Table des matières

1. Amélioration de Lamport

On passe d'une approche à 3 messages:

- `REQ` : demande d'accès à la section critique
- `ACK` : autorisation d'accès à la section critique
- `REL` : libération de la section critique

> 📝 – **Info**
> Ce que nous pouvons tirer comme conclusion est que nous sommes assez indirect dans notre approche. Ce que nous voulons est d'entrer en **section critique**.

1.1. Amélioration

App veut entrer en SC

J'envoie un `REQ` à tout le monde

Réception d'un `REQ`

Si je ne suis pas en SC

- Si je ne veux pas entrer en SC
  - Je réponds avec `OK`
- Si je veux entrer en SC
  - Si je n'ai pas la priorité, je réponds avec `OK`.
  - Si j'ai la priorité, j'attends d'avoir fini, puis je réponds avec `OK`.

Si je suis en SC

- J'attends d'avoir fini, puis je réponds avec `OK`.

App sort de SC

Je réponds par `OK` à tous les `REQ` en attente.

Réception d'un `OK`

Je le comptabilise.

- Si j'ai le `OK` de tout le monde, alors je peux entrer en SC.

Figure 1: Capture des slides du cours - Version améliorée

Grâce à cette nouvelle approche, nous pouvons désomrmais $2(n-1)$ messages par processus pour entrer en section critique.

1.1.1. Pseudo-code

Variables

`n`	*entier, constant*	nombre de processus
`self`	*entier, entre 0 et n-1*	mon numéro de processus
`ts`	*entier, init 0*	mon timestamp Lamport actuel
`hasRequested`	*booléen, init false*	ssi j'ai fait une demande de SC
`selfRequestTs`	*entier*	timestamp de cette demande
`missingOKs`	*entier*	nombre de OKs que j'attends encore
`waitingPs`	*ensemble d'entiers*	numéros des processus qui attendent mon OK.

Figure 2: Capture des slides du cours - Variables du pseudo-code

Initialisation

Écouter infiniment les événements suivants:

- Demande de SC de la couche applicative
- Sortie de SC dans la couche applicative
- Passage de `missingOKs` à `0`
- Réception de `{REQ, tsi}` du processus `i`
- Réception de `{OK, tsi}` du processus `i`

Figure 3: Capture des slides du cours - Initialisation du pseudo-code

Traitement : Demande de SC de la couche applicative.

```
ts += 1
hasRequested ← true
selfRequestTs ← ts
missingOKs ← n-1
Envoi de {REQ, ts} à tous les autres processus.
```

Traitement : Passage de `missingOKs` à `0`.

Autorisation de la couche application d'entrer en SC.

Traitement : Sortie de SC par la couche applicative.

```
ts += 1
hasRequested ← false
Envoi de {OK, ts} à tous les processus de waitingPs
Réinitialisation de waitingPs
```

Figure 4: Capture des slides du cours - Fonctionnement du pseudo-code

Traitement : Réception `{REQ, tsi}` du processus `i`.

```
ts ← max(tsi, ts) + 1
Si hasRequested et
   (selfRequestTs < tsi ou
   (selfRequestTs == tsi et self < i))
     Ajout de i dans waitingPs
Sinon
     Envoi de {OK, ts} à i
```

Traitement : Réception `{OK, tsi}` du processus `i`.

```
ts ← max(tsi, ts) + 1
waitingOKs -= 1
```

Figure 5: Capture des slides du cours - Fonctionnement du pseudo-code

1.2. Optimisation d'accès

Un autre point est que dans cette approche, si on récupère l'acces pour entrer en section critique, je peux en déduire que j'y ai accès tant que personne ne redemande l'accès.

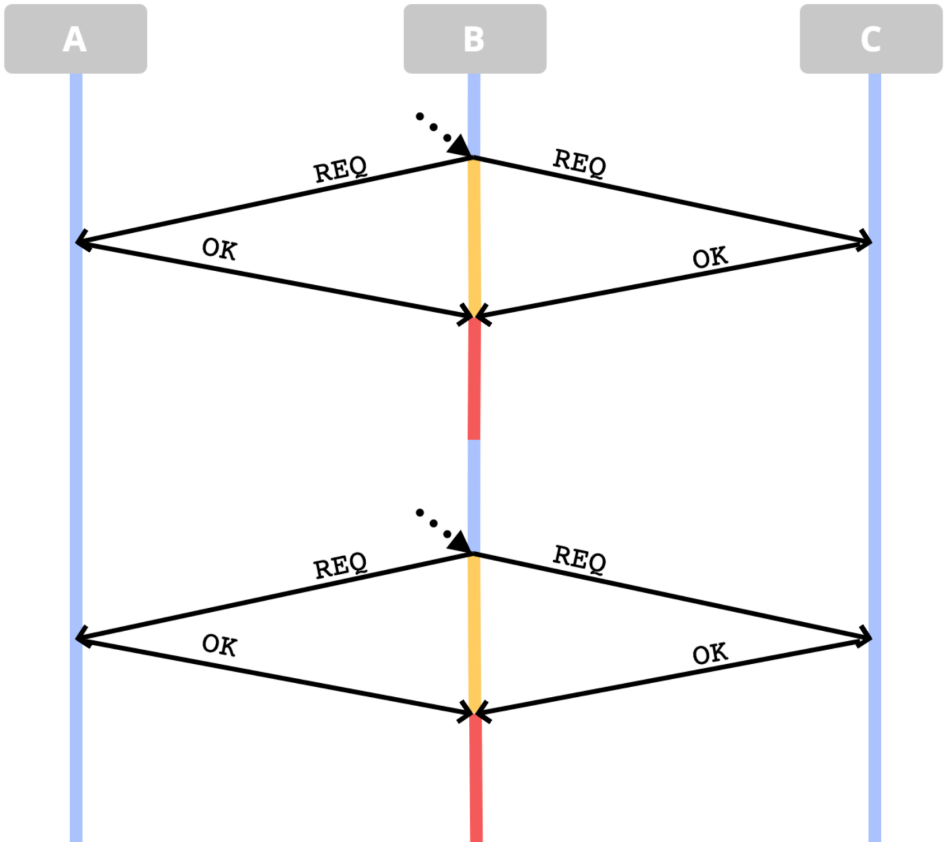

Figure 6: Capture des slides du cours - Optimisation d'accès

On peut voir cela un peu comme un jeton que l'on possède. Tant que l'on possède le jeton, on peut entrer en section critique.