PRR-Labo3

> Tiago Povoa et Burgener François

Donnée du laboratoire

Objectifs

- Comprendre le fonctionnement d'un algorithme d'élection avec pannes.
- Utiliser l'algorithme de Chang et Roberts avec pannes de processus pour déterminer le processus de meilleure aptitude parmi un ensemble de processus.
- Réaliser des communications UDP ou TCP en langage Go

Énnoncé du problème

> Réalisez un programme en langage GO qui implémente l'algorithme d'élection de Chang et Roberts avec pannes possibles des processus que nous avons vu en classe. Pour ce faire, nous avons des processus qui, de temps en temps, interrogent le dernier site élu, et si celui-ci n'est plus opérationnel, une élection est démarrée. Un processus en panne doit pouvoir se réinsérer au sein de l'anneau lors de sa reprise.

Comment démarrer

Nous avons 3 manières de lancer notre projet. Via deux script, windows ou alors via ligne de commande

Windows

Pour lancer le script il faut aller dans le dossier labo3 `prr-labo3/labo3` et de lancer le script `startWindows.bat`

```
$ ./startWindows.bat <nombre de processus> <liste aptitude>
```

Example

```
$ ./startWindows.bat 5 8 19 3 1 6
```

Cela va lancer 5 processus ou le processus :

- 0 aura l'aptitude 8
- 1 aura l'aptitude 19
- 2 aura l'aptitude 3
- 3 aura l'aptitude 1
- 4 aura l'aptitude 6

Ligne de commande

Pour lancer en ligne de commande il faudra tout d'abords aller dans le dossier `PRR-Labo2/labo2` et exécuter la ligne suivante dans différent terminal

```
go run main.go -proc <id du processus> -N <nombre de processus> -apt <apt du processus>
```

Les id des processus commencent à **0**

Example

```
go run main.go -proc 0 -N 3
go run main.go -proc 1 -N 3
go run main.go -proc 2 -N 3
```

Prise en main

Il est possible d'activer un mode debug dans `processus` pour le manager, le network ou task (au choix). Ceci facilitera l'analyse de l'exécution.

Travail réalisé

Task

Api disponible

Méthode	
Run	Démarre la tâche applicative. Elle va se charger d'émettre les echo OU tenter de lancer une élection si nécessaire.

Manager

Comme notre code est en anglais, nous avons utilisé "notification" à la place d'annonce.

Api disponible

Client

Méthode	
Init	"Constructeur": initialise la structure
RunElection	Prépare une élection à soumettre au réseau
GetElected	Obtient l'id du dernier processus élu

Réseau

Méthode	
SubmitNotification	Traîte la réception d'une requête d'annonce
SubmitResult	Traîte la réception d'une requête de résultat

En détails

On commence par démarrer le manager en appelant `Init()` puis, le network et task peuvent appeler les méthodes fournies par l'API.

Un Handler traîtera les demandes entrantes dans une goroutine `go handler()`.

Network

Api disponible

Méthode	
Init	"Constructeur": initialise la structure
EmitNotif	Envoie un message d'annonce au premier processus suivant trouvé
EmitResult	Envoie un message de résultat au premier processus suivant trouvé
EmitEcho	Appelé par la tâche applicative pour émettre un echo

En détails

Lorsque l'on doit émettre un message au suivant, notre algorithme va boucler comme ceci:

```
Pour N processus
avec k étant l'id de ce processus
on itère de k+1 jusqu'à k+N-1 % N
```

Si dans notre boucle, on obtient à nouveau l'id k (notre propre id), alors on sait qu'on est le seul processus actif dans le réseau.

La condition de passage d'un processus j à un processus j+1 est basée sur un timeout de 2T. Où T vaut 1s. Paramétrable dans `config.go`.

protocole

Type du Message	Format	Quoi
NOT	< NOT MessageNotif{ map[uint16]uint16 } >	Annonce
RES	< RES MessageResult{ id, map[uint16]bool } >	Résultat
ACK	< ACK Message{id} >	Acquittement
ECH	< ECH Message{id} >	Echo

Améliorations

- On pourrait limiter le nombre de messages transmis avec un algorithme de détection de suivant plus intelligent.