**TP noté n°2 2025 : Système d'exploitation**
**Durée : 2h30**
**Barème : Indicatif**
**Enseignante : Fetia Bannour**

# Exercice 1 (8 points)

Cet exercice a pour objectif d'évaluer votre compréhension des mécanismes de communication inter-processus via des pipes nommés, des flux et de la gestion des signaux.

1. Écrivez un programme **serveur** qui :
   (a) Crée deux pipes nommés : `fifo_to_server` et `fifo_to_client`.
   (b) Ouvre `fifo_to_server` en lecture bloquante et attend les messages d'un client.
   (c) Lit un message envoyé par un client et l'affiche dans la console.
   (d) Répond au client avec un message via `fifo_to_client`.
   (e) Gère le signal `SIGINT` pour supprimer les pipes avant de se terminer.
2. Écrivez un programme **client** qui :
   (a) Ouvre `fifo_to_server` en écriture et `fifo_to_client` en lecture.
   (b) Permet à l'utilisateur de saisir un message et l'envoie au serveur.
   (c) Affiche la réponse reçue du serveur.
   (d) Gère le signal reçu dans 5. pour afficher un message clair si le serveur est fermé.
3. Compilez les programmeset lancez le serveur dans un terminal et le client dans un autre.
4. Que se passe-t-il si le **serveur** est lancé sans client ? Expliquez l'état du processus.
5. Que se passe-t-il si le **client** tente d'écrire dans le pipe sans que le serveur soit lancé ? Quel est le signal reçu ? et au niveau de quel processus ? Rappelez le comportement par défaut de ce signal. Précisez également ce que retourne la fonction write() ainsi que la valeur de errno dans ces conditions.
6. Proposez une solution pour attraper le signal reçu dans la question 5. et affichez le message "signal attrapé" avant de terminer le processus. Rajoutez le code associé dans le programme concerné.

# Exercice 2 **(12 points)**

Réalisez un programme en C qui utilise deux processus fils pour effectuer une recherche parallèle de mots-clés dans les lignes d'un fichier texte. Chaque fils a un rôle spécifique et transmet son résultat au processus père via un seul pipe. Le père réagit également à la fin d'un fils (provoquée par l'utilisateur via une commande shell) en arrêtant immédiatement l'autre fils.

(a) **Rôle des processus fils :**

- Le programme utilise deux processus fils créés par le processus père :
  - Le **fils 1** recherche des mots-clés liés aux erreurs (par exemple, "error", "warning") et compte combien de lignes contiennent ces mots.
  - Le **fils 2** recherche des mots-clés liés au succès (par exemple, "success", "completed") et compte combien de lignes contiennent ces mots.

(b) **Lecture et distribution des lignes :**

- Le processus père lit un fichier texte passé en argument, ligne par ligne.
- Chaque ligne est envoyée simultanément aux deux fils :
  - **pipe_f1** : pour transmettre les lignes au fils 1.
  - **pipe_f2** : pour transmettre les lignes au fils 2.

(c) **Transmission des résultats :**

- Les deux fils envoient leurs résultats au père via un **seul pipe partagé** nommé **result_pipe**.
- Les fils écrivent leurs résultats sous forme de messages préfixés avec un identifiant :
  - Le **fils 1** écrit un message au format : `1:<nombre>` (par exemple, `1:5` pour 5 lignes contenant des erreurs).
  - Le **fils 2** écrit un message au format : `2:<nombre>` (par exemple, `2:3` pour 3 lignes contenant des succès).

(d) **Gestion des signaux :**

- Le père doit détecter la fin d'un fils grâce au signal **SIGCHLD**.
- Si un fils est terminé par l'utilisateur via une commande shell (`kill -SIGTERM <PID>`), le père doit immédiatement arrêter l'autre fils en lui envoyant un signal **SIGTERM**.
- Une fois les deux fils terminés, le père récupère les résultats depuis le pipe partagé et affiche les statistiques finales.

(e) **Affichage des résultats :**

- Une fois les deux fils terminés, le père affiche :

  - Le nombre total de lignes contenant des mots-clés d'erreur (résultat du fils 1).
  - Le nombre total de lignes contenant des mots-clés de succès (résultat du fils 2).
- Si un fils est interrompu avant la fin de son traitement, son résultat est considéré comme nul.

# Exemple d'exécution

## Fichier d'entrée

Un fichier `data.txt` contenant :

```
Operation completed successfully
Error: file not found
Warning: low memory
Process completed
Success: data saved
No issues detected
```

## Commande d'exécution

```
$ ./keyword_search data.txt
```

## Sortie intermédiaire

```
Fils 1 lancé avec PID 12345 pour rechercher les mots-clés d'erreur.
Fils 2 lancé avec PID 12346 pour rechercher les mots-clés de succès.
```

## Commande utilisateur (dans un autre terminal)

L'utilisateur décide de tuer le fils 1 : `$ kill -SIGTERM 12345`

## Sortie finale

```
Fils 1 (PID 12345) s'est terminé (signal SIGTERM).
Fils 2 (PID 12346) a été arrêté par le père (signal SIGTERM).
Nombre de lignes contenant des mots-clés d'erreur : 0 (fils interrompu).
Nombre de lignes contenant des mots-clés de succès : 3.
```

# Travail demandé

(a) **Création des processus fils :**

- Le père crée deux processus fils spécialisés :
  - Le **fils 1** traite les lignes pour rechercher les mots-clés d'erreur.
  - Le **fils 2** traite les lignes pour rechercher les mots-clés de succès.

(b) **Gestion des pipes :**

- Le père crée trois pipes :
  - **pipe_f1** : pour envoyer les lignes au fils 1.
  - **pipe_f2** : pour envoyer les lignes au fils 2.
  - **result_pipe** : pour recevoir les résultats des deux fils.

(c) **Traitement des fils :**

- Chaque fils analyse les lignes reçues via son pipe dédié (**pipe_f1** ou **pipe_f2**) et compte combien de lignes contiennent ses mots-clés.
- À la fin de son traitement, chaque fils écrit un message dans **result_pipe**.

(d) **Réaction à la mort d'un fils :**

- Si un fils se termine (par une commande shell ou naturellement), le père détecte l'événement grâce à **SIGCHLD**.
- Le père identifie quel fils s'est terminé, tue immédiatement l'autre fils, et traite les résultats.

(e) **Affichage des résultats :**

- Une fois les deux fils terminés, le père affiche les statistiques finales.