Projeto de um Shell Simplificado

DEL/Poli/UFRJ

PLE 2020

1 Entrega e Pontuação

A data de entrega dos trabalhos será 6h (da manhã) do dia 16/11/2020, ou seja, no final do período. **Não haverá adiamento da entrega**. O código deve ser entregue em um arquivo .zip pelo e-mail **rodsouzacouto@poli.ufrj.br**. **ATENÇÃO: Não envie executáveis, pois serão rejeitos pelo gmail. Enviem apenas o código fonte, com o Makefile.**

A ideia inicial era ter passado este trabalho na primeira semana do curso. Entretanto, achei melhor adiar, para dar tempo de vocês consolidarem os conceitos. Como atrasei a divulgação, eu simplifiquei bastante o trabalho.

O trabalho valerá 1,5 pontos na média final. Ele é bem simples, dado o atraso mencionado. O próximo trabalho, apesar de valer a mesma quantidade, será mais complexo.

2 Objetivo do Trabalho

Neste trabalho você fará o seu próprio shell do Linux, bem simplificado. Siga exatamente as exigências do trabalho. Por exemplo, por mais que seja uma melhor prática, não faça o programa pegar argumentos por argc e argv, como um shell comum. Se quiser fazer algo mais complexo, para aprender, submeta em um arquivo .zip separado. Mas o programa mais simples tem que seguir exatamente as exigências mencionadas.

3 Funcionamento básico

O shell desenvolvido, por ser simplificado, é diferente dos shell tradicionais. Ele não precisar pegar argumentos da linha de comando.O shell vai perguntando para o usuário cada argumento.

1. Ao abrir, o shell irá perguntar “Qual comando quer executar?”
2. O usuário deverá digitar um comando, sem argumento algum. Lembrando que programas ficam no /bin (Linux) ou no /sbin (mac), mas o usuário não precisar digitar isso. Seu código deve tratar essa situação.
3. Após digitar o comando, o shell irá perguntar “Quantos argumentos você quer digitar?”.
4. O usuário deverá digitar o número de argumentos. Por exemplo, para “ping 8.8.8.8”, o usuário digitará 1. Para “ping 8.8.8.8 -c 5”, o usuário digitará 3. Lembrem-se, para a chamada exec, o nome do programa também é um argumento, mas isso você não precisa pedir para o usuário digitar de novo.
5. Após digitação do número de argumentos, o shell irá iterar pedindo para o usuário digitar cada argumento. Ex: “Digite o argumento 1.”.
6. O shell irá executar o comando, usando chamadas fork e exec (pode ser qualquer chamada da família exec).

7. O shell deverá esperar a finalização do comando para finalizar. Por exemplo, o comando "ping 8.8.8.8 -c 3" irá finalizar após 3 envios de ping.

4 Recepção de sinais

O shell deve estar preparado para receber o sinal SIGUSR1 (código 10). Se receber o sinal enquanto estiver nos passos de 2 a 5, o shell deve voltar ao Passo 1 e esquecer o comando e argumentos digitados. Você não precisa se preocupar com a recepção do sinal a partir do passo 6.

O envio de sinais se dará por meio de um outro terminal com o comando "kill -10 pidDoProcesso".

Dica 1: Para saber o pid de um processo, faça "ps aux — grep nomeDoProcesso".

Dica 2: Vejam algumas informações práticas sobre sinais em
`https://www.thegeekstuff.com/2012/03/linux-signals-fundamentals/` e
`https://www.thegeekstuff.com/2012/03/catch-signals-sample-c-code/`. Fiz um resumo sobre o tema em `https://drive.google.com/file/d/1EieO-jz3AGPHuaQgZclyYWnjVl8fLV-0/view?usp=sharing`. Lembrando que é necessário logar com o PoliMail.

5 Outras Exigências

- O programa deve ser escrito em C;
- Não é permitido usar popen() ou outras funções que abram shells. A execução do processo filho tem que ser realizada a partir da chamada exec;
- O código deve vir acompanhado de Makefile. Códigos que não forem entregues em condições de serem compilados não serão corrigidos. o seguinte código tem exemplo de MakeFile `https://drive.google.com/file/d/1wWEsfPYXpFlcPJZywxCdjW_JX4oEA-aJ/view?usp=sharing`. Lembrando que é necessário logar com o PoliMail;
- Lembrem-se de escolher nomes adequados para as variáveis e identar;
- Muito importante: O código deve ser comentado para permitir o entendimento do raciocínio utilizado por vocês para escrever o programa;
- O código tem que ser tolerante a erros básico, como o usuário digitar um comando inválido;
- O trabalho deve ser realizado de forma individual;
- **Atentem-se para as regras de cópia de trabalho desta disciplina. Não haverá exceções!!!!!**

6 Plataforma

O código poderá ser desenvolvido para o Linux, para o MacOS, ou para qualquer outro sistema UNIX. Caso queiram utilizar o WSL do Windows, também podem tentar utilizá-lo. Entretanto, posso tentar ajudar a solucionar dúvidas, mas não darei suporte a problemas específicos do WSL. Caso queiram usar o WSL, vejam se conseguem fazer as chamadas fork e exec sem problemas. Vejam também se é possível executar comando kill em outro terminal. Ou seja, as plataformas recomendadas são o Linux (virtualizado ou não) e o MacOS.

Dica1: caso queiram testar se a chamad fork e exec funcionam no WSL, usem o seguinte código de exemplo
`https://drive.google.com/file/d/1wWEsfPYXpFlcPJZywxCdjW_JX4oEA-aJ/view?usp=sharing`

Dica2:

O código mencionado acima também é útil para quem quiser aprender a utilizar a chamada fork e exec.

Dica3:

O seguinte link possui uma VM com Linux para uso no Virtualbox. Seu uso não é obrigatório, mas pode ajudar quem não tiver muita memória RAM para máquina virtual:
`https://drive.google.com/file/d/1-R9i3QB3aqnLMBQhUlz03K2lM8yvxp7P/view?usp=sharing` . Lembrando que é necessário logar com o PoliMail.

Quem tiver qualquer problema para ter acesso a alguma plataforma necessária (Linux, Virtualbox, WSL, etc.), me avise o quanto antes para tentarmos resolver.