

Trabalho Prático #1 Desenvolvimento de um Shell básico + TOP

PARTE UM: Desenvolvimento de um Shell básico

1) Introdução

Neste trabalho iremos explorar alguns dos principais conceitos da primeira parte da disciplina. Em particular, iremos rever os conceitos de *pipes* e estruturas de processos de *kernel*. Para isso, você se familiarizará com a interface de chamadas de sistema do Linux implementando algumas funcionalidades de um shell bem simples. Para que você foque apenas na parte de chamadas de sistema, abra o arquivo fonte do shell *sh.c* e estude-o. O esqueleto do shell contém duas partes: *um processador de linhas de comando* e *o código para execução dos comandos*. Você não precisa modificar o processador de linhas de comando, **mas deve completar o código para execução dos comandos**. O processador de linhas só reconhece comandos simples como:

```
ls > y
cat < y | sort | uniq | wc > y1
cat y1
rm y1
ls | sort | uniq | wc
rm y
```

Se você não entende o que esses comandos fazem, estude o manual de um shell do Linux (por exemplo, do *bash*) bem como o manual de cada um dos comandos acima (*ls*, *cat*, *rm*, *sort*, *uniq*, *wc*) para se familiarizar. Copie e cole esses comandos num arquivo, por exemplo, *teste.sh*.

Você pode compilar o esqueleto do shell rodando:

```
$ gcc sh.c -o myshell.out
```

Nota: Nesta especificação colocamos um sinal de cifrão (\$) antes das linhas que devem ser executadas no shell do sistema (por exemplo, o *bash*). As linhas de comando sem cifrão devem ser executadas no shell simplificado que você está implementando.

Esse comando irá produzir um arquivo *myshell.out* que você pode rodar:

```
$ ./myshell.out
```

Para sair do shell simplificado aperte **ctrl+d** (fim de arquivo). Teste o shell executando os comandos no arquivo *teste.sh*:

```
$ ./myshell.out < teste.sh
```

Essa execução irá falhar pois você ainda não implementou várias funcionalidades do shell. É isso que você fará nesse trabalho.

2) Executando Comandos Simples

Implemente comandos simples, como:

```
ls
```

O processador de linhas já constrói uma estrutura `execcmd` para você, a única coisa que você precisa fazer é escrever o código do `case` ' ' (espaço) na função `runcmd`. Depois de escrever o código, teste execução de programas simples como:

```
ls
cat sh.c
```

Nota: Você não precisa implementar o código do programa `ls` e dos outros. O que você deve fazer é simplesmente implementar as funções no esqueleto do shell simplificado para permitir que ele execute comandos já existentes no sistema, como acima.

Dica: dê uma olhada no manual da função `exec` (`$ man 3 exec`) – **IMPORTANTE: NÃO use a função `system` para implementar as funções do seu shell.**

3) Redirecionamento de Entrada e Saída

Implemente comandos com redirecionamento de entrada e saída para que você possa rodar:

```
echo "Trabalho Prático 1 de Sistemas Operacionais " > x.txt
cat < x.txt
```

O processador de linhas já reconhece os símbolos `>` e `<` e constrói uma estrutura `redircmd` para você. Seu trabalho é apenas preencher o código na função `runcmd` para esses casos. Teste sua implementação com os comandos acima e outros comandos similares.

Dica: Dê uma olhada no manual das funções `open` e `close` (`$ man 2 open`). Caso você não conheça o esquema de entrada e saída padrão de programas, dê uma olhada no artigo da Wikipedia¹.

¹http://en.wikipedia.org/wiki/Standard_streams

4) Sequenciamento de Comandos

Implemente *pipes* para que você consiga rodar comandos do tipo:

```
ls | sort | uniq | wc
```

O processador de linhas já reconhece o `|` e constrói uma estrutura `pipecmd` pra você. A única coisa que você precisará fazer é completar o código para o case `|` na função `runcmd`. Teste sua implementação para o comando acima. Se precisar, leia a documentação das funções `pipe`², `fork`³ e `close`⁴.

5) Histórico de Comandos

Implemente um histórico para que você consiga visualizar os comandos executados anteriormente:

```
history
1 ls > y
2 cat < y | sort | uniq | wc > y1
3 cat y1
4 rm y1
5 ls | sort | uniq | wc
6 rm y
```

Você deve criar uma estrutura de buffer para armazenar até 50 comandos executados anteriormente em seu shell simplificado e, por meio do comando `history` exibir ao usuário todos os comandos armazenados no buffer. Caso novos comandos sejam inseridos de modo a exceder os 50 comandos armazenados, você deve limpar os comandos mais antigos primeiro, mantendo em seu buffer a ordem em que foram executados.

PARTE DOIS: Utilizando `/proc/` para fazer um TOP

A segunda parte deste trabalho prático consiste de um novo programa. Agora, vamos aprender um pouco sobre arquivos especiais e o comando `top`. Dê uma olhada nos arquivos `/proc/*/stat`. O `/proc` é um diretório de arquivos especiais do Linux que lista os processos em execuções. Este arquivo embora possa ser lido como um arquivo normal, na verdade não é um arquivo em disco e sim um file handle para um arquivo especial que lista informações do kernel.

Crie um programa que gera um executável `meutop`. Tal programa deve imprimir os processos sendo executados em sequência. Para cada programa, identifique o PID do programa, o usuário que

²<https://linux.die.net/man/2/pipe>

³<https://linux.die.net/man/2/fork>

⁴<https://linux.die.net/man/2/close>

está executando o mesmo e o estado do processo. Com isto, imprima os programas em execução em uma tabela como a abaixo. Tal tabela deve ser atualizada a cada 1 segundo.

Seu comando deve se chamar `meutop`.

PID	User	PROCNAME	Estado
1272	george	yes	S
1272	root	init	S

Dica Você pode limitar seu top para imprimir apenas os 20 primeiros processos que encontrar. Dê uma olhada no manpage do `/proc: Manpage`⁵

Outra fonte de dados é o código do `ps`: `PS`⁶

Vamos agora nos inspirar no comando `top` ou `htop` para aprender um pouco mais sobre sinais. O `htop` é um top avançado no linux que permite o envio de sinais para processos em execução.

Permita que seu comando TOP envie sinais. Isto é, crie uma função no seu TOP que enviar sinais para um PID. Tal função pode ser apenas digitar um "PID SINAL". Por exemplo, se o `signaltester` tem PID 2131, o código abaixo deve enviar o sinal `SIGHUP` para o mesmo.

PID	User	PROCNAME	Estado
2131	george	signaltester	S

> 2131 1

Com este sinal o processo deve morrer e sair da sua lista.

Dica 1: Possivelmente o `signaltester` não vai aparecer entre os 20 primeiros. Use o comando abaixo para descobrir o PID do mesmo e testar seu `topzera`.

Dica 2: O usuário que é dono do processo é o usuário dono do arquivo no `/proc`. A biblioteca `dirent.h` pode lhe ajudar.

Dica 3: A biblioteca `sys/stat.h` também pode ser de ajuda.

Dica 4: Para enviar o sinal use a função `kill`⁷

```
$ ps | grep signalteste
```

⁵ <http://man7.org/linux/man-pages/man5/proc.5.html>

⁶ <https://github.com/thlorenz/procps/blob/master/deps/procps/proc/readproc.c>

⁷ <https://linux.die.net/man/2/kill>

Código de Teste: Disponibilizei um código `signaltester` para você testar o seu trabalho. O mesmo faz um tratamento simples de sinais em C.

Compile o teste com a linha

```
$ gcc signaltester.c -o signaltester
```

Uma forma de permitir a leitura da entrada e a impressão da lista de processos é utilizando `pthread`s. Tente fazer isto.

Entrega

Esse trabalho poderá ser feito em dupla.

A data de entrega é 30/03/2020, segunda-feira, até as 23:55 via moodle.

Seu grupo deverá submeter no moodle um zip contendo **apenas** o arquivo `sh.c`, em um (único) arquivo chamado `sh.c` (o nome deve ser exatamente `sh.c` para que seu shell possa ser testado automaticamente) para a parte um do trabalho prático. Para a segunda parte, adicione ao mesmo zip o executável `meutop` tal como o arquivo `.c` utilizado para a implementação do mesmo. Além disso, o grupo também deve submeter um relatório descrevendo a solução implementada tal como as decisões de implementação tomadas para chegar em tal solução.

Esse trabalho vale **10** pontos.

Esclarecimentos

1. Esse trabalho utiliza obrigatoriamente a linguagem C.
2. Um grupo de discussão para esse trabalho será aberto no moodle para resolver quaisquer dúvidas.
3. Seu shell será testado com o script `grade.sh` disponibilizado na especificação. A saída será conferida automaticamente. Por causa disso, seu shell deve imprimir somente a saída dos programas em casos onde não ocorre erro. Use o script `grade.sh` disponibilizado para verificar a corretude de sua implementação.
4. Comece a fazer o trabalho o quanto antes!