iUT ORSAY
Département Informatique

DUT Informatique
Module Système S4 C
2009 / 2010

Travaux Pratiques n° 4 : Démons

Nom(s) :
Groupe :
Date :

Objectifs : savoir mettre en place un serveur démon sous Unix.

1 Avant propos

Durant ce TP, vous aurez à effectuer des manipulations en tant qu'administrateur (utilisateur *root*). Lorsque vous vous connecterez au système avec une interface graphique (c'est le mode par défaut), vous utiliserez l'utilisateur habituel `user1` (mot de passe : `user1`) puis utiliserez dans un terminal la commande `su` pour acquérir les droits d'administration (le mot de passe du super-utilisateur `root` est `azerty`).
Il est en effet préférable de ne pas lancer d'environnement graphique comme kde ou gnome sous `root` pour ne pas que tous vos processus aient les droits de `root` (ce qui pose des problèmes de sécurité et ne sert le plus souvent à rien). Par contre, si vous vous connectez au système en mode texte vous pouvez le faire directement sous `root`.

2 Client/Serveur sous Unix

Le serveur nécessaire à la commande `runame` développé dans le TP précédent doit s'exécuter en permanence sur la machine pour pouvoir être utilisé. Il accapare par conséquent des ressources du système qui ne sont réellement utilisées que rarement. De plus, ce processus est lié à l'utilisateur (et même au terminal) qui l'a lancé, donc si celui-ci se deconnecte de la machine (ou ferme le terminal), le serveur s'arrêtera. Pour éviter ces inconvénients, Unix prévoit un mécanisme permettant de détacher un processus de son processus père et de ne le réveiller qu'à la demande. On parle de *démon* (ou *daemon* en anglais).

Les démons rendent le plus souvent des services *via* le réseau, on doit donc être particulièrement vigilant aux aspects sécurité durant leur conception. Typiquement on veillera à trois aspects :

1. Le démon ne devra avoir que les droits strictement nécessaires à son exécution (création de fichiers, répertoires accessibles, utilisateur sous lequel tourne le démon).
2. Le démon tracera ses activités et autres messages d'erreur dans un fichier (appelé fichier de *log*), de manière à pouvoir suivre son exécution en détail.
3. Le démon devra pouvoir réagir à certains signaux pour, par exemple, pouvoir quitter proprement ou relire un fichier de configuration.

3 Passage (simple) d'un processus en démon

Dans un premier temps nous allons étudier le passage simple d'une application de serveur « standard » en un démon. Nous allons modifier le serveur réalisé dans le TP précédent pour cela (attention à bien conserver l'original que l'on reprendra pour une question ultérieure).

Lancez votre serveur dans un terminal. Depuis un autre terminal, entrez la commande `pstree` qui vous donne l'arborescence complète des processus. Dessinez les parties de l'arbre concernant votre serveur (pour les ancêtres, ne dessiner que quelques processus frères).

En principe, l'arbre généalogique du serveur sans modification montre que celui-ci dépend au moins d'un terminal (celui où il a été lancé) et du processus `init` (le processus racine). Il peut aussi dépendre de nombreux autres processus liés notamment à l'environnement graphique. Le but d'un passage en démon est de devenir indépendant, donc de se détacher de tous les processus à l'exception évidemment d'`init`. Cette opération se réalise en deux temps :

1. Le serveur va se dupliquer grâce à la primitive `fork` et faire terminer le processus père. Ainsi le fils orphelin sera adopté par le processus `init`. Le code du serveur continuera simplement dans le fils.
2. Le processus fils fera appel à la primitive `setsid` avant de continuer le code du serveur :

```
#include <unistd.h>
pid_t setsid(void);
```

   Cette primitive a pour but de rendre le fils « chef de sa session ». En effet il existe, en dehors des liens de filiation, une notion de *sessions* et de *groupes de processus* que nous n'étudierons pas en détail (disons simplement que les processus sont regroupés en groupes, eux-mêmes regroupés en sessions). Elle sert en particulier à la transmission des signaux. Par exemple quand un chef de session (ou de groupe) reçoit un signal, le signal est aussi envoyé à tous les processus de la session (ou du groupe). Un père est par défaut chef de groupe pour ses fils, ainsi lorsqu'on le tue (par `SIGKILL` par exemple) les fils sont tués aussi. En appelant `setsid`, le processus crée sa propre session dont il est le chef et se retrouve donc indépendant des autres.

Modifiez votre serveur pour qu'il devienne un démon. **Vous joindrez le code source du serveur au compte rendu.**

Après avoir lancé votre serveur démon, entrez la commande `pstree` et dessinez l'arbre généalogique du serveur. Comment faîtes-vous pour l'arrêter proprement ?

4 Super-démon `inetd`

Afin d'éviter que l'on ne mette en place perpétuellement un démon par service et donc d'avoir un nombre maximal de démons dont la plupart seront inutilisés, toutes les requêtes sont reçues par un processus particulier : le super-démon `inetd`. Sur réception d'une requête, ce démon :
– se duplique (par un appel à `fork`),
– côté père, se replace en attente de nouvelles requètes,
– côté fils, crée une socket destinée à la communication entre le client et le serveur. L'entrée standard et la sortie standard du fils sont redirigées vers cette socket (par des appels à `dup`). Le code du fils est alors remplacé par le code du serveur (par un appel à `exec`). Dans ce code du serveur, il suffit donc d'écrire sur la sortie standard pour envoyer des données par la socket créée et de lire sur l'entrée standard pour lire les données reçues sur la socket.

L'interface de configuration du super-démon `inetd` est le fichier */etc/inetd.conf*. L'ensemble des serveurs, les caractéristiques de la socket utilisée pour la communication entre le client et le serveur sont paramétrables à partir de ce fichier. Son contenu se présente de la manière suivante :

```
echo    dgram   udp     wait    root    internal
ident   stream  tcp     wait    identd  /usr/sbin/identd    identd
ftp     stream  tcp     nowait  root    /usr/sbin/proftpd   proftpd
```

Chaque ligne de ce fichier contient :

1. Le nom du service.
2. Le type de la socket : `stream` ou `dgram`.
3. Le protocole de niveau transport utilisé : `tcp` ou `udp`.
4. une option : toujours `wait` en sockets `dgram`, qui signifie que seul un serveur de ce type peut exister à un instant donné, et le plus souvent `nowait` en sockets `stream`.
5. Un nom d'utilisateur qui sera le propriétaire du démon lorsqu'il sera activé.
6. La référence absolue du fichier exécuté, ou `internal` dans le cas de services internes.
7. Des paramètres éventuels.

Quels sont les services activables par `inetd` sur votre système ? Pourquoi à votre avis, y en a-t-il si peu ?

Le numéro de port du service que devra utiliser un client pour un service donné devra se trouver dans */etc/services*.

Relevez quelques numéros de ports de services que vous connaissez.

5 Visualisation des sockets

La commande `netstat` permet de visualiser les sockets crées sur un système.

Expliquez à l'aide des pages de `man` le résultat des commandes suivantes :
– `netstat -at`

– `netstat -au`

`telnet` est un service `inetd` de connexion à distance : le client `telnet` fait une requête au serveur démon `telnetd`. Connectez-vous *via* `telnet` (commande `telnet nom_ou_IP_machine`) à la machine d'un autre binôme. (vous aurez peut-être besoin avant de démarer le super-démon `inetd` par la commande `/etc/init.d/openbsd-inetd restart`).

Comparez les résultats donnés par `netstat` avant et durant une session `telnet`.

6 Réalisation d'un démon Unix

On cherche à transposer en démon Unix le serveur réalisé dans le TP précédent pour répondre à la commande `runame`. Comme indiqué en section 4, le code est largement simplifié puisque c'est le super-démon `inetd` qui prend en charge à partir des fichiers de configuration la création et l'attachement de la socket dédiée au serveur. Les lectures sur la socket se font à présent simplement sur l'entrée standard (descripteur `0`) et de même les écritures se font sur la sortie standard (descripteur `1`) (en résumé, on a plus à se préoccuper de la gestion de la socket dans le code du serveur). Plus besoin non plus d'une boucle autour des réceptions/émissions, puisqu'à chaque requête, `inetd` se chargera de remettre en place le serveur.

Écrivez le programme de votre démon *runamed.c*, adapté de *serveur.c*. (ce code subira encore une évolution avant d'être remis au compte rendu).

Mettre en place le serveur pour qu'il soit pris en compte par `inetd` :

Quelles modifications faut-il faire dans */etc/services* ?

Quelles modifications faut-il faire dans */etc/inetd.conf* ?

Pour que les changements dans les fichiers de configuration soient pris en compte, il est nécessaire de demander à `inetd` de les relire. Cela se réalise en lui envoyant le signal `SIGHUP` à l'aide de la commande `killall -HUP inetd`.

Tester. Cela fonctionne-t-il ? Visualise-t-on un processus serveur ? Visualise-t-on de nouvelles sockets ?

7 Traçabilité de l'activité d'un démon

Par essence, un démon s'exécute silencieusement à l'écart des autres processus du système. Pour savoir si tout se déroule comme prévu, un démon rend généralement compte de son activité dans des fichiers de journaux (fichiers de *log*). Deux approches sont possibles. La première et la plus simple est de faire en sorte que le démon gère seul la création d'un fichier dans le répertoire standard */var/log* (dans notre cas, un fichier */var/log/runamed.log*) et ajoute à la fin de ce fichier de l'information à chaque fois qu'un évènement notable survient.

Une autre politique est de recourir au démon `syslogd`. Ce démon permet de regrouper des messages système de provenances diverses dans des classes (telles que `LOG_DAEMON` pour les démons) et de diriger ces classes dans des fichiers (en général */var/log/messages*). Il se configure *via* le fichier */etc/syslog.conf* où on définit des classes et des règles de redirection des messages en fonction de leur priorité (par exemple `LOG_ERR` pour les erreurs, `LOG_WARNING` pour les avertissements et `LOG_INFO` pour les messages d'information).

On considère le code suivant :

```
openlog("runamed",LOG_PID,LOG_DAEMON);
syslog(LOG_INFO,"traitement␣connexion␣%s",mon_client);
closelog();
```

En vous aidant des pages de `man` des trois primitives système appelées dans cette portion de code, expliquer en détail leurs fonctions et leur résultat après lecture du fichier */etc/syslog.conf*.

Complétez le programme de votre démon *runamed.c* de manière à ce que la traçabilité soit assurée par le démon `syslogd`. **Vous joindrez ce code au compte rendu de TP.**

8 Sécurité

L'utilisation basique de `inetd` est aujourd'hui considérée comme une faille de sécurité dans un système : aucun contrôle n'est effectué sur le client qui sollicite `inetd`. C'est pourquoi on met en place trois types de solutions :

– Désactiver tous (ou quasiment tous) les services exploitant `inetd`.
– Utiliser `inetd` en conjonction avec un outil nommé `TCPWrapper` qui filtre les requètes des clients en fonction de plusieurs critères : adresse du client, plage horaire... C'est cette solution qui est aujourd'hui la plus répandue. Elle ne remplace pas un vrai *firewall*.
– Utiliser `xinetd` qui remplit des fonctionnalités similaires à l'ensemble `inetd` + `TCPWrapper`. Cette solution remplace peu à peu la précédente mais n'est pas encore en place partout.

Ces aspects sécurité seront couverts en cours de réseaux, les principes vus dans ce TP pour la mise en place de démons restent exactement les mêmes avec `xinetd`.