Université Paris Diderot — Master 1 – Programmation Réseaux – Année 2011–2012

Examen – Vendredi 22 Juin 2012

Tout document papier est autorisé. Les téléphones portables, ordinateurs, comme tout autre moyen de communication vers vos voisins ou l'extérieur sont strictement interdits. Le temps à disposition est de 3 heures. Justifier correctement vos réponses. Lire le sujet dans son intégralité avant de commencer...

1 Exercice

Les systèmes Unix *à l'ancienne* gèrent les services réseau à l'aide d'un super-démon. On rappelle qu'un démon est simplement une application (souvent de type serveur) qui tourne en fond pour effectuer des tâches d'ordre général (gestion des impressions, surveillance, gestion du courrier, etc). Plutôt que de lancer les démons un par un, à la main (ou à l'aide d'un script, etc), ces systèmes utilisent une application paramétrable nommée `inetd` (InterNET Daemon). Son rôle essentiel est d'attendre des connexions entrantes sur différents ports, puis de lancer le service (l'application serveur) adéquat lorsqu'un client se connecte. Ainsi la gestion du réseau (connexion, etc) est réalisée par le super-démon lui-même (et donc centralisée) et les démons sont simplement réduits à la partie intéressante, sachant que pour le cas de TCP, les services reçoivent les données sur leur entrée standard et envoient leurs données sur la sortie standard (un service peut donc être facilement écrit dans n'importe quel langage, et n'importe quelle application Unix qui lit sur son entrée standard et écrit sur sa sortie standard peut facilement devenir un service réseau).

On se propose ici d'écrire une application de type `inetd` qui prendra sa configuration dans un fichier donné en paramètre et dont le contenu sera constituée de lignes comme dans l'exemple suivant :

```
3078 tcp /usr/bin/bidule -l -t -v
4093 tcp /usr/local/bin/machin -m
4095 tcp echo
4095 udp echo
```

Chaque ligne correspond donc à un service. Le premier mot est le numéro du port utilisé par le service, le mot suivant désigne le type de connexion attendue (tcp ou udp), le troisième mot désigne le chemin d'accès à l'application et les autres mots qui suivent les arguments à transmettre à l'application.

1. écrire en java une méthode `InetService [] parseTCP(String filename);` qui reçoit en paramètre le nom du fichier de configuration formaté comme indiqué au-dessus et renvoie un tableau de services (uniquement les services TCP). La classe `InetService` est simplement la suivante :

```
class InetService {
  public static final int TCP=0;
  public static final int UDP=1;
  public int port;
  public int mode; // TCP ou UDP
  public String[] command; // tous les arguments de la commande
  public SelectionKey cle;
}
```

Pour écrire la méthode, il est conseillé d'utiliser la classe `StringTokenizer` afin de découper une ligne en mots... Cette méthode ne remplit pas le champ `cle` (positionné à `null`).

2. écrire une méthode `SelectionKey addServerSocket(Selector s,int port);` et qui permet d'ajouter au sélecteur `s` une `ServerSocketChannel` correctement configurée et prête à attendre sur le `port` donné. La méthode renverra la clé de sélection associée...
3. écrire une méthode `Selector prepareSelector(InetService[] 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` du tableau des services...
4. écrire une méthode `InetService getService(InetService []services,SelectionKey cle);` qui renvoie le service associé à la `cle` donnée.
5. écrire une méthode `void superService(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 lors de la sélection le service correspondant et appelle une méthode de prototype `void StartService(String []command,Socket canal);`
6. écrire la méthode `void StartService(String []command,Socket canal);` qui si la commande à lancer ne commence pas par le caractère `/` appelle la méthode `void StartJavaService(String []command, Socket canal);` et `void StartNonJavaService(String []command,Socket canal);` sinon
7. écrire la méthode `void StartJavaService(String []command, Socket canal);` qui exécute le service implémenté sous la forme d'une classe Java (laquelle est exécutée par un appel à `Java classe args`). Cette application devra communiquer avec un Thread interne au super-démon qui se chargera de créer le `Process` adéquat, transmettre les données dans les deux sens et attendre la fin du `Process`.
8. écrire en Java un service `echo` qui se contente de renvoyer au client exactement ce que celui-ci lui envoie jusqu'à ce que le client rompt la connexion
9. écrire en Java un service `date` qui se contente de renvoyer la date courante au client et termine immédiatement
10. écrire les lignes de configuration correspondant à ces deux services