

## 1. Introdução

Este trabalho prático pode ser dividido em 4 partes: (1) Shell Básico, (2) PSTree, (3) Topzera e (4) Sinais.

O Shell Básico consiste na implementação de um novo Shell que permite a execução de comandos, o uso de redirecionadores e de pipes. A PSTree consiste numa árvore dos processos do sistema, organizada de forma a se identificar o pai e filhos de cada processo. O Topzera imprime uma lista dos processos do sistema, informando o PID, User, Nome e Estado de cada processo. Por fim, há o Sinais, que envia sinais para processos a partir do Topzera, modificando seus Estados.

## 2. Implementação

### 2.1. Shell Básico

Para construção do Shell Básico foi utilizado o esqueleto fornecido na especificação do TP. Neste esqueleto há três casos de operação: (1) executar comandos simples, (2) redir >, (3) redir < e (4) pipes. Uma vez compreendida a estrutura do esqueleto foram implementadas as funcionalidades das 4 operações acima.

Para permitir operações de comandos simples no Shell Básico foi utilizado o `execvp`, que substitui o processo atual com um novo a partir de seus argumentos. No primeiro argumento é passado o arquivo a ser executado, que no caso é o `argv[0]` do nosso Shell, que contém o nome do comando a ser executado. O segundo parâmetro são os argumentos do comando a ser executado, que é o `argv` passado pelo shell.

Para criar a funcionalidade de utilizar os redir < (redireciona saída) e > (redireciona entrada) são utilizados um `dup2` para cada.

Para o > é aberto um *filedescriptor* através da função `open`, com o acesso `O_WRONLY|O_CREAT|O_TRUNC` que permite criar o arquivo e abrir no modo escrita caso ele não exista e abrir no modo escrita caso ele já exista. Também é concedida a permissão `S_IRWXU` para tal *filedescriptor*, que fornece permissão para ler, escrever e executar. Na função `open` também é passado um arquivo, que possui as informações digitadas no Shell Básico após o símbolo >, correspondendo à entrada a ser redirecionada do `stdout`, de forma que ao invés de enviar o conteúdo para o `stdout`, ele envia esse novo arquivo com as informações digitadas no Shell Básico.

É utilizado então um *dup2*, que cria uma cópia do *filedescriptor* passado no segundo parâmetro (*STDOUT\_FILENO*, que corresponde ao *stdout*), com o conteúdo do primeiro parâmetro, que será o *filedescriptor* que foi criado com a função *open* e descrito acima. Dessa forma, ao invés da saída ser impressa no *stdout*, será impressa no arquivo desejado.

Para o < a lógica é a mesma, porém desejamos realizar o contrário, redirecionar a saída. Então nosso *filedescriptor* precisará apenas ler o arquivo que possui a nova saída desejada (modo *O\_RDONLY*). Para duplicar o processo é utilizado o *dup2*, com os parâmetros sendo o *filedescriptor* criado e *STDIN\_FILENO*, que corresponde ao *filedescriptor* do *stdin* do sistema. Assim é redirecionada a entrada, que ao invés de ser lido do *stdin* será lida do arquivo criado.

Para o | (pipe), a primeira abordagem foi semelhante ao *redir*, pensando em criar um arquivo para receber/transmitir o fluxo de dados (*iostream*), mas a solução apresentou alguns problemas. Após iluminação do professor, a utilização de arquivo temporário foi substituída pela função *pipe()*, que, de posse de dois descritores de arquivo, estabeleceu então o fluxo entre a saída (*stdout*) do processo filho e a entrada (*stdin*) do processo pai, auxiliada pela *dup2()*, que já foi explicada. Com isso, a função *runcmd* foi recursivamente chamada para o lado esquerdo (do pipe) no filho e para o lado direito no processo pai, após término do filho.

## 2.2. PSTree

Para Implementação da PSTree foi utilizado o diretório */proc*, que possui diretórios com diversas informações sobre os processos, separadas por pastas identificadas pelo PID de cada processo. Especificamente, foram utilizados os file handle *stat* e *children* dos processos do sistema.

O arquivo *stat* (*/proc/[pid]/stat*) contém diversas informações sobre o processo a qual se refere e dele foram utilizadas as seguintes informações:

- *pid*: ID do processo;
- *comm*: nome do processo.

Já o arquivo *children* (*/proc/[pid]/task/[tid]/children*) possui uma lista separada por espaços contendo os IDs dos filhos do processo ao qual o arquivo se refere.

Para implementação, primeiro é calculado a quantidade de processos do sistema, caminhando pelos diretórios do */proc* e verificando se a pasta corresponde a um processo (caso o nome for um número significa que a mesma se refere a um processo). Após essa contagem, o número de processos encontrado servirá para alocar a memória para os dados dos processos e caso mais um processo se inicie durante a criação da PSTree não ocorrerá segmentação de memória tentando ler mais processos do que o espaço alocado na memória permite.

Então é realizada novamente caminhada no procedimento *PreencheDadosProcessos* pelos diretórios do */proc*, porém dessa vez lendo o

arquivo *stat* de cada processo e armazenando em um vetor os dados desejados (pid e nome).

Para encontrar os filhos de cada processo é acessado o arquivo *children* de cada qual e são lidos os *tids* (ids dos processos filhos) pelo procedimento *LeChildren*. Uma matriz é alocada para armazenar os filhos de cada processo, de forma que cada linha da matriz representa um processo e os pids dos filhos sejam armazenados ao longo dessas linhas.

Então é impressa a PSTree através do procedimento *ImprimePSTree*, caminhando recursivamente pela matriz criada de forma a montar a árvore e coletando os nomes dos processos armazenados no vetor *processos* através de uma pesquisa binária.

### 2.3. Topzera

O objetivo dessa parte foi de, inspirado pelo comando *top*, criar uma lista de processos que é atualizada em tempo real. Essa lista contém o ID, o Usuário (dono), o Nome e o Estado dos processos, sendo que eles estão ordenados pelo PID e a atualização é feita a cada segundo.

Todo o código principal está dentro de um *while(1)*, responsável pelo loop da atualização. Dentro dele, verifica-se se já passou 1 segundo da última atualização e, em caso positivo, todo o processo (a ser descrito) é repetido.

Tal processo consiste em: limpar a tela, escrever o cabeçalho e ler os arquivos dos processos enquanto escreve na tela suas informações requeridas. O código ainda possui comandos para chegar até esses arquivos e verificar possíveis erros, sendo que foi todo desenvolvido em ANSI C.

A escolha da linguagem C contou com algumas dificuldades, como a de limpar a tela (que foi solucionada com *ASCII escape sequences*). Além disso, a única forma de interromper o loop é com Ctrl+C.

### 2.4. Sinais

Como foi proposto, o módulo de Sinais foi desenvolvido sobre o Topzera, mas com algumas alterações relevantes na essência. Para implementar a atualização da tabela juntamente com o recebimento de comandos (principalmente para enviar um sinal para um processo), utilizou-se o comando *select()* e outros comandos relacionados.

Sobre a estrutura geral, o *while(1)* foi substituído por um *while(keepLooping)*, sendo que *keepLooping* é uma variável que determina o não recebimento da expressão 'q' (quit), como muitos programas em linux fazem. Além disso, adicionou-se também tratamentos para o caso do próprio programa receber um dos três sinais para morrer (*SIGINT*, *SIGHUP* ou *SIGTERM*).

A estrutura interna, referente à montagem da tabela, permaneceu praticamente intacta, enquanto a parte verdadeiramente responsável pelos sinais

acompanhou a adição do reconhecimento da expressão 'q'. De forma geral, o `select()` espera 5 segundos para a chegada de um input (que se tornou o novo intervalo de atualização da tabela) e, caso esse não chegue, a tabela é atualizada e o recebimento de input volta a ser acionado.

Esse recebimento é determinado por um "\n" (*Enter* pressionado), sendo que as expressões aceitas são o 'q' (como já foi explicado) e o formato "<signalPID> <signalCode>". Essa última que caracteriza o envio de um sinal, de forma que o *signalPID* informa o ID do processo destinatário e *signalCode* o código do sinal que se deseja enviar.

### 3. Testes

#### 3.1. Shell Básico

Segue abaixo a execução de alguns comandos no Shell Básico criado.

```
fernanda@Terra:~/ShellBasico$ ./ShellBasico
$ ls > y
$ cat y
a.out
mys.c
README.md
sh.c
ShellBasico
signalingTop.cpp
teste.sh
topzera.c
y
$ cat < y | sort | uniq | wc > y1
$ cat y1
      9      9      78
$ rm y1
$ ls
a.out  README.md  ShellBasico  teste.sh  y
mys.c  sh.c        signalingTop.cpp  topzera.c
$ ls | sort | uniq | wc
      9      9      78
$ ls
a.out  README.md  ShellBasico  teste.sh  y
mys.c  sh.c        signalingTop.cpp  topzera.c
$ rm y
$ ls
a.out  README.md  ShellBasico  teste.sh
mys.c  sh.c        signalingTop.cpp  topzera.c
$ █
```

Também foram realizados os testes disponíveis juntamente com o esqueleto do Shell e todos rodaram com sucesso.

```
fernanda@Terra:~/ShellBasico$ ./ShellBasico test.sh
$ ^C
fernanda@Terra:~/ShellBasico$ ./ShellBasico test0.sh
$ ^C
fernanda@Terra:~/ShellBasico$ ./ShellBasico test1.sh
$ ^C
fernanda@Terra:~/ShellBasico$ ./ShellBasico test2.sh
$ ^C
fernanda@Terra:~/ShellBasico$ ./ShellBasico test3.sh
$ ^C
fernanda@Terra:~/ShellBasico$ ./ShellBasico test4.sh
$ ^C
fernanda@Terra:~/ShellBasico$ ./ShellBasico test5.sh
$ ^C
fernanda@Terra:~/ShellBasico$ ./ShellBasico test6.sh
$ ^C
fernanda@Terra:~/ShellBasico$ ./ShellBasico test7.sh
$ ^C
fernanda@Terra:~/ShellBasico$ ./ShellBasico test8.sh
$ █
```

### 3.2. PSTree

Para a PSTree é apresentado sua versão obtida através da implementação deste TP e a também a gerada pelo sistema, para fins comparativos. A PSTree do sistema também contém dados sobre as threads e, apesar de existir uma opção (-T) para remover tal informação, ela não está disponível na máquina de nenhum dos integrantes deste grupo. Portanto a PSTree do sistema foi impressa com as threads e com a opção -n, para ordenar pelo pid e facilitar a comparação com a PSTree criada no TP. Como a PSTree gerada é grande, aqui é exibida apenas uma parte da mesma.

```
fernanda@Terra:~$ pstree -n
systemd--systemd-journal
systemd--systemd-udev
systemd--systemd-timesyn--{sd-resolve}
accounts-daemon--{gmain}
accounts-daemon--{gdbus}
acpid
bluetoothd
avahi-daemon--avahi-daemon
cron
dbus-daemon
ModemManager--{gmain}
ModemManager--{gdbus}
thermald--{thermald}
NetworkManager--{gmain}
NetworkManager--{gdbus}
NetworkManager--dnsmasq
NetworkManager--dhclient
systemd-logind
snapd--6*[{snapd}]
smartd
rsyslogd--{in:imuxsock}
rsyslogd--{in:imklog}
rsyslogd--{rs:main Q:Reg}
whoopsie--{gmain}
whoopsie--{gdbus}
polkitd--{gmain}
polkitd--{gdbus}
lightdm--{gmain}
lightdm--{gdbus}
Xorg--{Xorg}
lightdm--{gmain}
lightdm--{gdbus}
upstart--sd_dummy--{sd_dummy}
upstart--sd_dummy--{threaded-ml}
upstart--sd_cicero--sd_cicero
upstart--sd_cicero--{sd_cicero}
upstart--sd_cicero--{threaded-ml}
upstart--sd_espeak--3*[{sd_espeak}]
upstart--sd_espeak--{threaded-ml}
upstart--sd_generic--{sd_generic}
upstart--sd_generic--{threaded-ml}

fernanda@Terra:~/ShellBasico$ ./myPSTree
(systemd)
(systemd-journal)
(systemd-udev)
(systemd-timesyn)
(accounts-daemon)
(acpid)
(bluetoothd)
(avahi-daemon)
(avahi-daemon)
(cron)
(dbus-daemon)
(ModemManager)
(thermald)
(NetworkManager)
(dnsmasq)
(dhclient)
(systemd-logind)
(snapd)
(smartd)
(rsyslogd)
(whoopsie)
(polkitd)
(lightdm)
(Xorg)
(lightdm)
(upstart)
(dbus-daemon)
(window-stack-br)
(ibus-daemon)
(ibus-dconf)
(ibus-ui-gtk3)
(ibus-engine-sim)
(upstart-udev-br)
(upstart-dbus-br)
(upstart-dbus-br)
(upstart-file-br)
(gvfsd)
(gvfsd-fuse)
(gpg-agent)
(ibus-x11)
(at-spi-bus-laun)
```

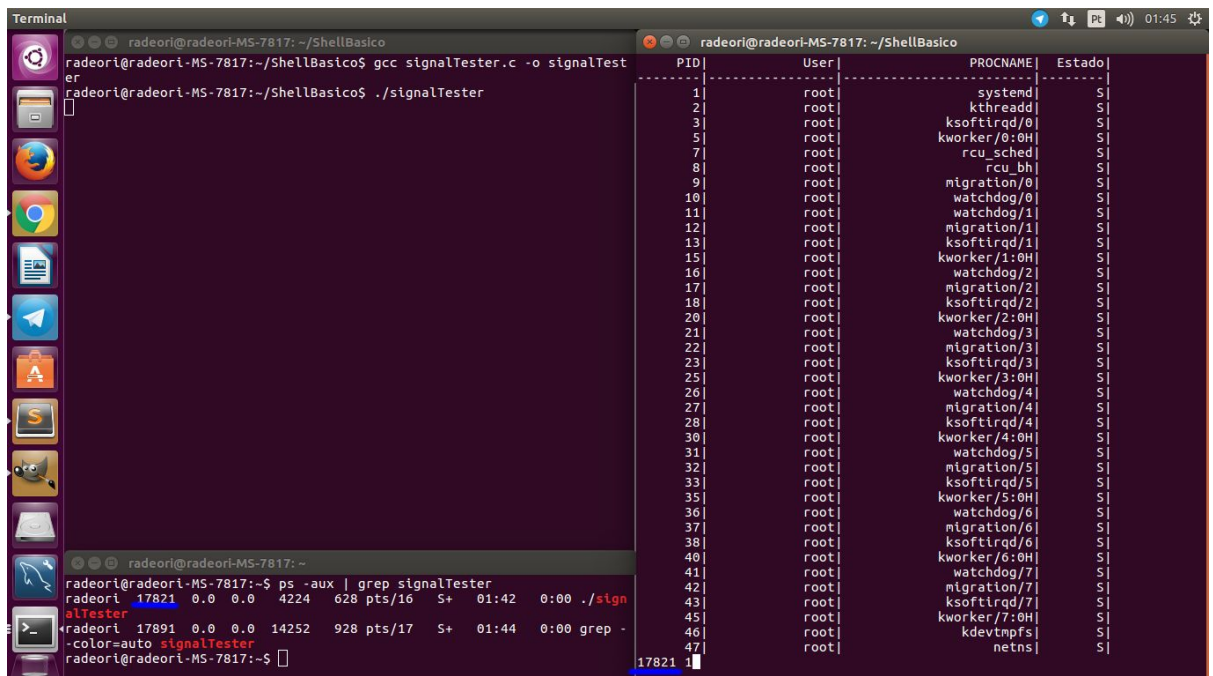


### 3.3. TOPzera

PID	User	PROCNAME	Estado
1	root	systemd	S
2	root	kthreadd	S
3	root	ksoftirqd/0	S
5	root	kworker/0:0H	S
7	root	rcu_sched	S
8	root	rcu_bh	S
9	root	migration/0	S
10	root	watchdog/0	S
11	root	watchdog/1	S
12	root	migration/1	S
13	root	ksoftirqd/1	S
15	root	kworker/1:0H	S
16	root	watchdog/2	S
17	root	migration/2	S
18	root	ksoftirqd/2	S
20	root	kworker/2:0H	S
21	root	watchdog/3	S
22	root	migration/3	S
23	root	ksoftirqd/3	S
25	root	kworker/3:0H	S
26	root	watchdog/4	S
27	root	migration/4	S
28	root	ksoftirqd/4	S
30	root	kworker/4:0H	S
31	root	watchdog/5	S
32	root	migration/5	S
33	root	ksoftirqd/5	S
35	root	kworker/5:0H	S
36	root	watchdog/6	S
37	root	migration/6	S
38	root	ksoftirqd/6	S
40	root	kworker/6:0H	S
41	root	watchdog/7	S
42	root	migration/7	S
43	root	ksoftirqd/7	S
45	root	kworker/7:0H	S
46	root	kdevtmpfs	S
47	root	netns	S
48	root	perf	S

Como foi dito, apesar de ter o cursor no fim da tabela, esse programa simplesmente fica atualizando a tabela mostrada. E, como optou-se por mostrar apenas um número de processos que coubesse na tela, começando do primeiro e sem saltar nenhum, a maior parte do tempo a atualização não é percebida. Dada a natureza do programa, nenhuma função extra foi testada.

### 3.4. Sinais



```
Terminal
radeori@radeori-MS-7817: ~/ShellBasico
radeori@radeori-MS-7817:~/ShellBasico$ gcc signalTester.c -o signalTester
radeori@radeori-MS-7817:~/ShellBasico$ ./signalTester

PID|-----|User|-----|PROCNAME|Estado|
1| 1|root|systemd|S
2| 2|root|kthreadd|S
3| 3|root|ksoftirqd/0|S
5| 5|root|kworker/0:0H|S
7| 7|root|rcu_sched|S
8| 8|root|rcu_bh|S
9| 9|root|migration/0|S
10|10|root|watchdog/0|S
11|11|root|watchdog/1|S
12|12|root|migration/1|S
13|13|root|ksoftirqd/1|S
15|15|root|kworker/1:0H|S
16|16|root|watchdog/2|S
17|17|root|migration/2|S
18|18|root|ksoftirqd/2|S
20|20|root|kworker/2:0H|S
21|21|root|watchdog/3|S
22|22|root|migration/3|S
23|23|root|ksoftirqd/3|S
25|25|root|kworker/3:0H|S
26|26|root|watchdog/4|S
27|27|root|migration/4|S
28|28|root|ksoftirqd/4|S
30|30|root|kworker/4:0H|S
31|31|root|watchdog/5|S
32|32|root|migration/5|S
33|33|root|ksoftirqd/5|S
35|35|root|kworker/5:0H|S
36|36|root|watchdog/6|S
37|37|root|migration/6|S
38|38|root|ksoftirqd/6|S
40|40|root|kworker/6:0H|S
41|41|root|watchdog/7|S
42|42|root|migration/7|S
43|43|root|ksoftirqd/7|S
45|45|root|kworker/7:0H|S
46|46|root|kdevtmpfs|S
47|47|root|netns|S

radeori@radeori-MS-7817:~$ ps -aux | grep signalTester
radeori 17821 0.0 0.0 4224 628 pts/16 S+ 01:42 0:00 ./signalTester
radeori 17891 0.0 0.0 14252 928 pts/17 S+ 01:44 0:00 grep --color=auto signalTester
radeori@radeori-MS-7817:~$ kill -1 17821
```

Na imagem acima, podemos ver como utilizar o programa de Sinais, juntamente com o `ps`, para identificar o `pid` de um processo e enviar um sinal para ele. (e.g. PID:17821 (`signalTester`) - sinal 1 (`SIGHUP`))





#### **4. Conclusão**

A execução deste trabalho prático exigiu conhecimentos e pesquisa a respeito dos processos do Sistema Operacional e de como o Linux fornece informações sobre seus processos. Uma vez compreendido o esqueleto da parte 1 tornou-se simples executá-la, apenas necessitando compreender as chamadas de sistema e como aplicá-las. Nas demais partes do TP o essencial foi compreender como conseguir as informações necessárias, o que foi possível através de consultas a manuais do Linux e pesquisas.

Um dos grandes desafios deste TP foi a linguagem C, que é um obstáculo bastante incômodo com relação a entradas de arquivos e manipulação do stdin (como na parte 4). No mais, o TP foi completado com sucesso e permitiu aprimorar nossos conhecimentos a respeito de processos e o Sistema Operacional.

## 5. Referências

stat. The Open Group Base Specifications Issue 6. Disponível em: <<http://pubs.opengroup.org/onlinepubs/009695399/functions/stat.html>>. Acesso em 16 de abril de 2017.

Linux Programmer's Manual PROC(5). Disponível em <<http://man7.org/linux/man-pages/man5/proc.5.html>>. Acesso em 10 de abril de 2017.

dup2(2) - Linux man page. Disponível em <<https://linux.die.net/man/2/dup2>>. Acesso em 22 de abril de 2017.

open(2) - Linux man page. Disponível em <<https://linux.die.net/man/2/open>>. Acesso em 22 de abril de 2017.

execvp(3) - Linux man page. Disponível em <<https://linux.die.net/man/3/execvp>>. Acesso em 22 de abril de 2017.

fork(2) - Linux man page. Disponível em <<https://linux.die.net/man/2/fork>>. Acesso em 22 de abril de 2017.

PSTREE(1). Disponível em <<http://man7.org/linux/man-pages/man1/pstree.1.html>>

Linux Programmer's Manual GETPID(2). Disponível em <<http://man7.org/linux/man-pages/man2/getpid.2.html>>. Acesso em 23 de abril de 2017.

pipe(2) - Linux man page. Disponível em <<https://linux.die.net/man/2/pipe>>. Acesso em 25 de abril de 2017.

waitpid(3) - Linux man page. Disponível em <<https://linux.die.net/man/3/waitpid>>. Acesso em 25 de abril de 2017.

Linux Programmer's Manual SELECT(2). Disponível em <<http://man7.org/linux/man-pages/man2/select.2.html>>. Acesso em 07 de maio de 2017.