UPMC
SORBONNE UNIVERSITÉS

SRCS : Systèmes Répartis Client/Serveur

TD 03 – Pool de threads & protocole client/serveur

Janvier 2016

Exercice 1 : Mise en œuvre centralisée

Le but d'un pool de threads est de limiter le nombre de threads créés par un programme. Dans de nombreux cas (par exemple dans un serveur Web, lorsque de nombreux clients arrivent simultanément), une concurrence débridée et non contrôlée peut amener un écroulement de la machine. Cet exercice a donc pour but d'étudier les mécanismes liés aux pools de threads.

On se place dans le cadre du serveur Web multi-threadé gérant les commandes GET (TD 3, exercice 2, questions 5 & 6). Plutôt que de créer systématiquement une requête à chaque arrivée de client, on utilise un pool de threads.

Question 1

Dans un premier temps on considère un **Pool constant** c'est à dire un pool qui contient un nombre constant de threads crées initialement lors de la mise en place du pool.

Comment fonctionne ce pool ? Quelle structure de données est nécessaire à son fonctionnement ? A quel « grand problème système » peut-on rattacher la gestion de cette structure de données ?

Question 2

On considère que :
- chaque arrivée de client est représentée par un objet de type Job,
- que la structure de données fournit :
  - une méthode `put` pour déposer un Job
  - une méthode `get` pour retirer un Job.
- que l'on dispose d'un objet liste permettant de gérer une liste de données,
- que l'on dispose des primitives de synchronisation `wait()` et `notify()` vues en cours.

Donner le pseudo-code des méthodes `get` et `put`.

Question 3

Donner le pseudo-code du serveur Web utilisant le pool de thread.

Question 4

Lorsqu'on ne connaît pas a priori le nombre de clients attendus, quel problème pose le pool fixe ? Proposer une solution pour y remédier.

Question 5

Dans système à **pool dynamique** pour éviter de créer un thread à chaque arrivée de client, il peut être rentable de toujours disposer de threads en attente. Quel problème introduit-on alors ? Comment le résoudre ?

Question 6

On désigne sous le terme de *supervisor*, l'objet chargé de mettre en œuvre la solution à la question précédente. Donner son pseudo-code.

Exercice 2 : Conception d'un protocole c/s au-dessus d'un transport non fiable

Le but de cet exercice est de concevoir un protocole client/serveur qui traite différents cas de pannes pouvant survenir dans un environnement distribué.

On rappelle qu'un protocole client/serveur met en relation deux entités (le client qui demande un service et le serveur qui exécute un traitement suite à ce service et fournit une réponse) et deux messages (le message d'appel envoyé par le client au serveur, et la réponse du serveur au client).

Dans l'ensemble de cet exercice, on fera l'hypothèse d'une couche de transport non fiable (aucune garantie quant à la livraison des messages – par exemple UDP) et ne fournissant aucun mécanisme de détection, notification, ou correction de pannes. Il s'agit donc d'intégrer ces éléments au protocole client/serveur.

Question 1

Dans le cadre d'un protocole client/serveur, quels types de pannes peut-on rencontrer ?

Question 2

En terme de programmation, quel mécanisme peut-on utiliser pour détecter une panne ?

Question 3

On suppose que le client fonctionne normalement. Quels sont les différents problèmes que l'on peut rencontrer lors de l'envoi du message d'appel ? Le client peut-il faire la distinction entre ces problèmes ? Proposer une solution pour les traiter.

Question 4

On suppose que le serveur fonctionne normalement. Quels sont les différents problèmes que l'on peut rencontrer lors de l'envoi du message de retour ? Proposer une solution pour les traiter.

Question 5

En supposant les problèmes des questions 3 et 4 résolus, on veut maintenant pouvoir détecter les pannes survenant pendant le traitement de la requête. Quels sont les problèmes à régler ? Proposer deux solutions différentes.