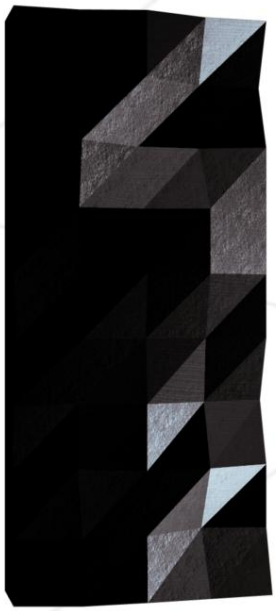

Miniconcha

Tão linda quanto uma concha

Resumo:

Este projeto trata da criação de um shell simples.

Sim, sua própria pequena festa.

Você aprenderá muito sobre processos e descritores de arquivos.

Versão: 7.1

Conteúdo

Capítulo I

Introdução

A existência de shells está ligada à própria existência de TI.

Na época, todos os desenvolvedores concordaram que *a comunicação com um computador usando Os interruptores 1/0 eram seriamente irritantes.*

Era lógico que tivessem tido a ideia de criar um software para comunicar com um computador utilizando linhas de comandos interactivas numa linguagem algo próxima da linguagem humana.

Graças ao Minishell você poderá viajar no tempo e voltar aos problemas as pessoas enfrentavam quando *o Windows* não existia.

Capítulo II

Instruções Comuns

- Seu projeto deve ser escrito em C.
- Seu projeto deverá ser escrito de acordo com a Norma. Se você tiver arquivos/funções bônus, eles serão incluídos na verificação de norma e você receberá 0 se houver um erro de norma.
- Suas funções não devem encerrar inesperadamente (falha de segmentação, erro de barramento, liberação dupla, etc.) além de comportamentos indefinidos. Caso isso aconteça, seu projeto será considerado não funcional e receberá nota 0 na avaliação.
- Todo o espaço de memória alocado no heap deve ser liberado adequadamente quando necessário. Sem vazamentos será tolerado.
- Se o assunto exigir, você deve enviar um Makefile que irá compilar seus arquivos fonte para a saída necessária com as flags -Wall, -Wextra e -Werror, use cc, e seu Makefile não deve relinkar.
- Seu Makefile deve conter pelo menos as regras $(NAME), all, clean, fclean e ré.
- Para entregar bônus ao seu projeto, você deve incluir uma regra bônus em seu Makefile, que adicionará todos os diversos cabeçalhos, bibliotecas ou funções que são proibidas na parte principal do projeto. Os bônus devem estar em um arquivo diferente _bonus.{c/h} se o assunto não especificar mais nada. A avaliação da parte obrigatória e da parte bônus é feita separadamente.
- Se o seu projeto permite que você use sua libft, você deve copiar seus fontes e seu Makefile associado em uma pasta libft com seu Makefile associado. O Makefile do seu projeto deve compilar a biblioteca usando seu Makefile e depois compilar o projeto.
- Nós encorajamos você a criar programas de teste para o seu projeto, mesmo que este trabalho **não precise ser enviado e não receba notas.** Isso lhe dará a chance de testar facilmente seu trabalho e o de seus colegas. Você achará esses testes especialmente úteis durante sua defesa. Na verdade, durante a defesa, você é livre para usar seus testes e/ou os testes do colega que está avaliando.
- Envie seu trabalho para o repositório git designado. Somente o trabalho no repositório git será avaliado. Se Deepthought for designado para avaliar seu trabalho, isso será feito

após suas avaliações por pares. Se ocorrer um erro em qualquer seção do seu trabalho durante a avaliação do Deepthought, a avaliação será interrompida.

Capítulo III

Parte obrigatória

Nome do programa	miniconcha
Entregar arquivos	Makefile, *.h, *.c NOME,
Argumentos	tudo, limpo, fclean, re
Makefile	
Funções externas.	readline, rl_clear_history, rl_on_new_line, rl_replace_line, rl_redisplay, add_history, printf, malloc, free, write, access, open, read, close, fork, wait, waitpid, wait3, wait4, signal, sigaction, sigemptyset, sigaddset, kill, exit, getcwd, chdir, stat, lstat, fstat, desvincular, execve, dup, dup2, pipe, opendir, readdir, closedir, strerror, perror, isatty, ttyname, ttyslot, ioctl, getenv, tcsetattr, tcgetattr, tgetent, tgetflag, tgetnum, tgetstr, tgoto, tputs
Libft autorizado	Sim
Descrição	Escreva uma concha

Seu shell deve:

- Exibir um **prompt** ao aguardar um novo comando.
- Ter um **histórico de trabalho.**
- Pesquise e inicie o executável correto (com base na variável PATH ou usando um relativo ou um caminho absoluto).
- Evite usar mais de **uma variável global** para indicar um sinal recebido. Considere as implicações: esta abordagem garante que o seu manipulador de sinal não acessará suas principais estruturas de dados.

Tome cuidado. Esta variável global não pode fornecer nenhuma outra informação ou acesso a dados além do número de um sinal recebido.
Portanto, usar estruturas do tipo "norma" no escopo global é proibido.

- Não interpretar aspas não fechadas ou caracteres especiais que não sejam exigidos pela assunto como \ (barra invertida) ou ; (ponto e vírgula).
- Handle ' (aspas simples) que deve evitar que o shell interprete o meta-caracteres na sequência citada.
- Handle " (aspas duplas) que deve evitar que o shell interprete os metacaracteres na sequência entre aspas, exceto $ (cifrão).
- Implementar **redirecionamentos:**
  - ÿ < deve redirecionar a entrada.
  - ÿ > deve redirecionar a saída.
  - ÿ << deve receber um delimitador e, em seguida, leia a entrada até que uma linha contendo o delimitador seja vista. No entanto, não é necessário atualizar o histórico!
  - ÿ >> deve redirecionar a saída no modo de acréscimo.
- Implementar **pipes** (caractere |). A saída de cada comando no pipeline é conectado à entrada do próximo comando através de um tubo.
- Lidar com **variáveis de ambiente** ($ seguido por uma sequência de caracteres) que devem se expandir para seus valores.
- Lidar com $? que deve se expandir para o status de saída do pipeline em primeiro plano executado mais recentemente.
- Manipule ctrl-C, ctrl-D e ctrl-\ que devem se comportar como no bash.
- No modo interativo:
  - ÿ ctrl-C exibe um novo prompt em uma nova linha.
  - ÿ ctrl-D sai do shell.
  - ÿ ctrl-\ não faz nada.
- Seu shell deve implementar os seguintes **componentes internos:**
  - ÿ eco com opção -n
  - ÿ cd com apenas um caminho relativo ou absoluto
  - ÿ senha sem opções
  - ÿ exportar sem opções
  - ÿ não definido sem opções
  - ÿ env sem opções ou argumentos
  - ÿ sair sem opções

A função readline() pode causar vazamentos de memória. Você não precisa consertá-los. Mas isso **não significa que seu próprio código, sim, o código que você escreveu, pode ter vazamentos de memória.**

Você deve se limitar à descrição do assunto. Tudo o que não é pedido não é obrigatório.

Se você tiver alguma dúvida sobre um requisito, faça o bash como referência.

Capítulo IV

Parte bônus

Seu programa deve implementar:

- && e || com parênteses para prioridades.
- Curingas * devem funcionar para o diretório de trabalho atual.

A parte bônus só será avaliada se a parte obrigatória for PERFEITA. Perfeito significa que a parte obrigatória foi feita integralmente e funciona sem mau funcionamento. Se você não passou em TODOS os requisitos obrigatórios, sua parte do bônus não será avaliada de forma alguma.

Capítulo V

Envio e avaliação por pares

Entregue sua tarefa em seu repositório Git normalmente. Somente o trabalho dentro do seu repositório será avaliado durante a defesa. Não hesite em verificar novamente o nomes de seus arquivos para garantir que estejam corretos.

M QL * de Q z Ó kv P
:)? 6 você ok] R *
f *q IV - Ó , ?ZP
7 c 1V H
e 9 () 8
você
' > D é / c
; E ' >M
q < >

zb H (3
eu X 0gE 6