Mini projet

Module : Théorie des systèmes d'exploitation & SE Windows/Unix/Linux

1. Contexte :

Dans le cadre de l'automatisation des processus informatiques, l'exécution de suites de commandes de base ou d'outils Unix/Linux standard est courante. Automatiser l'installation, la configuration et l'exécution de ces traitements peut optimiser le temps et l'effort, minimiser les erreurs de saisie, les mauvaises implémentations d'étapes ou les défaillances dans la préparation des données nécessaires.

2. Objectif :

Développer un **script Bash** qui répond à un besoin spécifique clairement identifié. Ce besoin doit être décrit en détail dans un compte rendu associé au projet. Une présentation concise du script et de son application sera également programmée. Toutes les étapes et livrables doivent se conformer strictement aux instructions et directives énoncées dans ce document.

3. Instructions :

3.1. Portée du programme :

Votre script devra s'appuyer sur les connaissances acquises dans le module "**Théorie des systèmes d'exploitation & SE Windows/Unix/Linux**", accessible via Google Classroom. Vous êtes également encouragés à intégrer les concepts et techniques abordés dans le cours «**Linux Unhatched Cisco**» disponible sur la plateforme NETACAD, à laquelle vous avez préalablement reçu un accès.

3.2. Directives à respecter:

3.2.1. Identification et description du besoin :

Le besoin à automatiser doit refléter un nouveau besoin réel des utilisateurs (développeurs, administrateurs système, etc.) ou une amélioration significative d'un traitement existant, qui aura un impact considérable sur le temps, l'effort, et la qualité du résultat final.

3.2.2. Développement des traitements:

- Votre script doit incorporer des commandes ou des outils Unix/Linux de base. Il peut également faire appel à des scripts externes développés en Bash, en langage C, etc.
- Il doit prendre en charge une ou plusieurs données en paramètre, avec au moins une donnée obligatoire.
- Le script doit aussi proposer, au moins, six options obligatoires, telles que :
  - **-h** (help): Affiche une documentation détaillée du programme.
  - **-f** (fork): Permet une exécution par création de sous-processus avec fork.
  - **-t** (thread): Permet une exécution par threads.
  - **-s** (subshell): Exécute le programme dans un sous-shell.

- **-l** (log): Permet de spécifier un répertoire pour le stockage du fichier de journalisation.
- **-r** (restore): Réinitialise les paramètres par défaut, utilisable uniquement par des administrateurs.
- …

- L'activation de certaines options doit nécessiter des privilèges administrateurs.
- Implémentez des concepts de base du shell Linux vus en cours ou en TP, comme les conditions, boucles, fonctions, variables d'environnement, expressions régulières, fonction de manipulation des fichiers, Recherche/archivage/compression des fichiers, contrôle d'accès, pipes et filtres.
- Les sorties standard et d'erreur de votre script doivent être gérées de manière spécifique: elles doivent être redirigées simultanément vers le terminal et vers un fichier de journalisation nommé **history.log**, situé dans le répertoire **/var/log/yourprogramname**. Chaque entrée dans ce fichier doit être précédée de la date et de l'heure au format **yyyy-mm-dd-hh-mm-ss**, suivies du nom de l'utilisateur connecté et du type de message (**INFOS** pour les messages de sortie standard et **ERROR** pour les messages d'erreur). Par exemple:

```
yyyy-mm-dd-hh-mm-ss : username : INFOS : message de l'output standard
yyyy-mm-dd-hh-mm-ss : username : ERROR : message de l'erreur standard
```

3.2.3. Gestion d'erreur :

Le programme doit activement gérer les erreurs résultant d'une utilisation incorrecte, telles que le nombre inapproprié d'options ou l'échec d'un traitement exécuté par le script. Pour chaque type d'erreur, un code spécifique doit être attribué afin de faciliter l'identification et la résolution des problèmes. Voici quelques exemples de codes d'erreur :

- 100 : Option saisie non existante
- 101 : Paramètre obligatoire manquant
- …

En outre, le script doit afficher, après chaque erreur déclenchée, un message d'aide explicatif contenant la documentation détaillée du programme, similaire au message généré par l'option -h précédemment implémentée.

3.2.4. Exécution et test :

La syntaxe d'exécution du programme doit respecter la structure d'une commande de base Linux, à savoir : **programname [options] [paramètre]**

La démonstration doit mettre en évidence les différentes fonctionnalités implémentées tout en spécifiant la directive à laquelle chaque fonctionnalité fait référence. Le test à exécuter doit inclure au moins trois scénarios différents, permettant d'évaluer le programme lors de traitements léger, moyen et lourd, afin de tester l'efficacité de la création des processus fils via subshell, fork et thread.

3.2.5. *Documentation :*

Une version simplifiée de la documentation, conforme aux standards Linux, doit être incluse dans le programme et accessible via l'option **-h** pour affichage dans le terminal. Le compte rendu à fournir doit représenter la version étendue de cette documentation, incluant des

captures d'écran et des détails sur les spécifications des directives implémentées. Il devra également présenter des exemples concrets d'utilisation du programme dans différents contextes : exécution normale, via subshell, fork/thread.

4. Critères de Notation :

La qualité, l'originalité et la pertinence du besoin identifié, ainsi que la clarté du programme, sa documentation et le respect des directives seront évalués.

5. Directives de Soumission :

Soumettez votre projet dans un document compressé zip que vous allez renommer selon ce format : **TeamID-devoir-shell.zip**. Ce dernier doit contenir les fichiers suivants :

- **Un compte rendu** sous format PDF nommé **TeamID-devoir-shell.pdf**
- **Une présentation** PowerPoint nommée **TeamID-devoir-shell.pptx**, composée d'une seule diapositive résumant l'ensemble du projet, sera présentée en **180 secondes**[1], suivie d'une démonstration n'excédant pas **5 minutes**.
- **Un script shell principal** dont le nom correspond aux fonctionnalités proposées par votre programme. Ce nom peut être un mot du dictionnaire ou une abréviation de plusieurs mots. Des scripts complémentaires, qu'ils soient en shell ou en langage C, peuvent également être inclus s'ils sont essentiels à la bonne exécution de votre programme.

Date Limite de Soumission : **01/06/2025 23:59:59**

Bonne chance !

[1] Ce concept de présentation s'inspire de Three minute thesis (3MT®) : https://threeminutethesis.org/