SAE 3.03 - Réseau et application serveur

10 janvier 2025

Dominique Colnet et Emmanuel Nataf

Le sujet est composé de deux parties qui peuvent se faire pratiquement de façon indépendante l'une de l'autre. Sur le total des 24 heures allouées à ce projet, il est demandé de consacrer environ 16h sur la partie `mbash` décrite en section 1 et environ 8h sur la partie serveur décrite en section 2. Pour ce projet, vous devez travailler en binôme et ne faire qu'un seul dépôt par binôme.

1 `mbash` : une version miniature de `bash`

L'objectif consiste à écrire une version miniature de `bash`, nommée `mbash` qui permet de lancer interactivement des commandes en utilisant exactement la même syntaxe que `bash`. Étant donné le nombre d'heures que vous allez pouvoir consacrer à ce projet, `mbash` est vraiment une version extrêmement limitée par rapport à `bash`. Le minimum demandé consiste à pouvoir se déplacer avec la commande `cd`, à pouvoir afficher le répertoire courant avec `pwd` et à lancer une commande, comme le fait `bash`, c'est-à-dire en respectant le contenu de la variable `PATH`. Il est également demandé de pouvoir disposer, du caractère `&` qui permet de lancer la commande en arrière plan.

Pour écrire votre version miniature de `bash` vous pouvez procéder de plusieurs façons à condition de respecter les contraintes suivantes :

1. le programme doit être écrit en C sous la forme d'un seul fichier compilable `mbash.c`, pas trop gros,
2. mettre tout ce qu'il faut dans l'unique fichier `mbash.c` sans oublier de préciser en commentaire la ligne de commande pour compiler votre programme si vous utilisez des bibliothèques,
3. comme vous pouvez vous en douter, il est interdit d'appeler le véritable `bash` mais, si vous voulez, vous pouvez même utiliser `system` pour vous simplifier les choses, si vous optez pour cette solution, vous avez une très grande partie de ce qui est demandé qui est déjà réalisé, à vous de vérifier ce qui marche et ce qui ne marche pas,
4. si, plutôt que d'utiliser `system`, vous décidez d'utiliser `execve`, et comme c'est plus difficile, la programmation du minimum demandé vous permettra également d'avoir une très bonne note,
5. enfin, vous pouvez aussi opter pour une solution intermédiaire en utilisant par exemple `execlp`, `execvp` ou `execpe`, dans le but d'orienter votre travail vers un autre aspect du `bash`.

En plus du fichier compilable `mbash.c`, donnez un petit résumé de ce que votre version est ou non capable de faire, ce qui marche et ce qui ne marche pas. Il est demandé de faire en sorte que le `cd` marche, mais par exemple, si la variable `$$` ou `$?` ne marche pas, ce n'est pas très important. Inversement, si vous avez fait cette amélioration, pensez à bien l'indiquer.

Rappelons que les notations (ou fonctionnalités) doivent être exactement les mêmes que celles de `bash`. A priori, il n'est pas demandé de se préoccuper de la programmation des raccourcis clavier du `bash` comme par exemple le `Ctrl-R`. Ceci étant, si c'est cet aspect qui vous intéresse, pourquoi pas, à condition de mettre tout ce qu'il faut dans l'unique fichier `mbash.c`. De manière générale, dans le but d'avoir vraiment un travail propre à chaque binôme, vous pouvez travailler sur les aspects que vous préférez, comme par exemple le paramétrage du prompt via la variable `PS1` ou encore le built-in `history` par exemple. Autre exemple : la prise en compte du `Ctrl-D` pour sortir de `mbash` et/ou la prise en compte du built-in `exit`. Si cela vous intéresse, vous pouvez aussi décider d'implanter les variables (`$MACHIN export = ...`) ou encore la substitution des caractères spéciaux (`* ? [a-z] ...`). Ou encore, la prise en compte du `if-then-elif-fi` ou du `while`, ou d'une autre instruction de

votre choix. Comprenez bien qu'il n'est pas demandé de tout faire car c'est un travail bien trop important pour ce projet que de reprogrammer complètement `bash`. Il est donc interdit d'en faire trop et si ce que vous avez choisi n'est fait par aucun autre groupe, c'est encore mieux.

Concernant l'analyse de la ligne de commande, sauf si vous pensez que ce n'est pas la bonne solution, il est conseillé d'utiliser un ou plusieurs automates. Si vous avez par exemple décidé de gérer les variables et aussi l'expansion des caractères magiques (`* ? [a-z] ...`), il est par exemple possible de faire passer un premier automate pour les variables seulement. Le deuxième automate peut ensuite se charger des caractères spéciaux. Bien entendu, c'est juste un exemple d'utilisation d'automates dans le cas ou vous avez décidé de prendre en compte les variables et les caractères spéciaux.

Le `bash` comporte tellement d'aspects qu'il est *a priori* très peu probable que deux groupes choisissent exactement les mêmes. Pour continuer la liste des aspects que vous pouvez imaginer prendre en compte : le point virgule pour séparer deux commandes, le `wait`, le pipe, `&&`, `||`, etc., etc., etc.

Pour ceux qui ne savent vraiment pas par où commencer, on donne le programme `mbash.c`, mais c'est juste pour vous donner un tout petit point de départ. Pensez à bien précisez ce que vous avez choisi de prendre en compte, par exemple, en commentaire dans le fichier `mbash.c`.

2 Serveur de package Debian

Cette partie est complémentaire de la précédente, mais elle peut être faite indépendamment.

2.1 Utilisation

On veut que depuis votre machine (virtuelle ou personnelle, sous linux), on puisse taper :

```
sudo apt install mbash
```

pour que le programme `mbash` s'installe sur la machine et soit utilisable.

Pour cela, il faut que votre machine soit une source de dépôt et que vous ayez intégré votre programme `mbash` dans un package Debian (avec un nom et un numéro de version).

Vous pouvez réaliser cela sur une unique machine, mais il serait plus réaliste qu'une machine du binôme soit le serveur de package et que l'autre soit celle d'un utilisateur client (l'un n'empêchant pas l'autre).

2.2 Réalisation

2.2.1 Partie serveur

Il faut créer un dépôt debian (outils utilisés : dpkg-sig et reprepro). Il contiendra le programme `mbash` exécutable, dans un emplacement permettant son utilisation facilement (dans `/usr/bin` par exemple). Les packages doivent être signés avec une clé que vous devrez générer. Cette clé sera composé en fait de deux clés : la privée et la publique. La privée est pour le serveur de package. Pour être accessible, le serveur de package utilise un autre protocole, habituellement HTTP, ce qui nécessite la présence d'un serveur HTTP, comme apache2.

2.2.2 Partie client

Il faut configurer son système pour que le serveur soit dans la liste des serveurs de packages qui sont utilisés par la commande `apt`. Il faut également récupérer la clé publique pour authentifier le package `mbash`.

2.2.3 Cycle de vie

Un logiciel ne cesse d'évoluer, votre `mbash` démarre avec le numéro de version 0.1. Voici un scénario habituel :

1. Vous installez `mbash` : `sudo apt install mbash`
2. Les développeurs de `mbash` produisent une nouvelle version, la 0.2 par exemple
3. Vous la mettez à jour votre base locale de version avec `sudo apt update`
4. et comme il y a une nouvelle version : `sudo apt upgrade` vous installe la dernière version

Tout ceci doit fonctionner, si vous avez bien fait les parties serveur et client