Projet – spooler

1 Introduction

Ce projet consiste à réaliser ce qu'on appelle communément un « *spooler* », c'est-à-dire un dispositif qui permet à des utilisateurs de déposer (ou retirer) des « *jobs* » dans une file d'attente (un « *spool* ») ; la file d'attente est examinée par un « *démon* », c'est-à-dire un programme qui ne s'arrête jamais : lorsqu'un nouveau job est trouvé dans la file d'attente, celui-ci est traité, puis supprimé.

Un exemple classique de spooler est le système d'impression : on évite ainsi :

— que l'imprimante soit ouverte simultanément par plusieurs processus (et donc que les impressions soient mélangées sur le papier) puisque seul le démon accède à l'imprimante, matérialisée par un fichier spécial dans `/dev` ;
— l'attente si l'imprimante est déjà en train d'imprimer un gros fichier, puisque le dépôt d'un fichier dans la file d'attente n'est pas bloquant ;
— les problèmes de sécurité puisque seul le propriétaire du processus démon est autorisé à ouvrir le fichier spécial correspondant à l'imprimante.

Sous Unix, par exemple, la commande `lp` ou `lpr` (suivant les systèmes) dépose le fichier à imprimer dans la file d'attente, que le démon (programme `lpd` ou `cupsd` selon les systèmes) surveille. Dès qu'un fichier est déposé dans la file, c'est ce programme démon qui ouvre le périphérique (ou qui accède à l'imprimante par le réseau le cas échéant) et y recopie les données à imprimer.

2 Implémentation

2.1 File d'attente

La file d'attente est traditionnellement matérialisée par un répertoire spécifique dans le système de fichiers (par exemple dans `/var/spool`). Le démon, quand il est désœuvré, explore périodiquement (par exemple toutes les secondes) ce répertoire pour y chercher des fichiers ayant une caractéristique spécifique, comme par exemple ceux dont le nom commence par `j` pour repérer les jobs prêts à être traités.

La partie après le préfixe (« `j` » dans notre exemple) est constituée d'une suite de caractères résultant d'un appel à la fonction de bibliothèque `mkstemp`. On prendra comme convention que cette suite de caractères est l'identificateur du job.

Un job peut être constitué de plusieurs informations : par exemple, un fichier à imprimer est constitué des données elles-mêmes ainsi que de méta-informations, comme par exemple le nom de l'utilisateur ayant demandé l'impression. Dans ce cas, on peut utiliser deux fichiers, par exemple l'un dont le nom commence par `d` pour les données, et l'autre dont le nom commence par `j` pour la description du job.

2.2 Verrouillage

Le dépôt d'un job n'est pas instantané puisqu'il consiste, entre autres, à recopier un fichier. D'autres opérations peuvent également prendre un certain temps. Or, il ne faut pas que le démon et les autres programmes associés prennent en compte des fichiers dans un état transitoire.

Pour éviter cela, on utilisera le verrouillage de fichier grâce à la primitive `lockf`. Lors du démarrage du démon, celui-ci crée si nécessaire un fichier, que vous appellerez « `verrou` ». Par la suite, toute opération pouvant mener à ce que des programmes accèdent à une file d'attente dans un état transitoire doit au préalable verrouiller la file d'attente en utilisant `lockf` sur le fichier verrou. Bien évidemment, le verrouillage doit être aussi court que possible pour éviter que le spool soit inaccessible aux autres programmes.

2.3 Sécurité

Les problèmes de sécurité, s'ils sont simplifiés par la présence d'un processus démon unique disposant de l'identité d'un utilisateur spécifique (non privilégié, mais seul à pouvoir accéder à l'imprimante), ne sont toutefois pas nuls : le dépôt d'un job dans la file d'attente pouvant être demandé par n'importe quel utilisateur, on utilise le mécanisme du bit « *set-user-id-on-exec* » : la commande de dépôt dispose de ce bit, et tout fichier déposé appartient ainsi à l'utilisateur spécifique.

L'« élévation de privilège » rendue possible par ce bit impose de réaliser la commande de dépôt particulièrement soigneusement, en évitant tout bug (de type débordement mémoire ou autre) pouvant affecter la sécurité de l'utilisateur spécifique.

Les autres commandes manipulant le spool pour le compte des utilisateurs non privilégiés doivent également disposer du bit « *set-user-id-on-exec* ».

On notera toutefois que pour accéder aux fichiers d'un utilisateur afin de le déposer dans le spool, il faut disposer des privilèges de cet utilisateur, notamment si l'utilisateur a protégé son fichier comme le fichier `test` dans l'exemple ci-après. Ceci impose de changer d'identité effective (grâce à la primitive `seteuid`) le temps d'ouvrir le fichier à déposer.

3 Travail à réaliser

Le projet consiste à créer un système de spooler tel qu'esquissé ci-dessus.

Le travail du démon sera simplifié par rapport à un spooler d'imprimante : pour chaque job détecté, le démon exécutera une commande déterminée (on utilisera dans ce projet la commande `gzip` vers un fichier temporaire), puis il ajoutera dans un fichier unique la date de traitement, les caractéristiques du job (nom de l'utilisateur ayant déposé le fichier, obtenu par la fonction de bibliothèque `getlogin`, nom et taille du fichier original, date de dépôt dans le spool) ainsi que la taille du fichier compressé.

On vous demande de réaliser les commandes suivantes :

- `demon` [`-d`] [`-f`] [`-i` *délai*] *fichier*
  Ce programme constitue le démon. Il prend en argument l'emplacement du fichier dans lequel le démon enregistre les informations après le traitement de chaque job. Ce fichier doit être remis à 0 à chaque démarrage du démon.
  L'option `-i` spécifie le délai (en secondes) entre deux parcours de la file d'attente, lorsque le démon est désœuvré. L'option `-d` provoque l'affichage d'informations de débogage (à votre convenance) sur la sortie standard.
  Normalement, le démon doit se mettre en arrière plan de lui-même : pour ce faire, le démon génère un processus fils qui fait le travail, le processus père se terminant dès la création du fils. Avec l'option `-f` (pour *foreground*), le démon ne se met pas en arrière plan, ce qui facilite par exemple l'usage d'un débogueur.
- `deposer` *fichier ... fichier*
  Ce programme dépose le ou les fichiers indiqués dans le spool. Pour chaque fichier, son identificateur est affiché sur la sortie standard.
- `lister` [`-l`] [`-u` *utilisateur*]
  Ce programme liste tous les jobs dans le spool en indiquant, pour chacun, l'identificateur du job, le nom de l'utilisateur et la date de dépôt. Vous utiliserez la fonction de bibliothèque `getlogin` pour obtenir le nom de l'utilisateur courant.
  Les options `-l` et `-u` ne sont utilisables que par le propriétaire du spool (vous devez le vérifier) : l'option `-l` provoque en plus l'affichage du nom du fichier, et l'option `-u` permet de limiter l'affichage aux jobs déposés par l'utilisateur spécifié.
- `retirer` *id ... id*
  Ce programme retire du spool le ou les jobs indiqués par leur identificateur, sous réserve que l'utilisateur du job soit le même, ou alors que l'utilisateur réel soit le propriétaire de la file d'attente.

Le répertoire utilisé pour la file d'attente est normalement fixe (via un `#define` commun dans tous les programmes). Pour faciliter les tests, cette définition devra être modifiable à l'exécution en utilisant la variable d'environnement nommée `PROJETSE` (si la variable existe, son contenu remplace la valeur définie par le `#define`).

3.1 Traitement des erreurs

Votre programme devra être exempt d'erreurs de type fuite mémoire, débordement de tableau ou toute autre erreur pouvant affecter la sécurité du système de spool. De même, vous devez rester très attentif au code de retour de toutes les fonctions pouvant échouer.

En cas de problème fatal (rencontré par exemple dans une primitive système ou une fonction de bibliothèque), vous êtes autorisé, si cela alourdit trop votre implémentation, à terminer le programme courant avec un message approprié. Dans ce cas d'arrêt exceptionnel, on ignorera les fuites de mémoire.

3.2 Fonctions à utiliser

De manière générale, vous devrez privilégier les primitives systèmes par rapport aux fonctions de bibliothèque. C'est impératif pour tout ce qui est gestion des processus. Pour la gestion de fichiers, vous pouvez utiliser, si cela s'avère plus confortable, les fonctions de bibliothèque.

Pour analyser les options, vous utiliserez impérativement la fonction de bibliothèque `getopt`.

Note : cet énoncé fait référence à des primitives systèmes (`lockf` et `seteuid`) et à des fonctions de bibliothèque (`mkstemp`, `getlogin`) dont vous trouverez une description dans la notice en ligne (ces fonctions ont été ajoutées par rapport à la version que vous avez eue sous forme papier).

3.3 Exemple

Dans l'exemple ci-après, un utilisateur fait appel aux trois commandes de gestion du spool :

```
turing> ls -l test truc
-rw------- 1 pda users    13 Oct  2 12:36 test   # notez les droits
-rw-r--r-- 1 pda users  1278 Oct  2 14:12 truc

turing> lister                                    # rien dans le spool

turing> deposer test truc                         # test est protégé
3sbrzf
mEB2Gl

turing> lister
3sbrzf pda              Mon Oct  2 14:15:53 2017
mEB2Gl pda              Mon Oct  2 14:15:53 2017

turing> lister -l
lister: Option -l or -u are restricted to spool owner

turing> retirer mEB2Gl

turing> retirer mEB2Gl
Cannot find id mEB2Gl

turing> lister
3sbrzf pda              Mon Oct  2 14:15:53 2017
```

Le fichier produit par le démon suite à ces manipulations est :

```
Starting at Mon Oct  2 14:01:48 2017            # date de démarrage du démon
id=3sbrzf orgdate=Mon Oct  2 14:15:53 2017 user=pda file=truc orgsize=1278
        curdate=Mon Oct  2 14:16:49 2017 gzipsize=574 exit=0
```

3.4 Nécessité d'un deuxième utilisateur

Pour réaliser ce projet, il faut avoir accès à un autre compte utilisateur. Vous pouvez réaliser ce projet en binôme sur turing, ou alors créer un nouvel utilisateur si vous utilisez votre machine personnelle.

3.5 Jeu de tests

Nous vous fournissons, sur moodle, un script de tests qui vérifie que votre projet respecte un certain nombre des spécifications de ce sujet, mais pas toutes. La notation tiendra compte du passage de ces tests. Faites en sorte que votre projet rendu passe ce script avec succès, sans modification du script, et sans erreurs ni affichages superflus.

Vous développerez des tests supplémentaires qui complèteront les tests du script fourni. Vous pouvez pour cela réutiliser le script fourni si vous le souhaitez.

4 Modalités de remise

Le projet est à réaliser par groupes de deux étudiants. Votre projet doit contenir :

- les fichiers sources, y compris un fichier `Makefile` pour générer les programmes susmentionnés et les installer avec les bonnes permissions ;
- un rapport au format PDF contenant une description de votre implémentation, le résultat de jeux de tests, l'analyse de l'absence de fuite mémoire, et la mesure de couverture obtenue avec `gcov`.

Vous déposerez votre projet, débarrassé de la hiérarchie de test et de tout fichier binaire autre que votre rapport, sous forme d'une archive au format *tar.gz* sur Moodle dans l'espace prévu à cet effet, avant le mercredi 8 novembre 2017 à 20 heures.