Mini-Shell (`mysh`)

Description du Projet

`mysh` est un mini-shell interactif implémenté en C, offrant une interface utilisateur simple pour exécuter des commandes système, gérer des tâches en arrière-plan, manipuler des variables locales et d'environnement, et gérer des redirections et pipelines. Ce projet illustre le fonctionnement d'un interpréteur de commandes Unix personnalisé.

Fonctionnalités

- **Commandes système :**
    - Exécution de commandes système standard (ex. `ls`, `cat`, `grep`).
    - Gestion des pipelines (`|`) pour enchaîner plusieurs commandes.
    - Redirection d'entrée/sortie (`>`, `<`, `2>`) avec support pour l'ajout (`>>`).
- **Commandes internes :**
    - `cd`: Changement de répertoire.
    - `exit`: Quitter le shell.
    - `status`: Afficher le statut de la dernière commande exécutée.
    - `myjobs`: Afficher les tâches en cours en arrière-plan.
    - `myfg <job_id>`: Ramener une tâche en arrière-plan au premier plan.
    - `mybg <job_id>`: Relancer une tâche suspendue en arrière-plan.
    - `set name=value`: Définir une variable locale.
    - `unset name`: Supprimer une variable locale.
    - `setenv name=value`: Définir une variable d'environnement.
    - `unsetenv name`: Supprimer une variable d'environnement.
    - `echo $var`: Afficher la valeur d'une variable.
- **Commandes personnalisées :**
    - `myls`: Listage avancé des fichiers avec tri, couleur, et récursivité.
    - `myps`: Affichage des processus en cours avec des informations détaillées (CPU, mémoire, utilisateur, etc.).

Structure du Projet

Répertoire principal :

- `mysh.c`: Point d'entrée principal du shell, gestion des commandes utilisateur et des fonctionnalités internes (`cd`, `exit`, etc.).
- `executor.c`: Contient la logique pour exécuter des commandes externes et gérer les tâches en arrière-plan.
- `parser.c`: Analyse les commandes utilisateur (supporte `&&`, `||`, `;`).
- `redirection.c`: Gère la redirection des entrées/sorties.
- `wildcard.c`: Implémente l'expansion des caractères génériques (`*`, `?`, `[...]`).
- `process_manager.c`: Gère les tâches en arrière-plan et leur état (`myjobs`, `myfg`, `mybg`).
- `variable.c`: Gestion des variables locales et d'environnement avec mémoire partagée.
- `myls.c`: Implémentation de la commande `myls`.
- `myps.c`: Implémentation de la commande `myps`.

Répertoire `include` :

Contient les fichiers d'en-tête pour chaque composant du projet, par exemple :

- `executor.h`
- `parser.h`
- `redirection.h`
- `wildcard.h`
- `process_manager.h`
- `variable.h`

Utilisation

Exemples de commandes :

1. **Commandes standard** :

   ```
   Copier le code
   ls -l | grep ".c" > files.txt
   ```

2. **Commandes internes** :

   ```
   cd /home/user
   echo $HOME
   setenv MY_VAR=123
   unsetenv MY_VAR
   status
   ```

3. **Commandes personnalisées** :

   ```
   myls -aR /path/to/directory
   ```

```
myps
```

4. **Gestion des tâches** :

```
sleep 30 &
myjobs
myfg 1
mybg 1
```

Architecture

1. **Shell interactif (`mysh`)** :
    - Gestion de la boucle principale.
    - Analyse des commandes utilisateur.
    - Exécution des commandes internes et externes.
2. **Exécution des commandes (`executor`)** :
    - Gère les processus enfants pour les commandes externes.
    - Supporte les redirections et pipelines.
3. **Redirections (`redirection`)** :
    - Implémente `>`, `>>`, `<`, `2>` pour les redirections de flux.
4. **Variables (`variable`)** :
    - Gère les variables locales et d'environnement.
    - Utilise la mémoire partagée pour persistance des variables d'environnement.
5. **Tâches en arrière-plan (`process_manager`)** :
    - Permet de lister, suspendre et relancer des tâches en arrière-plan.
6. **Caractères génériques (`wildcard`)** :
    - Supporte les motifs comme `*.c`, `file[1-9].txt`.
7. **Commandes personnalisées** :
    - `myls`: Listage de fichiers avec détails, couleurs, et récursivité.
    - `myps`: Affiche les informations sur les processus en cours.