en details

Pour réaliser ce projet, vous devrez écrire un programme en utilisant un langage de programmation de votre choix. Vous devrez utiliser des fonctions et des structures de données pour gérer les différentes étapes mentionnées ci-dessus.

Voici quelques étapes plus précises pour réaliser ce projet:

1. Afficher un prompt en attente d'une nouvelle commande: vous pouvez utiliser la fonction `printf` ou `cout` pour afficher un prompt sur la console, par exemple "> ".
2. Gérer l'historique des commandes: vous pouvez utiliser un tableau ou une liste chainée pour stocker les commandes saisies. Chaque fois qu'une nouvelle commande est saisie, vous pouvez l'ajouter à l'historique en utilisant une fonction d'ajout. Vous pouvez également ajouter une fonction de recherche dans l'historique pour pouvoir accéder à des commandes précédemment saisies.
3. Rechercher l'exécutable correspondant à la commande saisie: vous pouvez utiliser la fonction `getenv` pour accéder à la variable d'environnement `PATH` qui contient les chemins où sont stockés les exécutables. Vous pouvez parcourir ces chemins pour trouver l'exécutable correspondant à la commande saisie. Si la commande est un chemin relatif ou absolu, vous pouvez vérifier si l'exécutable existe en utilisant la fonction `stat`.
4. Éviter l'utilisation de plus d'une variable globale: vous pouvez utiliser une seule variable globale pour stocker l'historique des commandes. Vous devrez justifier pourquoi vous utilisez cette variable globale, par exemple en expliquant qu'elle est utilisée pour accéder à l'historique des commandes depuis n'importe quelle fonction de votre programme.
5. Ne pas interpréter les guillemets et les caractères spéciaux: vous pouvez utiliser une fonction de parsing de la commande saisie pour séparer les différents éléments de la commande (exécutable, options, arguments, redirections, pipes, etc.). Vous pouvez utiliser des expressions régulières ou des algorithmes de parsing pour extraire ces éléments de la commande. Vous devrez également écrire une fonction pour échapper les guillemets et les caractères spéciaux avant de lancer l'exécutable correspondant.

6. Gérer les guillemets simples: vous pouvez utiliser une fonction pour détecter les guillemets simples dans la commande saisie et échapper les méta-caractères contenus entre ces guillemets. Vous pouvez utiliser une variable booléenne pour stocker l'état "guillemets simples" et changer cet état chaque fois qu'un guillemet simple est détecté. Lorsque l'état est "guillemets simples", vous pouvez échapper les méta-caractères présents dans la séquence entre guillemets.
7. Gérer les guillemets doubles: vous pouvez utiliser une fonction similaire à celle pour les guillemets simples pour détecter les guillemets doubles dans la commande saisie. Vous pouvez utiliser une autre variable booléenne pour stocker l'état "guillemets doubles" et changer cet état chaque fois qu'un guillemet double est détecté. Lorsque l'état est "guillemets doubles", vous pouvez échapper les méta-caractères présents dans la séquence entre guillemets, à l'exception du signe dollar ($).
8. Implémenter les redirections: vous pouvez utiliser les fonctions `dup2` et `pipe` pour gérer les redirections. Pour `<`, vous pouvez utiliser `dup2` pour rediriger l'entrée de la commande vers un fichier spécifié par l'utilisateur. Pour `>`, vous pouvez utiliser `dup2` pour rediriger la sortie de la commande vers un fichier spécifié par l'utilisateur. Pour `<<`, vous pouvez utiliser `dup2` pour rediriger l'entrée de la commande vers un tube créé à l'aide de `pipe`. Pour `>>`, vous pouvez utiliser `dup2` pour rediriger la sortie de la commande vers un fichier spécifié en mode append.
9. Gérer les pipes: vous pouvez utiliser la fonction `pipe` pour créer un tube entre deux commandes. Vous pouvez utiliser `dup2` pour rediriger la sortie de la première commande vers le tube et la rediriger vers l'entrée de la seconde commande. Vous pouvez répéter cette étape pour chaque commande dans la pipeline.
10. Gérer les variables d'environnement: vous pouvez utiliser la fonction `getenv` pour accéder à la valeur d'une variable d'environnement. Vous pouvez parcourir la commande saisie pour détecter les $ suivis d'une séquence de caractères et remplacer cette séquence par la valeur de la variable d'environnement correspondante.
11. Gérer $?: vous pouvez utiliser la fonction `wait` pour attendre la fin d'exécution de la dernière pipeline exécutée en premier plan. Vous pouvez utiliser la valeur renvoyée par `wait`
    pour remplacer $? dans la commande.

12. Gérer les commandes de contrôle: vous pouvez utiliser la fonction `signal` pour gérer les signaux envoyés par les commandes de contrôle (ctrl-C, ctrl-D et ctrl-). Pour ctrl-C, vous pouvez utiliser `signal` pour enregistrer un gestionnaire qui affiche un nouveau prompt sur une nouvelle ligne. Pour ctrl-D, vous pouvez utiliser `signal` pour enregistrer un gestionnaire qui quitte le shell. Pour ctrl-, vous pouvez utiliser `signal` pour enregistrer un gestionnaire qui ne fait rien.
13. Implémenter les built-ins: vous pouvez écrire des fonctions pour chaque built-in mentionné dans le sujet (echo, cd, pwd, export, unset, env, exit). Chaque fonction devra implémenter les comportements spécifiés pour chaque built-in, par exemple pour echo vous devrez gérer l'option -n et afficher les arguments sur la sortie standard.

En général, vous devrez utiliser des concepts de base en programmation tels que les fonctions, les structures de données, les algorithmes de parsing, les expressions régulières, les manipulations de fichiers et de tubes, les signaux et les variables d'environnement pour réaliser ce projet. Vous devrez également faire preuve de créativité pour implémenter les différentes fonctionnalités demandées dans le sujet.