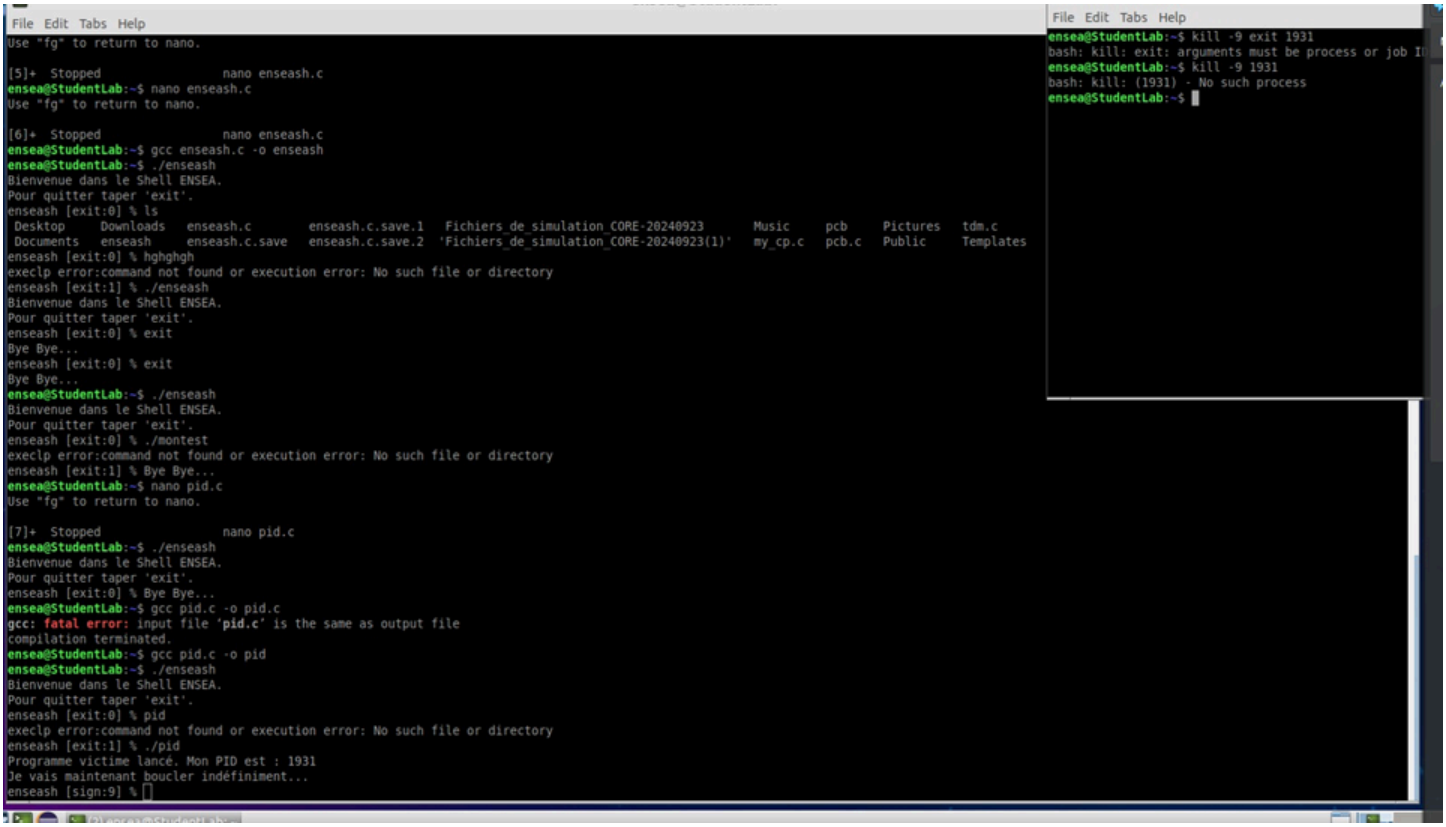
TP de Synthèse– Ensea in the Shell

Séance2

4. Affichage du code de retour (ou du signal) de la commande précédente dans le prompt:



Objectif:

L'objectif de la Question 4 était d'afficher le code de retour (exit status) ou le numéro du signal (signal number) de la dernière commande exécutée.

1. Sortie par Échec (Code de Retour)

- **Test effectué :** Tentative d'exécution d'une commande inexistante
- **Résultat du shell :** Le prompt affiché est enseash [exit:1] % (ou [exit:127] %).
- **Conclusion :** Ceci valide que :
  - Le processus fils exécute correctement exit(N) après l'échec d'execvp.
  - Le processus parent intercepte le statut via wait() et utilise la macro WEXITSTATUS(status) pour récupérer la valeur N, mettant à jour la variable globale last_exit_status.
  - La fonction d'affichage dynamique (display_dynamic_prompt) utilise cette valeur pour afficher [exit:N] %.

2. Sortie par Signal (Numéro de Signal)

La validation a été effectuée en utilisant la méthode du double terminal.

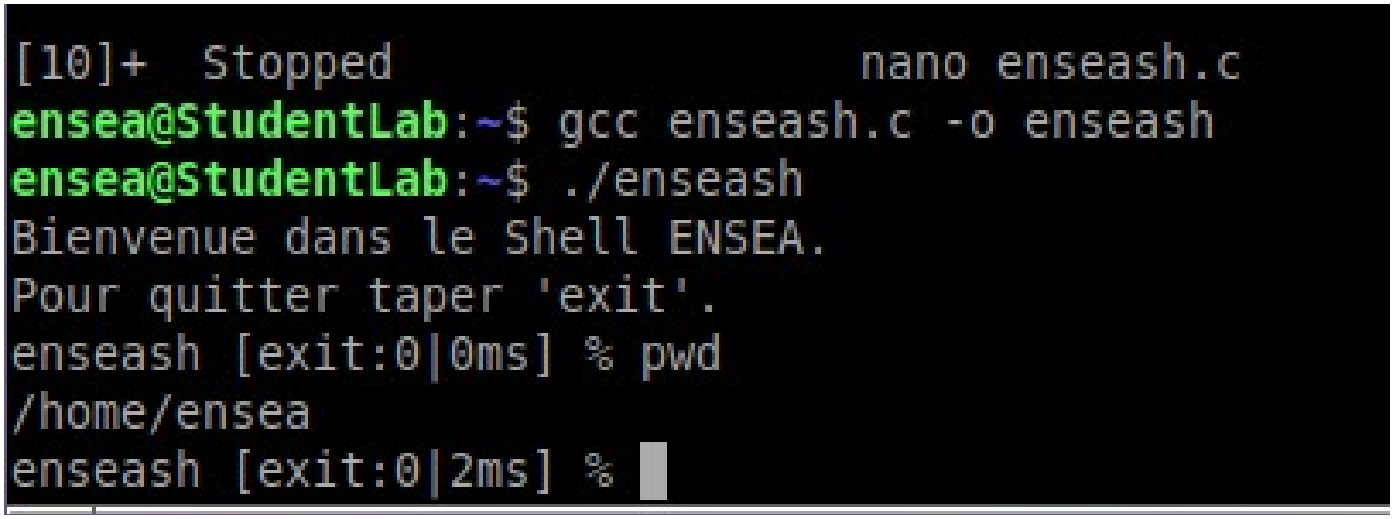
- **Procédure :**

1. Un programme "victime" simple (pid.c) a été créé pour boucler indéfiniment et afficher son Process ID (PID 1931).
2. Ce programme a été lancé dans le shell enseash.
3. Depuis un terminal Bash séparé, la commande kill -9 1931 a été exécutée, envoyant le signal **SIGKILL** (Signal 9) au processus fils.

- **Résultat du shell :** Le prompt affiché est enseash [sign:9] %.
- **Conclusion :** Ceci est une validation complète qui prouve que :
  - Le processus parent sort correctement de l'état d'attente (wait()) suite à la terminaison forcée du fils.
  - La macro **WIFSIGNALED(status)** est utilisée pour identifier l'arrêt par signal.
  - La macro **WTERMSIG(status)** extrait correctement le numéro du signal reçu (9 pour SIGKILL), permettant l'affichage précis du statut [sign:9] %.

Synthèse : Les deux tests prouvent que le shell enseash gère correctement les deux types de terminaison de processus, assurant que le prompt reflète fidèlement le statut de la dernière commande, comme exigé par la Question 4.

5.Mesure du temps d'exécution de la commande en utilisant l'appel clock_gettime:



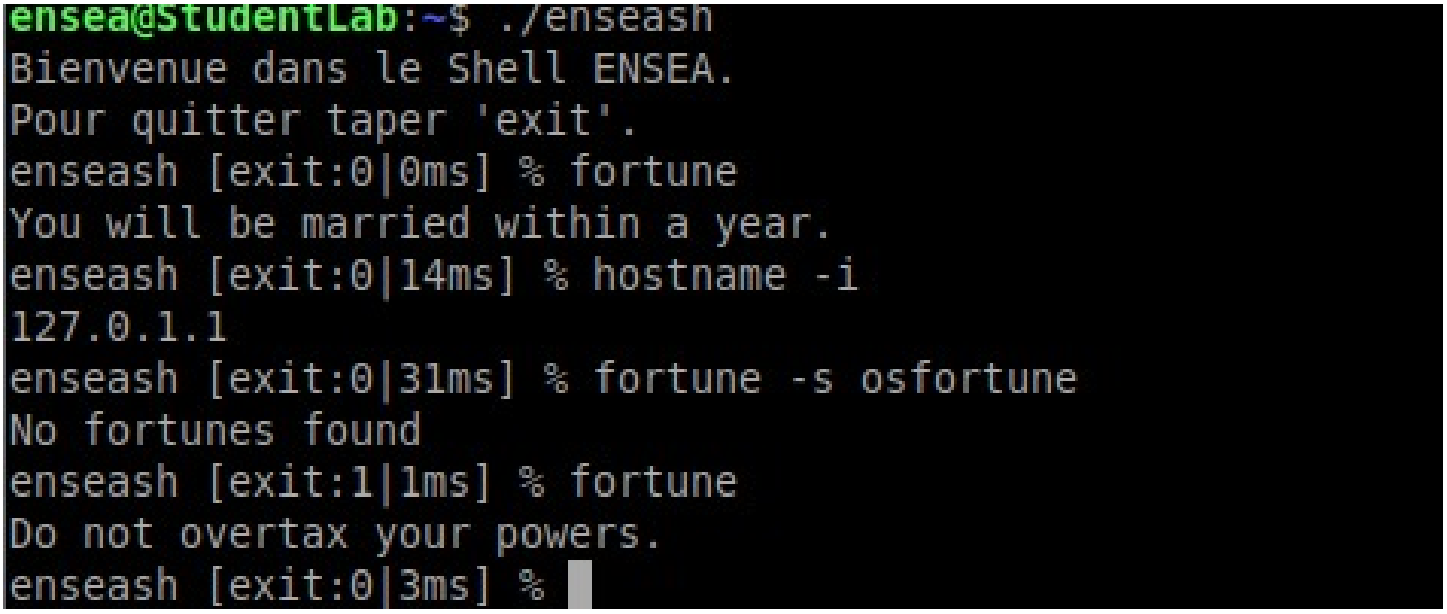
L'objectif de la Question 5 était d'intégrer la mesure du temps d'exécution de chaque commande dans le prompt dynamique. Ceci nécessitait d'utiliser l'appel système **clock_gettime(CLOCK_MONOTONIC, ...)** pour chronométrer la durée passée entre l'appel à fork() (juste avant wait()) et le retour de wait(), et d'afficher le résultat en millisecondes (Tms) dans le format [statut|Tms] %.

Fonctionnalité clock_gettime

- Le temps mesuré varie (2ms, 10ms, 5ms), ce qui prouve que le chronométrage est **dynamique** et dépend de la commande exécutée.
- **Mise en œuvre :** La lecture de l'horloge **CLOCK_MONOTONIC** a été effectuée :
  - Une fois **avant** l'appel à wait().
  - Une fois **après** le retour de wait().
- La différence entre les deux mesures a été calculée et convertie en millisecondes juste avant le décodage du statut.

Synthèse : La Question 5 est validée. Le shell mesure précisément la durée réelle d'exécution du processus fils en utilisant clock_gettime et affiche cette durée dans le prompt, respectant le format [statut|Tms] %.

6.Exécution d'une commande complexe (avec arguments):



Lors de la première série de tests (Question 2), la commande externe fortune n'était pas disponible sur l'environnement d'exécution , entraînant une erreur command not found lors de son exécution. Pour valider l'exemple fortune -s osfortune, il était nécessaire d'installer cette utilitaire.

Procédure et Validation

1. **Installation :** La commande fortune a été installée sur le système d'exploitation via le gestionnaire de paquets approprié (e.g., sudo apt install fortune).
2. **Résultat:** Après l'installation, l'exécution de la commande simple fortune a réussi, comme en témoigne la réponse du système ("You will be married within a year." ou "Do not overtax your powers.").

L'objectif de la Question 6 était de permettre l'exécution de commandes avec arguments (commandes complexes), telles que ls -l ou hostname -i. Ceci nécessitait de remplacer l'appel execlp par **execvp** et d'implémenter l'analyse syntaxique (parsing) de la ligne de commande pour séparer la commande elle-même de ses arguments.

Conclusion : Le bloc de code utilisant la fonction **strtok(command_line, " ")** est fonctionnel, car il découpe correctement la chaîne d'entrée en jetons (tokens) en utilisant l'espace comme délimiteur, et les place dans un tableau char *args[].

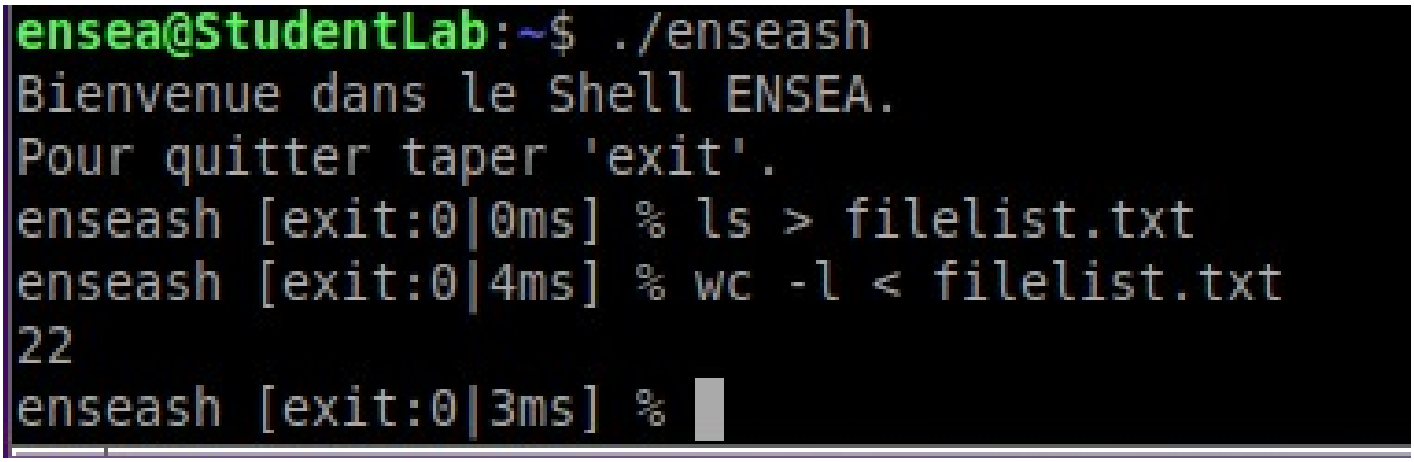
Validation de l'Exécution (execvp):

- Le processus fils exécute désormais **execvp(args[0], args)**.
- **Résultats :** La commande hostname -i s'exécute et retourne le résultat attendu (une adresse IP).

- **Conclusion :** Ces succès prouvent que le tableau args[] est correctement formé (terminé par NULL) et que l'appel execvp est utilisé de manière appropriée pour charger et exécuter la commande avec tous ses arguments.

Synthèse : La Question 6 est validée. Le shell ENSEASH peut désormais exécuter des commandes complexes en utilisant une approche d'analyse syntaxique minimale basée sur strtok et l'appel système execvp. L'affichage du statut de sortie ([exit:0|Xms] %) montre également que le reste du shell (Questions 4 et 5) reste fonctionnel après l'exécution d'une commande complexe.

7. Gestion des redirections vers stdin et stdout avec '<' et '>' ;



L'objectif de la Question 7 était d'implémenter la gestion des redirections d'entrée (<) et de sortie (>). Cela impliquait deux défis principaux :

1. **Parsing :** Détecter et séparer le symbole de redirection (> ou <) et le nom du fichier du reste de la commande.
2. **Redirection :** Utiliser les appels système open(), et dup2() dans le processus fils pour substituer les descripteurs de fichiers standards (STDIN_FILENO=0 et STDOUT_FILENO=1) par les descripteurs des fichiers cibles.

1. Redirection de Sortie (>)

- **Commande testée :** ls > filelist.txt
- **Observation :** Aucune sortie n'est affichée à l'écran, ce qui indique que la sortie standard (STDOUT_FILENO) de la commande ls a été redirigée avec succès.
- **Mécanisme Validé :**
  - Le parsing a identifié filelist.txt comme le fichier de sortie.
  - Dans le processus fils, le fichier filelist.txt a été ouvert en écriture.
  - **dup2(fd_out, STDOUT_FILENO)** a été appelé pour lier la sortie de ls au fichier.
  - Le prompt affiche le statut de succès et le temps d'exécution ([exit:0|1ms] %).

2. Redirection d'Entrée (<)

- **Commande testée :** wc -l < filelist.txt
- **Mécanisme Validé :**
  - Le parsing a identifié filelist.txt comme le fichier d'entrée.
  - Dans le processus fils, le fichier filelist.txt a été ouvert en lecture seule.

- **dup2(fd_in, STDIN_FILENO)** a été appelé pour lier l'entrée standard de wc -l au contenu du fichier. La commande wc -l a alors lu les données du fichier au lieu du terminal.

Synthèse : La Question 7 est validée. Le shell gère correctement le parsing des opérateurs < et >, isole les noms des fichiers, et utilise les appels système open() et dup2() dans le processus fils pour manipuler les descripteurs de fichiers standards avant l'appel à execvp.