LES DÉMONS

Guillaume Chanel

LES ÉTAPES (LES MARCHES DE L'ENFER)

Pour créer un démon il faut suivre les étapes suivantes:

1. se démoniser (fork);
2. créer une nouvelle session et se déconnecter du terminal (setsid);
3. changer / sécuriser son répertoire de travail (chdir / chroot);
4. ajuster les droits utilisateur (setuid);
5. changer les droits fichiers (umask).

Il est possible de changer (l'ordre de) ces opérations selon ce que l'on veut faire mais il faut prendre garde à la sécurité.

1. DEMONISATION

SE DÉMONISER

La première étape consiste à:

- effectuer un fork;
- laisser le parent mourir;
- continuer les opérations avec l'enfant.

Ce qui permet:

- **que le processus grand-parent ne reste pas bloqué en attende de son enfant (e.g. l'utilisateur peut continuer à utiliser le terminal);**
- que le parent du démon soit systemd (i.e. il peu se terminer sans soucis);
- que le démon ne soit pas le maitre d'un groupe de processus, ce qui est nécessaire pour l'étape 2.

2. GAGNER SON INDÉPENDANCE

SESSION

- une session est crée lors du login d’un utilisateur sur une machine et possède un identifiant SID;
- cette session dure jusqu’a déconnection de l’utilisateur;
- un processus, appelé maitre de la session, est lancé au démarrage d’une session (généralement un shell de login, ou une session graphique);
- les processus descendants font parti de la même session (même identifiant - SID);

CRÉATION D'UNE SESSION

Un processus peut créer une nouvelle session; dans ce cas:

- **il deviens le maitre de cette session;**
- **il ne reçoit plus les signaux du terminal (SIGHUP, SIGTSTP, etc.).**
- **il continue de s'executer même si l'utilisteur termine sa session**

Pour créer une nouvelle session on utilise:

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

DÉCONNECTION DU TERMINAL

Les descripteurs STDIN/STDERR/STDOUT restent hérités du parent il est préférable de les rediriger vers:

- des fichiers de logs si l'on veut conserver ces informations;
- /dev/null pour ne pas utiliser d'entrées/sorties standard.

```
descr = open("/dev/null", O_WRONLY); //open("/var/logfile", O_WRONLY)
dup2(descr, STDIN); //ferme aussi STDIN, STDIN remplacable par STDOUT/STDERR
close(descr);
```

3. RÉPERTOIRE DE TRAVAIL

CHANGER LE CWD

Lorsqu'un processus est lancé sont répertoire de travail correspond au répertoire à partir duquel il a été lancé (pas au répertoire de l'exécutable).

Il est donc préférable de changer le répertoire de travail par un répertoire connu et sure afin d'éviter que:

- des fichiers soient créé par le démon n'importe ou;
- on ne puisse pas démonter le répertoire de travail du démon.

On peut gérer le répertoire de travail avec les fonctions:

```c
#include <unistd.h>
int chdir(const char *path); //changer le répertoire de travail en path
int fchdir(int fd); //changer le répertoire de travail par l'inode fd
char *getcwd(char *buf, size_t size); //répertoire dans buf qui est de taille size
char *get_current_dir_name(void); //effectue un malloc -> penser au free !
```

CHROOT

Les démons sont souvent des portes d'entrée pour des attaques sur la machine. Pour éviter une partie de ces attaques une solution est de **changer la racine «/» vue par le démon**.

Pour cela on utilise:

```c
#include <unistd.h>
int chroot(const char *path);
```

Sachant que:

- après appel de cette fonction la hiérarchie des fichiers vue par le processus démon se limite à `path`, **on ne pourra donc plus utiliser de fichiers en dehors de `path`** (e.g. dans le /etc, /bin, /usr/lib …)
- il faut donc penser à faire les liens nécessaires avant le chroot;
- il faut des droits privilégiés pour appeler chroot (e.g. root).

RÉSUMÉ DES RÉPERTOIRES

Soit `path` un répertoire dédié à notre démon

1. il faut changer le répertoire du démon à `path` (fonction `chdir`)
2. on peu se limiter à ce répertoire en "s'emprisonnant" dans le dossier `path` (fonction `chroot`)

Attention, il est toujours possible de se sortir de la prison par un autre appel à `chroot`.

4. DROITS UTILISATEUR

BESOIN

Il n’est pas rare qu’un démon ai besoin de droits particuliers pour accomplir certaines tâches (e.g. chroot, setrusage, …).

Toutefois il est préférable pour des raisons de sécurité que le démon ne garde pas des privilèges indéfiniment.

Par exemple pour créer une «prison chroot»:

- le chroot précédent n’est pas suffisant pour garantir la sécurité car comme le processus est privilégié il peut toujours sortir du chroot;
- il faut donc que le processus perde ses privilèges (i.e. reprenne des droit utilisateur normaux) pour garantir que les processus restera dans son dossier «prison».

IDENTIFIANTS UTILISATEURS

Un processus Linux possède trois versions des UID (user ID):

- **RUID: real user id, la personne qui lance le programme;**
- **EUID: effective user id, la personne dont le processus a les droits;**
- SUID: saved user id, utilisé pour stocker un uid et le réutiliser plus tard.

Exemple de RUID!= EUID:

- un programme appartient à Pierre mais est exécuté par Jean;
- ce programme active le possesseur comme utilisateur effectif à l'exécution (i.e. chmod u+s);

RUID = Jean; EUID = Pierre; SUID = Pierre.

Nous nous concentrons ici sur les ID utilisateur (UID) mais les ID de groupe se comporte similairement (nom des fonctions différentes).

chmod

Pour qu'un programme s'exécute avec un EUID différent de l'UID il faut **changer le mode du fichier** avec:

```
$ chmod u+s myProgramFile
```

Par exemple:

```
-rwsrwxr-x. 483 root root 43 Nov 14 11:13 euidProg
-rwxrwxr-x. 483 root root 43 Nov 14 11:13 uidProg
```

euidProg sera executé avec:

- euid = **root**
- uid = user

uidProg sera executé avec:

- euid = user
- uid = user;

SETUID / SETEUID

Lors de l'exécution d'un processus, les fonctions permettant de manipuler toutes les formes de UID sont:

```c
#include <unistd.h>
#include <sys/types.h>
uid_t getuid(void); //retourne le ruid
uid_t geteuid(void); //retourne le euid
int setuid(uid_t uid); //modifie l'EUID; c.f. tableau ci-dessous
int seteuid(uid_t euid); //modifie l'EUID; c.f. tableau ci-dessous
int setreuid(uid_t ruid, uid_t euid); //voir man
int setresuid(uid_t ruid, uid_t euid, uid_t suid); //voir man
```

Fonction	EUID = Root ?	EUID = Autre ?
setuid(uid)	RUID = uid EUID = uid SUID = uid	EUID = uid mais fonctionne ssi: uid == RUID ou uid == EUID ou uid == SUID.
seteuid(uid)	EUID = uid	

RÉSUMÉ DES DROITS DU DÉMON

En résumé:

- setuid permet de s'assurer qu'un processus privilégié perds ses droits pour toujours;
- seteuid permet de modifier l'EUID d'un processus mais de reprendre des droits privilègiés (l'UID / SUID) plus tard.

Un démon souhaitant avoir des privilèges puis les relâcher devra donc:

1. être possédé par root (ou un utilisateur avec les privilèges adéquates);
2. être exécuté avec le possesseur comme utilisateur effectif (chmod u+s);
3. effectuer les opérations nécessitants des droits spécifiques;
4. appeler setuid() pour perdre ses droits et travailler en sécurité.

5. DROITS DES FICHIERS

CHANGER LE `umask`

Un processus hérite le masque de création `umask` de son parent. Il est possible que ce masque ne convienne pas pour un démon.

Il faut donc le modifier par exemple à 0.

```c
#include <sys/types.h>
#include <sys/stat.h>
mode_t umask(mode_t mask);
```

c.f. cours fichiers.