PC3R - TME4: Interfaces Distantes en *Go*

Equipe Enseignante PC3R

18/02/2021

¡¡¡¡¡¡¡ HEAD Lien vers les ressources de l'UE: `https://www-master.ufr-info-p6.jussieu.fr/2019/PC2R`
Lien vers la liste des conseillers municipaux: `https://s.42l.fr/pc3r-conseillers-municipaux` =======
Lien vers la liste des conseillers municipaux:
`https://www.data.gouv.fr/fr/datasets/r/d5f400de-ae3f-4966-8cb6-a85c70c6c24a`
(ou chercher `"data gouv elus municipaux"` dans un moteur de rechercher) ¿¿¿¿¿¿¿ 2db2b592e045fe0e362fd5de68030f676
Lien vers le squelette du TME: `https://frama.link/pc3r-tme4-squelette`

Objectif: Manipuler les canaux synchrones, manipuler des implémentations distantes d'interface, créer un mini-système client-serveur.

Rendu: Le rendu final pour ce TME doit être une unique archive contenant une arborescence de fichiers sources correspondant à celle du squelette fourni

Description Générale du Système

Les processus du système manipulent des **paquets** qui sont des entités abstraites (ce sont des "boîtes noires", qui ne donne pas accès directement à leur contenu) sur lequelles ont peut effectuer quatre opérations:

- `donne_statut` permet de connaitre le statut d'un paquet: Vide (`V`), En cours de modification (`R`) ou Fini (`C`).
- `initialise` prend un paquet Vide, lui donne un contenu initial, et une **liste de tâches à réaliser** sur ce contenu et passe son statut à En cours de modification.
- `travaille` effectue une unique tâche de la liste sur le contenu d'un paquet En cours de modification. Ensuite, s'il n'y a plus de tâche à effectuer, passe son statut à Fini.
- `vers_string` convertit le contenu du paquet en une chaîne de caractères.

Les processus du système sont divisés ainsi[1]:

- les **producteurs** fabriquent des paquets dont le contenu est vide qu'ils passent aux gestionnaires.
- chaque **gestionnaire** maintient sa propre file de paquets, qu'il récupère (et enfile) des producteurs et des ouvriers. Il propose le paquet en tête de file aux ouvriers. La file est de taille limitée (il ne peut plus accepter de paquet quand elle est pleine).
- les **ouvriers** récupèrent les paquets des gestionnaires et examinent leur statut: si le paquet est Vide, ils l'initialisent puis le renvoient aux gestionnaires, si le paquet est En cours de modification, ils travaillent une fois et le renvoient aux gestionnaires, et si le paquet est Fini, ils l'envoient au collecteur.
- le **collecteur** récupère les paquets terminés et accumule leurs contenus.

Le TME est divisé en deux parties (la première incluse dans la deuxième):

- dans la **Partie 1** les paquets produits sont **locaux**: l'appel à une de leur opération déclenche un calcul local.
- dans la **Partie 2** certains producteurs produisent des paquets **distants**: le contenu des paquets est effectivement stocké, et les méthodes sont effectivement calculées, **ailleurs sur le réseau**. Les paquets locaux sont implémentés par des moignons: l'appel à une des opérations du moignon

[1] C'est le pandémonium habituel des processus de TME, on pourra reprendre et modifier le TME3, si nécessaire.

déclenche un appel sur l'opération du paquet réel à travers le réseau. Un serveur est chargé de maintenir les paquets réels et leurs opérations.
- la **transparence** doit être absolue pour les gestionnaires, ouvriers et collecteur: dans leur comportement (leur code), rien ne leur permet de savoir s'ils manipulent un paquet local ou distant.

Il est conseillé de commencer par faire un schéma du système, des différents composants et des différentes communications entre composants.

Installation et Déroulement

1. Récupérer le squelette du TME à `https://frama.link/pc3r-tme4-squelette` et l'extraire dans un dossier.
2. Récupérer le fichier source à `https://frama.link/pc3r-conseillers-municipaux` et le sauvegarder **à la place** du fichier `/client/conseilleurs-municipaux.txt`
3. Remplir le code de `client/client.go` pour la partie 1 uniquement.
4. Compiler dans `/client` uniquement. Tester.
5. Modifier le code de `client/client.go` pour la partie 2.
6. Remplir le code de `serveur/serveur.go` et `serveur/travaux/travaux.go`.
7. Compiler dans `/client` et dans `/serveur`.
8. Lancer le client et le serveur, sur la même machine, dans deux terminaux différents, avec le même port passé en argument. Tester.

Partie 1: Personnes locales

Dans ce TME, les paquets manipulés par les processus du système doivent implémenter l'interface `personne_int`, qui contient les quatre opérations décrites ci-dessus.

Dans cette partie, chaque paquet local, de type `personne_emp` implémentant `personne_int` contient une `Personne` définie dans `/client/structures/structures.go`. Une fois créée, cette personne, définie[2] par son prénom, son nom, son âge et son code sexe, n'est pas manipulable autrement que par les opérations de l'interface. On décrit plus précisément la création et les deux opérations principales des paquets `personne_emp`:

- à la *création*, le paquet contient une personne vide, et attribut entier `ligne` correspondant à un numéro de ligne du fichier source et tableau `afaire` **vide** de fonctions des personnes dans les personnes. Son statut est `V`.
- `initialise` récupère dans le fichier source la `ligne`-eme ligne de texte, la convertit en `Personne` (le code de `personne_de_ligne` est donné) et remplit le contenu du paquet avec. Ensuite elle insere dans le tableau `afaire` un nombre aléatoire (1-5) de fonctions de travail (en appelant la fonction `UnTravail` de `/client/travaux/travaux.go`). Puis elle passe le statut du paquet à `R`.
- `travaille` applique à la personne contenue dans le paquet la fonction de travail en tête de `afaire` (cela modifie la personne) et retire cette fonction de `afaire`. Puis, si `afaire` est vide, elle passe le statut du paquet à `C`.

Quelques caractéristiques attendues du système:

- La lecture dans le fichier source est effectuée par une unique goroutine *lectrice*, appelée dans le code de `initialise`.
- Les gestionnaires reçoivent des paquets des producteurs **et** des ouvriers, et envoient les paquets aux ouvriers. Comme ils ont une capacité limitée, il est possible que les producteurs inondent les gestionnaires de nouveaux paquets, empêchant les ouvriers de retourner les paquets sur lequels ils ont terminé de travailler. Il faut implémenter un mécanisme empêchant la famine des ouvriers.
- Le collecteur maintient un journal (une chaîne de caractères). A chaque fois qu'il reçoit un paquet terminé il récupère son contenu avec `vers_string` l'ajoute comme nouvelle ligne à son journal. A la fin du temps, il imprime son journal sur la sortie standard.

[2] C'est réducteur, bien sûr.

- On passe à l'exécutable, en ligne de commande, un délai en millisecondes ; à la fin de ce délai, la goroutine principale prévient le collecteur de la fin du temps, attends une confirmation de celui-ci, puis s'arrête.

Partie 2: Personnes distantes

Dans cette partie, les paquets distants, de type `personne_dist` sont des moignons implémentant `personne_int` et ne contiennent pas de `Personne`. Ils font référence (à l'aide d'un identifiant unique) à une `personne_serv` située sur le serveur. Les personnes `personne_serv` implémentent les quatre opérations de l'interface `personne_int` (de manière tres similaire à `personne_emp`). On décrit plus précisément la manipulation des `personne_dist`:

- à la *création*, un identifiant unique frais est récupéré et écrit dans le moignon, ensuite, le serveur est contacté pour qu'une `personne_serv` vide y soit créée.
- les quatres autres opérations, quand elles sont appelées, déclenchent un appel au serveur, le lancement de l'opération sur la `personne_serv` correspondante sur le serveur, et le retour du résultat de l'opération.

Quelques caractéristiques attendues du système:

- L'initialisation d'une `personne_serv` sur le serveur ne fait pas obligatoirement appel à la lecture d'un fichier, on pourra, pour gagner du temps, faire en sorte que toutes les `personne_serv` sont initialisées avec la même personne (ou les mêmes personnes issues d'un petit ensemble).
- Un *proxy* doit être utilisé sur le client pour contacter le serveur: les opérations du moignon contactent le proxy (sur un canal), qui s'occupe de l'ouverture d'une *socket* TCP, de l'envoi d'un message indiquant quelle méthode de quel paquet (représenté par son identifiant) est appelée. Même si la méthode n'a pas de valeur de retour (`initialise` par exemple) le proxy attend une confirmation du serveur avant de rendre la main au moignon.
- Un *mainteneur* doit être utilisé sur le serveur. Il maintient une table d'association entre les identifiants et les `personne_serv` et permet de retrouver la personne serveur sur laquelle une méthode doit être appliquée.
- Le protocole de communication entre le client et le serveur est libre. Par exemple, le client peut envoyer l'identifiant et le nom de l'opération séparés par un caractère spécial, et le serveur peut répondre par une unique chaîne qui contient soit la valeur de retour de l'opération, soit `"OK"` pour les opérations sans valeur de retour. On peut se servir du fait que les opérations ne prennent pas d'argument et que les valeurs de retour éventuelles sont des chaînes de caractères.
- Les fonctions de travail (qui sont appelée lors de l'opération `travaille`) du serveur doivent être légèrement différentes de celles du client, afin qu'on puisse deviner dans le journal du collecteur, si une ligne correspond à une personne distante ou locale.
- Une goroutine du client est chargée de produire des identifiants frais pour les producteurs distants.