université PARIS DIDEROT PARIS 7

EXAMEN PROGRAMMATION RÉSEAU 15/05/2014 8^{30}-11^{30}

Positionnez impérativement vos mobiles en mode « avion ».
Les seuls documents autorisés sont les documents « sur papier » ;
à l'exclusion de la copie ou les brouillons des voisins.

Problème

Les systèmes d'exploitation centralisent la gestion des services réseaux. Plutôt que de contrôler indépendamment le lancement de chaque application chargée d'un service réseau, les systèmes de la famille FreeBSD proposent d'utiliser une application de nom `launchd` qui se charge des connexions entrantes. Le rôle de `launchd` est donc d'attendre des connexions, puis de lancer l'application adéquate lorsqu'un client se connecte sur un port donné. De la sorte, les applications « service » ne se préoccupent que de gérer la partie « intéressante ».
On se propose ici d'écrire une application de type `launchd` qui prendra sa configuration dans un fichier donné en paramètre et dont le contenu sera constituée de lignes comme dans l'exemple suivant :

```
3078 J SuperService
4093 J MegaService dobedo
4095 J HyperService bebopalula sheismybaby
8192 E trucmuch bidule
```

Chaque ligne correspond donc à un service. Le premier « mot » est le numéro du port du service sur lequel se connectent les clients, le deuxième « mot » (une lettre) désigne le type de service (`J` pour un service « Java », `E` pour un service non-java), le troisième paramètre désigne la classe Java qui implémente le service (pour un service « Java » ou un exécutable de `/usr/sbin` pour les services non-java. Les autres « mots » sont de simples arguments à transmettre tels quels au service (arguments de type « ligne de commande »).
Ainsi, la seconde ligne correspond à un service délégué à l'exécution de la classe `MegaService.class` dans une machine virtuelle qui recevra en argument l'argument `dobedo`, lorsqu'un client se connectera en TCP sur le port 4093.

1. Écrire en java une méthode `Service[] parseConfigFile(String filename);` qui reçoit en paramètre le nom du fichier de configuration formaté comme indiqué ci-dessus et renvoie un tableau de services. La classe `Service` est simplement la suivante :

   ```
   enum ServiceType { JAVA, NON_JAVA }

   class Service {
     public ServiceType  type;
     public int          port;
     public String       command;
     public SelectionKey cle;
   }
   ```

   Cette méthode ne « remplit » pas le champ `cle` qui devra être positionné à `null`.
2. Écrire une méthode `SelectionKey addServerSocketOnPort(Selector s,int port);` et qui permet d'ajouter au sélecteur `s` une `ServerSocketChannel` correctement configurée et prête à attendre sur le `port` TCP donné. La méthode renverra la clé de sélection associée...
3. Écrire une méthode `Selector prepareSelector(Service[] services);` qui renvoie un sélecteur prêt à attendre des connexions entrantes sur chacun des ports du tableau des services. Au passage, cette méthode remplira les champs `cle` des entrées du tableau des services...
4. Écrire une méthode `Service getService(Service []services,SelectionKey cle);` qui renvoie le service associé à la `cle` donnée.
5. Écrire une méthode `void launchdMain(String filename);` qui combine des appels aux fonctions précédentes de sorte que soit correctement paramétré un sélecteur, et qui dans une boucle infinie :
   - effectue une attente de sélection,
   - retrouve pour toutes les opérations possibles de la sélection le service correspondant et
   - appelle une méthode de prototype `void StartService(Service s,Socket canal);` sur chacun d'eux.
6. Écrire la méthode `void StartService(Service s,Socket canal);` qui
   - lance l'exécution externe d'un `Process`. Dans le cas d'un service Java (par exemple `MegaService dobedo`) il faut lancer une machine java via `java MegaService dobedo` ; dans le cas d'un service non java (par exemple `trucmuch bidule` il faudra lancer l'exécution de la commande `/usr/sbin/trucmuch bidule`

— démarre un (ou plusieurs selon votre façon de voir les choses) `Thread` lequel renvoie en entrée du `Process` tout ce qui est lu sur le `canal` et à l'inverse renvoie sur le canal ce qui est lu en sortie du `Process`. On peut ici utiliser plusieurs threads ou un thread avec un selecteur...

On notera donc que les services se contentent de lire et écrire sur leurs entrées et sorties standards.
`Process Runtime.exec()` est certainement une méthode qui vous sera utile, etc.

7. Écrire en C un service `echo` qui se contente de renvoyer sur sa sortie standard tout ce qu'il lit sur son entrée standard, jusqu'à épuisement de l'entrée.
8. Écrire en Java un service `date` qui se contente d'écrire la date courante sur sa sortie standard et sous la forme d'une chaîne de caractères et termine immédiatement.
9. Écrire les lignes de configuration correspondant à ces deux services.
10. On aurait pu se passer entièrement de threads, comment ? (Décrivez simplement la solution que vous entrevoyez, n'écrivez pas de code). On aurait pu se passer entièrement de sélecteurs, comment ? (Même type de description).