Projet Periodic : Tâches périodiques
Système et programmation système

Éric MERLET
Université de Franche-Comté

Licence 2 Informatique
2016 – 2017

Le but de ce projet est de réaliser un clone simplifié de `cron(8)` et `at(8)`, permettant de définir des actions à effectuer à intervalle de temps régulier. Le projet est implémenté en langage C.

Spécification

Le projet se compose de trois programmes :
- un lanceur de daemon générique : `launch_daemon` (pouvant éventuellement servir pour lancer d'autres daemons)
- le programme exécuté par le daemon : `period`
- un outil en ligne de commande : `periodic`.

L'outil `periodic` permet de spécifier une action ainsi que la date de départ et sa période. Une période de 0 indique que l'action doit être effectuée une seule fois (à la manière de `at(8)`. La syntaxe est la suivante :

```
./periodic start period cmd [args]
```

où `start` est l'instant de départ, `period` est la période (en secondes) et `cmd` est la commande à exécuter avec ses éventuels arguments (`args`). Les trois premiers paramètres sont obligatoires. L'instant de départ peut être spécifié de différentes manières :
- en indiquant le nombre de secondes depuis Epoch (1er janvier 1970) ;
- en indiquant la chaîne de caractères `now`, pour indiquer l'instant présent ;
- en indiquant le caractère `+` suivi du nombre de secondes avant de lancer la commande.

Quelques exemples :

```
./periodic 1491751378 0 echo Bonjour
# appelle echo à partir du dimanche 9 avril 2017, 17:22:58, une seule fois
./periodic now 1 ls
# appelle ls toutes les secondes à partir de maintenant
./periodic +10 5 uptime
# appelle uptime toutes les 5 secondes en commençant dans 10 secondes
./periodic +5 2 date +%H:%M:%S
# appelle 'date +%H:%M:%S' toutes les 2 secondes en commençant dans 5 secondes
```

Pour communiquer avec le daemon, l'outil `periodic` utilisera un tube nommé `/tmp/period.fifo` et des signaux. Plus précisément, l'outil `periodic` enverra `SIGUSR1` ou `SIGUSR2` de manière à réveiller le daemon puis communiquera via le tube nommé.

Le programme `period` est chargé de l'exécution des différentes commandes. Pour cela, sa tâche principale est de dormir jusqu'à ce qu'une commande doive être exécutée. Il reçoit les commandes à exécuter de l'outil `periodic`. L'ensemble des commandes est placé dans une table. On suppose qu'il ne peut y avoir plus de 1024 commandes dans la table.

Pour obtenir un daemon, il faut lancer le programme `period` en utilisant `launch_daemon` :

```
./launch_daemon  period
```

Outils complémentaires et bibliothèque

Exercice 1 : Préliminaires

Question 1.1 Écrire un programme en C `now` qui donne le nombre de secondes depuis Epoch pour l'instant présent. Indice : `time(2)`.

Question 1.2 Écrire un programme en C `when` qui prend en paramètre un nombre de secondes depuis Epoch et qui renvoie la date à laquelle ce nombre correspond. Indice : `ctime(3)`.

Exercice 2 : Communication via un tube nommé

Le but de cette partie est d'écrire une bibliothèque `libmessage` qui contiendra des fonctions communes aux deux programmes. Ces fonctions sont essentiellement des outils pour communiquer via un tube nommé (associé à un descripteur de fichier). Il faudra faire un en-tête `message.h` contenant les prototypes et un fichier `message.c` contenant les définitions.

Question 2.1 Écrire une fonction `send_string` qui envoie une chaîne de caractères via un descripteur de fichier. Pour cela, on enverra d'abord la taille de la chaîne puis son contenu. La fonction a le prototype suivant :

```
int send_string(int fd, char *str);
```

Question 2.2 Écrire une fonction `recv_string` qui reçoit une chaîne de caractères via un descripteur de fichier. Pour cela, on recevra d'abord la taille de la chaîne, puis on allouera l'espace nécessaire (via `malloc(3)`, attention au caractère '\0' final), puis on lira la chaîne. La fonction a le prototype suivant :

```
char *recv_string(int fd);
```

Question 2.3 Écrire une fonction `send_argv` qui envoie un tableau de chaînes de caractères. On enverra d'abord la taille du tableau puis chacune des chaînes de caractères (le dernier pointeur de ce tableau vaut NULL). La fonction a le prototype suivant.

```
int send_argv(int fd, char *argv[]);
```

Question 2.4 Écrire une fonction `recv_argv` qui reçoit un tableau de chaînes de caractères. On recevra d'abord la taille du tableau puis on allouera l'espace nécessaire puis on lira chacune des chaînes de caractères. La fonction a le prototype suivant :

```
char **recv_argv(int fd);
```

L'outil `periodic`

Exercice 3 : Gestion des paramètres

Question 3.1 Déterminer la date de départ et la période en fonction des arguments. On affichera une aide si une erreur a eu lieu (mauvais nombre d'arguments, arguments erronés). On utilisera `strtol(3)` pour déterminer les nombres passés en paramètre et détecter s'il y a une erreur (par exemple, s'il y a autre chose que des chiffres).

Exercice 4 : Ajout d'une entrée dans la table

Pour ajouter une entrée dans la table du daemon, on va d'abord le prévenir qu'on va lui envoyer des données en lui envoyant un signal, puis on va lui transmettre les informations nécessaires.

Question 4.1 Écrire une fonction qui permet de lire le PID du daemon dans le fichier **texte** `/tmp/period.pid`.

Question 4.2 Envoyer le signal `SIGUSR1` au daemon qui lui indique qu'il va devoir lire des données.

Question 4.3 Envoyer les informations nécessaires (date de départ, période, commande avec ses arguments) via le tube nommé `/tmp/period.fifo` en se servant de la bibliothèque que vous avez écrite.

Exercice 5 : Récupération de la table courante

L'outil doit permettre de récupérer la table courante, c'est-à-dire l'ensemble des entrées de la table.

Question 5.1 Envoyer le signal `SIGUSR2` au daemon qui lui indique qu'il va devoir écrire des données.

Question 5.2 Recevoir l'ensemble des commandes et les afficher.

Le lanceur de daemon `launch_daemon`

Exercice 6 : Implémentation du lanceur de daemon

Question 6.1 Implémenter la méthode du double `fork(2)` pour lancer le daemon. Pour rappel, voici les actions à effectuer :

- Le processus principal forke et attends son fils
- Le fils devient leader de session via `setsid(2)`
- Le fils forke et quitte via `exit(2)`, rendant le petit-fils orphelin
- Le petit-fils n'est pas leader de session, ce qui l'empêche d'être rattaché à un terminal
- Le processus principal termine, laissant le petit-fils être le daemon

Explications supplémentaires :

- Le shell qui lance le processus principal crée un nouveau groupe de processus, dont le leader est le processus principal. Or pour créer une nouvelle session, un processus ne doit pas être leader de son groupe. En effet, la création de la session passe par une étape de constitution d'un nouveau groupe prenant l'identifiant du processus appelant. Il est indispensable que cet identifiant ne soit pas encore attribué à un groupe qui pourrait contenir éventuellement d'autres processus.
- à partir du moment où le fils crée une nouvelle session, tous ses descendants (qui ne créent pas eux mêmes une nouvelle session) appartiennent à cette session.
- Tous les processus d'une session partagent (éventuellement) un même terminal de contrôle. Une session sans terminal de contrôle en acquiert un lorsque le leader de session ouvre un terminal pour la première fois. Ici, le leader de session se termine sans ouvrir de terminal. Donc la session, à laquelle appartient le petit fils, ne pourra plus jamais acquérir un terminal de contrôle.

Question 6.2 Changer la configuration du daemon.

- Changer le répertoire courant à `/` via `chdir(2)`
- Changer le umask à 0 via `umask(2)`
- Fermer tous les descripteurs de fichiers des flux standard

Question 6.3 Charger le nouveau programme à exécuter : `period`

Le programme `period`

Exercice 7 : Initialisations diverses

Question 7.1 Écrire le pid du processus dans le fichier **texte** `/tmp/period.pid`. Si ce fichier existe déjà, le programme doit se terminer.

Question 7.2 Établir une redirection de la sortie standard vers le fichier `/tmp/period.out` et de l'erreur standard vers `/tmp/period.err`.

Question 7.3 S'il n'existe pas, créer le tube nommé `/tmp/period.fifo`.

Question 7.4 S'il n'existe pas, créer le répertoire `/tmp/period`

Exercice 8 : Ajout d'une entrée dans la table

Question 8.1 Quand le signal `SIGUSR1` est reçu, lire la nouvelle entrée envoyé par l'outil `periodic` à travers le tube nommé.

Exercice 9 : Envoi de la table courante

Question 9.1 Quand le signal `SIGUSR2` est reçu, envoyer l'ensemble des entrées de la table à l'outil `periodic` à travers le tube nommé.

Exercice 10 : Gestion des commandes à exécuter

Question 10.1 Définir une structure de données pour chaque entrée de la table.

Question 10.2 Définir une structure pour la table ainsi que les opérations associées : ajout, etc.

Question 10.3 Parcourir la table pour déterminer la prochaine action à effectuer. Attention à bien être sûr que des actions n'ont pas été oubliées.

Question 10.4 Déterminer la durée à attendre avant la prochaine action et prévoir une alarme.

Question 10.5 Se mettre en attente d'un signal.

Question 10.6 Vérifier qu'il y a bien une (ou plusieurs) action(s) à effectuer et les lancer. Pour chaque action, on fera les redirections nécessaires : l'entrée standard sera redirigée depuis `/dev/null` ; la sortie standard et l'erreur standard seront redirigées respectivement vers les fichiers `/tmp/period/X.out` et `/tmp/period/X.err` où `X` est l'indice de la commande dans la table (entre 0 et 1023). Ces deux fichiers seront ouverts en ajout (Pourquoi ?).

Exercice 11 : Gestion des fin de processus

Question 11.1 Gérer le signal `SIGCHLD` qui est envoyé quand un processus fils est terminé, de manière à ne pas créer de zombis (bien vérifier que tous les processus zombies sont éliminés).

Exercice 12 : Gestion de la fin du daemon

Question 12.1 Gérer les signaux `SIGINT`, `SIGQUIT` et `SIGTERM` afin que l'envoi de l'un de ces signaux termine "proprement" le daemon :

- supprimer le fichier `/tmp/period.pid`
- libérer l'espace mémoire alloué dynamiquement
- éventuellement terminer et éliminer tous les processus créés restant

Bonus

Exercice 13 : Retrait d'une commande

Question 13.1 Ajouter la possibilité de retirer une commande de la liste. Pour cela, il faut introduire un code à envoyer au daemon pour distinguer l'ajout d'une entrée dans la table du retrait d'une entrée.

Évaluation

La note du projet tiendra compte des points suivants (liste non-exhaustive) :
- la séparation du projet en plusieurs unités de compilation et bibliothèque(s) ;
- la présence d'un Makefile fonctionnel ;
- les commentaires dans le code source (chaque fonction doit être précédée d'un bloc de commentaires indiquant notamment son rôle et les paramètres attendus) ;
- le contrôle des valeurs de retour des appels système et fonctions ;
- l'absence de fuites mémoire

Vous devrez déposer une archive compressée contenant vos programmes source et un fichier texte README dans le dépôt MOODLE prévu à cet effet avant la date indiquée. Cette archive aura pour nom les deux noms du binôme, mis bout à bout et séparés par le caractère underscore (exemple : «Dupont_Durand.tar.gz»).

Le fichier README doit contenir :
- une conclusion/bilan sur le produit final (ce qui est fait, testé, non fait)
- une présentation des améliorations possibles mais non réalisées
- un bilan par rapport au travail en binôme