Mini-coque

Aussi beau qu'une coquille

Résumé :
Ce projet consiste à créer un shell simple.
Oui, votre propre petite fête.
Vous en apprendrez beaucoup sur les processus et les descripteurs de fichiers.

Version : 7.1

Contenu

Chapitre I

Introduction

L’existence des coquilles est liée à l’existence même de l’informatique.

À l’époque, tous les développeurs s’accordaient sur le fait que communiquer avec un ordinateur via des Les commutateurs 1/0 étaient vraiment irritants.

C'est tout naturellement qu'ils ont eu l'idée de créer un logiciel permettant de communiquer avec un ordinateur à l'aide de lignes de commandes interactives dans un langage assez proche du langage humain.

Grâce à Minishell, vous pourrez voyager dans le temps et revenir aux problèmes auxquels les gens étaient confrontés lorsque Windows n'existait pas.

Chapitre II

Instructions communes

- Votre projet doit être écrit en C.
- Votre projet doit être rédigé conformément à la Norme. Si vous avez des fichiers/fonctions bonus, ils sont inclus dans le contrôle de norme et vous recevrez un 0 s'il y a une erreur de norme à l'intérieur.
- Vos fonctions ne doivent pas s'arrêter de manière inopinée (erreur de segmentation, erreur de bus, double libre, etc) en dehors de comportements non définis. Si cela se produit, votre projet sera considéré comme non fonctionnel et recevra un 0 lors de l'évaluation.
- Tout l'espace mémoire alloué au tas doit être correctement libéré si nécessaire. Pas de fuites sera tolérée.
- Si le sujet l'exige, vous devez soumettre un Makefile qui compilera vos fichiers sources vers la sortie requise avec les drapeaux -Wall, -Wextra et -Werror, utilisez cc, et votre Makefile ne doit pas établir de nouveau lien.
- Votre Makefile doit contenir au moins les règles $(NAME), all, clean, fclean et concernant.
- Pour donner des bonus à votre projet, vous devez inclure une règle bonus dans votre Makefile, qui ajoutera tous les différents en-têtes, librairies ou fonctions interdites sur la partie principale du projet. Les bonus doivent être dans un fichier différent _bonus.{c/h} si le sujet ne précise rien d'autre. L'évaluation des parties obligatoires et bonus est effectuée séparément.
- Si votre projet vous permet d'utiliser votre libft, vous devez copier ses sources et son Makefile associé dans un dossier libft avec son Makefile associé. Le Makefile de votre projet doit compiler la bibliothèque en utilisant son Makefile, puis compiler le projet.
- Nous vous encourageons à créer des programmes de tests pour votre projet même si ce travail ne devra pas être soumis et ne sera pas noté. Cela vous donnera l'occasion de tester facilement votre travail et celui de vos pairs. Vous trouverez ces tests particulièrement utiles lors de votre soutenance. En effet, lors de la soutenance, vous êtes libre d'utiliser vos tests et/ou les tests du pair que vous évaluez.
- Soumettez votre travail au référentiel git qui vous est attribué. Seul le travail dans le référentiel git sera noté. Si Deepthought est chargé de noter votre travail, cela sera fait

après vos évaluations par les pairs. Si une erreur se produit dans une section de votre travail lors de la notation de Deepthought, l'évaluation s'arrêtera.

Chapitre III

Partie obligatoire

Nom du programme	mini-coque
Remettre les fichiers	Makefile, *.h, *.c NOM,
Arguments	all, clean, fclean, re
du Makefile	
Fonctions externes.	readline, rl_clear_history, rl_on_new_line, rl_replace_line, rl_redisplay, add_history, printf, malloc, gratuit, écrire, accéder, ouvrir, lire, fermer, fourchette, attendre, waitpid, wait3, wait4, signal, sigaction, sigemptyset, sigaddset, tuer, sortir, getcwd, chdir, stat, lstat, fstat, unlink, execve, dup, dup2, pipe, opendir, readdir, closeir, strerror, perror, isatty, ttyname, ttyslot, ioctl, getenv, tcsetattr, tcgetattr, tgetent, tgetflag, tgetnum, tgetstr, tgoto, tputs
Libft autorisé	Oui
Description	Écrire un shell

Votre shell devrait :

- Afficher une invite en attendant une nouvelle commande.
- Avoir un historique de travail.
- Rechercher et lancer le bon exécutable (en fonction de la variable PATH ou à l'aide d'un chemin relatif ou absolu).
- Évitez d'utiliser plus d' une variable globale pour indiquer un signal reçu. Considérez les implications : cette approche garantit que votre gestionnaire de signaux n'accédera pas à vos principales structures de données.

Sois prudent. Cette variable globale ne peut fournir aucune autre information ou accès aux données que le numéro d'un signal reçu.
Par conséquent, utiliser des structures de type « norme » dans la portée globale est interdit.

- Ne pas interpréter les guillemets non fermés ou les caractères spéciaux qui ne sont pas requis par le sujet tel que \ (barre oblique inverse) ou ; (point-virgule).
- Handle ' (guillemet simple) qui devrait empêcher le shell d'interpréter la méta-caractères dans la séquence citée.
- Handle " (guillemet double) qui devrait empêcher le shell d'interpréter les méta-caractères dans la séquence citée à l'exception de $ (signe dollar).
- Implémenter des redirections :
  - < devrait rediriger l'entrée.
  - > devrait rediriger la sortie.
  - << doit recevoir un délimiteur, puis lire l'entrée jusqu'à ce qu'une ligne contenant le délimiteur apparaisse. Cependant, il n'est pas nécessaire de mettre à jour l'historique !
  - >> devrait rediriger la sortie en mode ajout.
- Implémenter des tuyaux (caractère |). La sortie de chaque commande dans le pipeline est connecté à l'entrée de la commande suivante via un tuyau.
- Gérer les variables d'environnement ($ suivi d'une séquence de caractères) qui devraient s'étendre à leurs valeurs.
- Gérer $? qui devrait s'étendre au statut de sortie du pipeline de premier plan exécuté le plus récemment.
- Gérez ctrl-C, ctrl-D et ctrl-\ qui doivent se comporter comme dans bash.
- En mode interactif :
  - ctrl-C affiche une nouvelle invite sur une nouvelle ligne.
  - ctrl-D quitte le shell.
  - ctrl-\ ne fait rien.
- Votre shell doit implémenter les éléments intégrés suivants :
  - écho avec l'option -n
  - cd avec seulement un chemin relatif ou absolu
  - mot de passe sans options
  - exporter sans options
  - désarmé sans options
  - env sans options ni arguments
  - quitter sans options

La fonction readline() peut provoquer des fuites de mémoire. Vous n'êtes pas obligé de les réparer. Mais cela ne signifie pas que votre propre code, oui, le code que vous avez écrit, peut avoir des fuites de mémoire.

Vous devez vous limiter à la description du sujet. Tout ce qui n'est pas demandé n'est pas obligatoire.
Si vous avez le moindre doute sur une exigence, prenez bash pour référence.

Chapitre IV

Partie bonus

Votre programme doit mettre en œuvre :

- && et || avec des parenthèses pour les priorités.
- Les caractères génériques * devraient fonctionner pour le répertoire de travail actuel.

La partie bonus ne sera évaluée que si la partie obligatoire est PARFAITE.
Parfait signifie que la partie obligatoire a été intégralement réalisée et fonctionne sans dysfonctionnement. Si vous n'avez pas satisfait à TOUTES les exigences obligatoires, votre partie bonus ne sera pas du tout évaluée.

Chapitre V

Soumission et évaluation par les pairs

Remettez votre devoir dans votre référentiel Git comme d'habitude. Seul le travail à l'intérieur de votre
Le référentiel sera évalué lors de la soutenance. N'hésitez pas à vérifier
noms de vos fichiers pour vous assurer qu'ils sont corrects.

M LQ * de Q z Ô kv P.
:)? 6 tu nk] R *
F *q IV - Ô , ?ZP
7 c 1V H
e 9 () 8
U
' > D s / w
; E ' >M
q < >

par exemple H (3
je X 0g E 6