatre sites,
• le sémaphore est initialisé à `0`,
• le site `0` effectue un `V()`,
• puis les sites `1` et `2` effectuent un `P()`,
• enfin le site `3` effectue un `V()`.

Question 4 Pourquoi cette gestion est-elle équitable ?

Question 5 Un grand nombre d'algorithmes sont basés sur le principe d'un jeton circulant sur un anneau. Quels sont les avantages et les inconvénients de ce mécanisme ?

Sujet 5 : Gestion de sections critiques par partage "semi-global" de jetons

Dans tout ce qui suit, l'ensemble des sites est {1,2,...,n}. On se propose de définir un nouvel algorithme d'accès à une section critique adapté de l'algorithme de Carvahlo et Roucairol à partir des observations suivantes sur cet algorithme :

- Il y a un ensemble de jetons partagés $\{\{i,j\}\}_{i,j \text{ de 1 à n},i\neq j}$ entre l'ensemble des sites.
- Le site i doit posséder le sous-ensemble de jetons $\{\{i,j\}\}_{j \text{ de 1 à n},j\neq i}$ pour entrer en section critique.

L'objectif est de diminuer les tailles de l'ensemble des jetons partagés et des sous-ensembles nécessaires à chaque site pour entrer en section critique. On appellera R_i le sous-ensemble de jetons nécessaires à i pour entrer en section critique.

Question 1 Quelle condition doivent vérifier les sous-ensembles R_i et R_j pour être assuré que les sites i et j ne rentrent pas simultanément en section critique ?

On décide de partager n jetons numérotés de 1 à n entre l'ensemble des sites. Le site i décide de l'attribution du jeton i en fonction des différentes requêtes qu'il reçoit. On l'appelle le propriétaire du jeton. Pour définir les sous-ensembles R_i, on range d'abord les identités des sites dans le tableau carré de taille suffisante. Voici deux exemples de tableau pour n=7 et n=11.

1	2	3
4	5	6
7		

1	2	3	4
5	6	7	8
9	10	11	

Pour un site i, l'ensemble des jetons R_i qu'il doit obtenir correspond à la ligne et à la colonne où il apparaît. Ainsi dans le premier tableau on a $R_5=\{2,4,5,6\}$ et $R_7=\{1,4,7\}$. Dans le deuxième tableau, on a $R_8=\{4,5,6,7,8\}$ et $R_{11}=\{3,7,9,10,11\}$.

Question 2 Montrez que la condition de la question 1 est toujours satisfaite. Donnez la taille maximale et la taille minimale d'un ensemble R_i en fonction de n. Indication : les expressions de ces bornes font intervenir les fonctions $\sqrt{x}$ et $\lceil x \rceil$ qui désigne le plus petit entier supérieur ou égal à x.

Description de l'algorithme

Initialement, un site i est au repos.

Lorsqu'il désire entrer en section critique, il envoie à tous les sites de R_i (y compris à lui-même) une requête datée de son heure logique courante puis il passe en attente.

Lorsque le site reçoit tous les jetons, il entre en section critique. Pour tester la condition d'entrée, il dispose d'un tableau de booléens indicé par R_i.

A la sortie de la section critique, il renvoie les jetons à leurs propriétaires respectifs à l'aide d'un message de libération et repasse au repos.
Quelque soit son état, chaque site maintient l'état du jeton dont il est le propriétaire (présent, prêté, réclamé) – initialement présent - et une file des requêtes qu'il a reçues. La file est triée selon l'âge des requêtes (c'est à dire le couple <heure logique, identité>). Cette file est initialement vide. Lorsqu'un site reçoit une requête, il l'insère dans sa file. Plusieurs cas se présentent alors :

1. Le jeton est présent (la file était vide), il envoie le jeton à l'émetteur de cette requête à l'aide d'un acquittement. Le jeton est prêté.
2. La file n'était pas vide et la requête n'est pas rangée en tête de la file (autrement dit, ce n'est pas la requête la plus ancienne), il ne fait rien de plus.
3. La file n'était pas vide et la requête est rangée en tête de la file. Si le jeton est prêté, il envoie une réclamation à l'émetteur de la seconde requête de la file (c'est à dire celui qui possède actuellement le jeton). Le jeton passe à l'état réclamé. Si le jeton était déjà réclamé, le site ne fait rien.

A la réception d'une réclamation, si un site est en attente et s'il possède le jeton, il renvoie le jeton à son propriétaire par un message de retour. Sinon il ignore la réclamation, car un message de libération a été ou sera envoyé.

A la réception d'un message de libération, le propriétaire extrait la requête de sa file et si la file est non vide, le jeton est prêté à la tête de la file par un acquittement. Sinon le jeton reste présent. A la réception d'un message de retour, le jeton est prêté à la tête de la file par l'envoi d'un acquittement.

Question 3 Décrivez les variables de chaque site nécessaires à cet algorithme et leur initialisation éventuelle. R_i sera considéré comme une constante calculée par le mécanisme indiqué plus haut.

Question 4 Ecrivez les primitives suivantes :

- prologue()
- épilogue()
- sur_réception_de(j,(req,h))
- sur_réception_de(j,(acq,h))
- sur_réception_de(j,(lib,h))
- sur_réception_de(j,(reclam,h))
- sur_réception_de(j,(retour,h))

On supposera que la file est dotée de méthodes d'insertion, d'extraction, de recherche de tête et de test de file vide. De plus, l'insertion préserve l'ordre des âges des requêtes.

Question 5 Lorsqu'un site demande un jeton, cela provoque au pire l'envoi d'une requête, d'une réclamation, d'un retour, de deux acquittements (vers le site demandeur et ultérieurement vers le site auquel on retire le jeton) et d'une libération. En vous servant du majorant de la taille d'un R_i trouvé en question 2, donnez une borne sur le nombre de messages échangés pour entrer en section critique.

6 Références

[Car83] O.S.F. Carvalho, G. Roucairol "On mutual exclusion in computer networks" Communication of ACM vol 26,2 février 1983 pp 146-147.

[Lam78] L. Lamport "Time, clocks, and the ordering of events in a distributed system" Communications of the ACM 21,7 juillet 1978 pp 558-565.

[Ric81] G. Ricart, A.K. Agrawala "An optimal algorithm for mutual exclusion in computer networks" Communication of ACM vol 24, janvier 1981 pp. 9-17.