

# TP1: Shell + TOP + Sinais

---

1. Entrega no domingo após a primeira prova
2. Pode ser feito em dupla

Parte deste material foi adaptado do material do [Prof. Italo Cunha](#)

Neste TP vamos explorar alguns conceitos da primeira parte da disciplina. Em particular, vamos rever os conceitos de Pipes, Estruturas de Processos do Kernel e Sinais.

## Parte 1: Desenvolvendo um Shell Básico

---

Neste trabalho você se familiarizará com a interface de chamadas de sistema do Linux implementando algumas funcionalidades num shell simples. Para que você foque apenas na parte de chamadas de sistema, baixe o [esqueleto](#) do shell e o estude. O esqueleto do shell contém duas partes: um processador de linhas de comando e 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, grade.sh.

Você pode compilar o esqueleto do shell rodando:

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

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

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

```
$ ./myshell
```

Para sair do shell simplificado aperte ctrl+d (fim de arquivo). Teste o shell executando o grade.sh:

```
$ grade.sh
```

Essa execução irá falhar pois você ainda não implementou várias funcionalidades do shell. É isso que você fará nesse trabalho. 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`; você deve simplesmente implementar as funções no esqueleto do shell simplificado para permitir que ele execute comandos simples como acima.

Se ainda não conhecê-la, dê uma olhada no manual da função `exec` (`$ man 3 exec`). Importante: não use a função `system` para implementar as funções do seu shell.

### Redirecionamento de entrada/saída

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

```
$ echo "DCC605 is cool" > x.txt  
$ cat < x.txt
```

O processador de linhas já reconhece ">" 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`). Se você não conhece o esquema de entrada e saída padrão de programas, dê uma olhada no artigo da Wikipedia [aqui](#).

### Sequenciamento de comandos

Implemente pipes para que você consiga rodar comandos tipo

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

O processador de linhas já reconhece '|' e constrói uma estrutura `pipecmd` pra você. A única coisa que você precisa 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`, `fork` e `close`.

1. <https://linux.die.net/man/2/fork>
2. <https://linux.die.net/man/3/exec>
3. <https://linux.die.net/man/2/pipe>
4. <https://linux.die.net/man/2/dup>

### Esclarecimentos

1. Não use a função `system` na sua implementação. Use `fork` e `exec`.

### Exemplos

1. [Fork](#)
2. [Fork com pipe](#)

## Parte 2: Utilizando `/proc/` para fazer um TOP

A segunda parte do TP 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 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	flavio	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](#)

Outra fonte de dados é o código do `ps`: [PS](#)

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     | flavio  | 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`

```
$ ps | grep signalteste
```

**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`. Tente fazer isto.