

Rapport de projet de TP

Daemon client/serveur d'information sur les utilisateurs et les processus

Sommaire

Daemon client/serveur d'information sur les utilisateurs et les processus.....	1
1) Manuel d'utilisation.....	3
1.1) Téléchargement et lancement.....	3
1.2) Commandes client.....	3
1.2.1) Commande usern.....	3
1.2.2) Commande useru.....	3
1.2.3) Commande proc.....	3
1.2.4) Commande exit.....	4
1.2.5) Commande help.....	4
1.3) Commandes serveur.....	4
2) Rapport Technique.....	4
2.1) Util.....	4
2.2) Client.....	4
2.3) Server.....	5
2.4) Info proc.....	5
2.5) Info user.....	5
2.6) Makefile.....	6
2.7) Améliorations possibles.....	6
3) Conclusion.....	6

1) Manuel d'utilisation

1.1) Téléchargement et lancement

Le code source est disponible sur GitHub à l'adresse suivante :

<https://github.com/antoineguillory/DaemonUserProcInfo> . Le serveur se lance bien entendu avant le client sans arguments. Le client se lance également sans arguments.

1.2) Commandes client

Lors du lancement du client (si vous avez lancé le serveur au préalable) vous êtes invité.e à renseigner une commande parmi celles suivantes

1.2.1) Commande usern

Le client peut demander des informations sur un utilisateur en renseignant son nom d'utilisateur via cette commande. Plusieurs informations sont recueillies et affichées.

```
Command ?> usern
Username ?>root
[CLIENT] : root
Username = root
Password = x
Uuid = 0
Guid = 0
GECOS = root
Directory = /root
Shell = /bin/bash
```

Texte 1: la commande usern

1.2.2) Commande useru

Le client peut demander des informations sur un utilisateur en renseignant son UID via cette commande. Les mêmes types d'informations que pour *usern* sont affichées

```
Command ?> useru
UID ?>39
[CLIENT] : 39
Username = irc
Password = x
```

Texte 2: la commande useru

1.2.3) Commande proc

Cette commande permet au client d'afficher les informations relatives à un processus en particulier, comme son PID, son PPID ou les GID des groupes auxquels le processus appartient par exemple

```
Command ?> proc
PID ?>23255
[CLIENT] : 23255
Cmdline: /usr/lib/x86_64-linux-gnu/notify-osd
Name: notify-osd
Umask: 0002
State: S (sleeping)
Pid: 23255
PPid: 1440
Uid: 1000 1000 1000 1000
Gid: 1000 1000 1000 1000
Groups: 4 24 27 30 46 113 128 1000
Threads: 4
```

Texte 3: la commande proc

1.2.4) Commande exit

Cette commande permet de quitter le client.

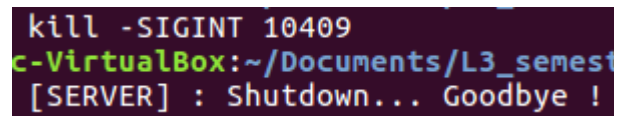
1.2.5) Commande help

Affiche l'aide relative aux commandes du client.

1.3) Commandes serveur

Le serveur n'a pas de commandes, cependant, vous pouvez libérer les ressources proprement en envoyant un SIGINT au serveur : ce signal est traité.

Remarque : il est impossible de lancer deux serveurs sur la même machine.



```
kill -SIGINT 10409
c-VirtualBox:~/Documents/L3_semes
[SERVER] : Shutdown... Goodbye !
```

Texte 4: Envoi d'un SIGINT au serveur

2) Rapport Technique

2.1) Util

Nous avons créé un fichier de fonctions utilitaires pour effectuer différentes tâches redondantes et communes au client et au serveur, afin de factoriser le code. On peut y retrouver notamment une fonction de concaténation, de génération aléatoire de chaîne de caractères ou encore un prédicat permettant de déterminer si un fichier existe, on peut y retrouver également une fonction de projection de mémoire de la mémoire partagée.

2.2) Client

Les requêtes sont des structures composées de la SHM permettant au serveur de communiquer par la suite ses résultats. Une requête est composée d'un sémaphore permettant l'attente passive du client, la mémoire partagée correspondant à la réponse du serveur, du nom de la commande, les paramètres de la commande et le PID du client ; si un problème interne au serveur arrive, le serveur doit pouvoir être capable de tuer le client avant de se tuer.

A sa création, le client ouvre le tube nommé unique initialisé par le serveur qui permet le passage des requêtes, initialise sa mémoire partagée et son sémaphore. Il tente ensuite de répondre au attente de l'utilisateur pour former une nouvelle requête grâce à la fonction `extract_request()`.

Une fois la requête correctement formulée, le client écrit cette dernière dans le tube nommé décrit précédemment puis attend passivement (grâce au sémaphore) la réponse du serveur pour la lire dans la mémoire partagée.

2.3) Server

A son ouverture le serveur initialise son tube nommé et son comportement pour le traitement de SIGINT. Il attend passivement une requête de la part du client, puis lance un nouveau thread. Ce dernier va traiter la requête à l'aide de la fonction `treatment_request()`. Grâce à un tube anonyme, le traitement va lancer un processus fils exécutant la commande demandée qui écrit la réponse dans le tube anonyme. Le processus père va lire la réponse de la commande dans le tube anonyme à l'aide de la fonction `read_response` puis l'écrire dans la mémoire partagée du client et l'informer.

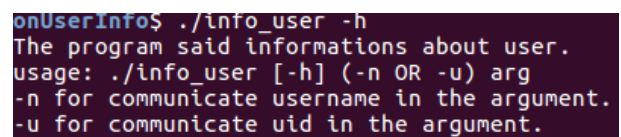
2.4) Info proc

Ce programme permet d'en apprendre plus sur un processus. En communiquant le pid d'un processus, on obtient les informations tel que son nom par exemple.

Grâce à la fonction `info_proc_read()`, on trouve, dans le fichier de chemin de path « `/proc/[PID]/` », les informations lues par la fonction passée en paramètre : par exemple, pour lire la ligne de commande utilisée pour lancer le processus, on va lire dans le fichier « `cmdline` » avec la fonction `read_content()` pour lire la totalité du fichier. Mais aussi dans le fichier « `status` », on lit les informations désirées grâce à la fonction `read_status()`. Cette fonction sélectionne les informations en passant celles inutiles grâce à `skip()` et en lisant celles utiles grâce à `read_line()`.

2.5) Info user

Nous avons désiré créer un `info_user` indépendant. Celui-ci a donc toutes les fonctionnalités d'un programme autonome : option d'aide « `-h` » par exemple.



```
onUserInfo$ ./info_user -h
The program said informations about user.
usage: ./info_user [-h] (-n OR -u) arg
-n for communicate username in the argument.
-u for communicate uid in the argument.
```

Texte 5: Option d'aide de `info_user`

Nous avons deux options possibles comme l'indique l'aide :

- n : pour indiquer le nom du l'utilisateur dont on veut avoir les informations
- u : pour indiquer l'uid du l'utilisateur dont on veut avoir les informations

ces deux fonctionnalités sont le résultat de la fonction `info_user()`. Celle-ci ouvre le fichier `/etc/passwd` pour y lire les différents utilisateurs. Suivant l'option sélectionnée, la fonction `select_user()` va, grâce au pointeur de fonction « `cmp` » et « `get` », trouver ou pas la ligne correspondante à l'utilisateur cherché en comparant, soit les UID, soit les noms. On pourra ensuite écrire les informations extraites de cette ligne ou que l'utilisateur n'existe pas.

```
onUserInfo$ ./info_user -n root
Username = root
Password = x
Uuid = 0
Guid = 0
GECOS = root
Directory = /root
Shell = /bin/bash
```

Texte 6: Option "-n" pour le nom

```
onUserInfo$ ./info_user -u 39
Username = irc
Password = x
Uuid = 39
Guid = 39
GECOS = ircd
Directory = /var/run/ircd
Shell = /usr/sbin/nologin
```

Texte 7: Option "-u" pour l'

2.6) Makefile

Le Makefile du projet vous permet de créer indépendamment le serveur, le client, `info_proc` et `info_user`, ou bien tout d'un coup si `make all`.

2.7) Améliorations possibles

Notre projet pourrait être amélioré de quelques manières afin de faciliter son utilisation, par exemple, nous aurions pu mettre en place un système d'historique des commandes pour que l'utilisateur puisse naviguer avec les flèches directionnelles sur ses dernières commandes. On pourrait également complexifier les requêtes pour que l'utilisateur puisse demander précisément l'information spécifique sur l'utilisateur et ou le processus ; par exemple si un utilisateur veut juste le PGID d'un processus par exemple lui permettre d'afficher que le PGID.

3) Conclusion

Ce projet de TP nous a permis de manipuler des IPC : des tubes, des SHM, des signaux, d'appliquer nos connaissances sur les sémaphores, et d'utiliser un VCS (ici, GitHub). Le fait que le modèle de l'application soit contraint (schéma dans l'énoncé) nous a permis de nous conformer à un cahier des charges et à rendre un livrable correspondant à celui-ci.