TP de Synthèse– Ensea in the Shell

L'objectif de cette phase était de créer la structure de base d'un shell fonctionnel en C, capable d'exécuter des commandes simples (REPL) et de se terminer proprement. Pour cela, nous avons dû utiliser exclusivement les appels systèmes de bas niveau (write, read, fork, exec*, wait).

1. Affichage d'un message d'accueil, suivi d'un prompt simple.

On a réalisé l'affichage du message d'accueil et du prompt initial (enseash %) qui a été implémenté en utilisant l'appel système write().

Nous avons utilisé le descripteur de fichier STDOUT_FILENO (valeur 1) pour diriger les messages vers la sortie standard du terminal. Ceci est crucial pour le bon fonctionnement d'un shell, car STDOUT_FILENO est le canal attendu pour les sorties non-critiques (comme les résultats de ls ou le prompt lui-même).

Le code C utilise write(STDOUT_FILENO, MESSAGE, strlen(MESSAGE)) pour garantir que seuls les octets nécessaires soient transmis au noyau, sans utiliser la librairie standard C.

2. Exécution de la commande saisie et retour au prompt (REPL : read–eval–print loop) :

```
ensea@StudentLab:~$ gcc enseash.c -o enseash
ensea@StudentLab:~$ ./enseash
Bienvenue dans le Shell ENSEA.
Pour quitter taper 'exit'.
enseash % ls
 Desktop     enseash.c.save                          my_cp.c    tdm.c
 Documents   enseash.c.save.1                        pcb        Templates
 Downloads   Fichiers_de_simulation_CORE-20240923    pcb.c      Videos
 enseash    'Fichiers_de_simulation_CORE-20240923(1)' Pictures
 enseash.c   Music                                   Public
enseash % pwd
/home/ensea
enseash % ceci
execlp error:command not found or execution error: No such file or directory
enseash %
```

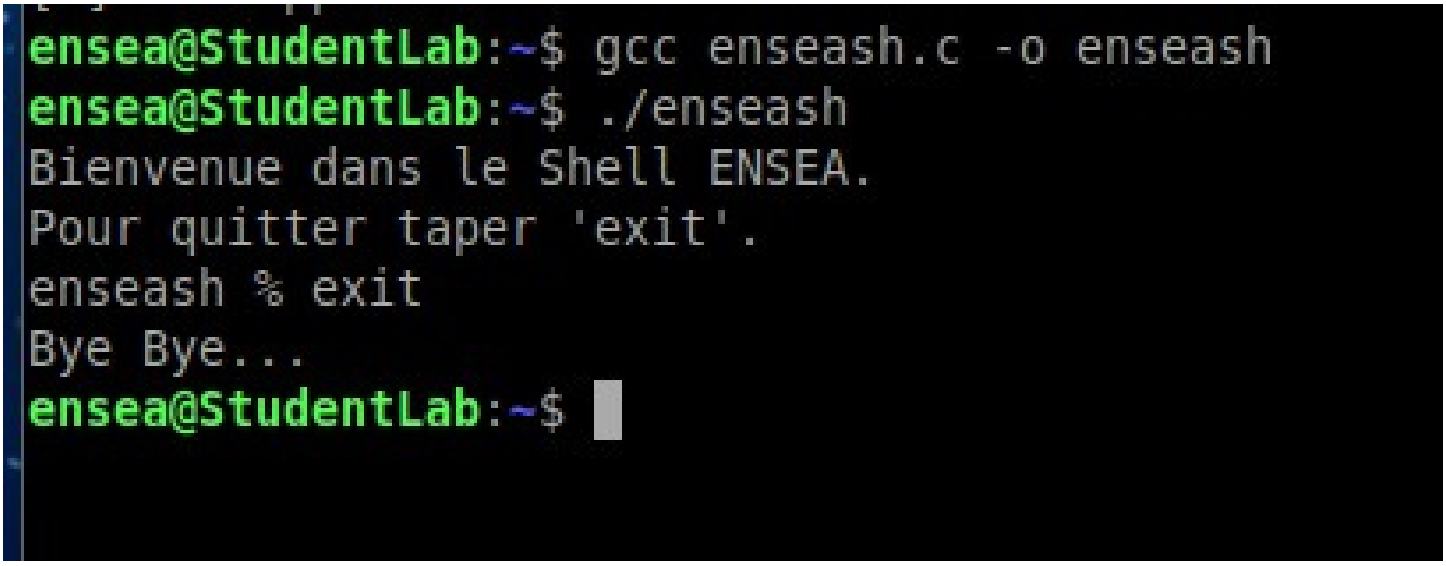
Cette section met en œuvre le cycle fondamental du shell (REPL):

1. Lecture (2a) : L'appel read(STDIN_FILENO, command_line, ...) capture l'entrée de l'utilisateur. Le caractère de fin de ligne (\n) est ensuite remplacé par le caractère nul (\0) pour préparer la chaîne.

2. Exécution (2b) : La combinaison fork() et execlp() est utilisée. Le processus père appelle fork() puis wait() pour attendre la terminaison. Le processus fils utilise execlp(cmd, cmd, NULL) pour exécuter la commande simple demandée.
3. Boucle (2c) : L'ensemble du processus est contenu dans une boucle while(1), permettant le retour immédiat au prompt après que le processus père ait terminé son wait().

→ La **Figure 2** montre l'exécution de commandes simples (ls, pwd) et un test d'échec. L'affichage des résultats de ls et pwd sur le terminal est rendu possible car le processus fils a hérité du descripteur STDOUT_FILENO du père et l'utilise pour sa propre sortie

3. Gestion de la sortie du shell avec la commande “exit” ou un ctrl+d;



Nous avons géré la sortie du shell comme suit:

1. Commande exit : La commande interne est détectée par strcmp. Si elle est saisie, la boucle while(1) est interrompue (break).
2. <ctrl>+d : Le signal de fin de fichier (EOF) est détecté lorsque l'appel read() retourne 0. Cette condition interrompt également la boucle.

Dans les deux cas, juste avant la terminaison du programme, le message "Bye bye..." est affiché par write(STDOUT_FILENO, BYE_MSG, ...).

→ La **Figure 2** confirme visuellement que le shell gère correctement la commande exit: le message "Bye Bye..." est affiché, et le contrôle est rendu au shell parent (Bash).