

# TP 1

SHELL + PS(TOP) + SINAIS + MODULOS

Cesar Nascimento & Maximilian Araujo

| DCC605 | Flavio Figueiredo

May 6, 2017

## Especificação do Projeto

O trabalho proposto foi adaptado do material do Professor Ítalo Cunha. Conceitos como Pipe, Estrutura de Processos do Kernel e Sinais, dentro da disciplina de Sistemas Operacionais foram abordados.

O trabalho foi dividido em 4 partes. A seguir um descritivo de cada parte:

1. Desenvolvendo um Shell Básico: Baixar o esqueleto do shell e implementar o código para execução dos comandos como Redirecionamento de entrada/saída; Sequenciamento de Comandos
2. Lendo o /proc/ para fazer um OS-Tree: Imprimir a árvore de processos usando tab para separa-los.
3. Uma TOP Simples (topzera): Modificar o comando PS para imprimir os processos em sequência (remover os tabs). Além disto, alterar o mesmo para identificar o PID do programa, o usuário que está executando o mesmo e o estado do processo. Com isto, imprimir os programas em execução em uma tabela.
4. Sinais: Permitir que o comando TOP envie sinais, ou seja, criar uma função no TOP que envia sinais para um PID. Tal função pode ser apenas um "PID SINAL".

A seguir, para cada parte será apresentado uma breve descrição de como foi feita sua implementação.

## Parte 1: Desenvolvendo um Shell Básico

Case 'exec':

```
execvp(ecmd->argv[0], ecmd->argv);
```

Os comandos da família exec substituem a imagem do processo corrente com a imagem do novo processo. O primeiro argumento da função exec é o nome do arquivo que será executado. A utilização do execvp ocorreu por uma questão de facilidade no entendimento de seu funcionamento. Apenas passamos o arquivo a ser executado por parâmetro e a função se encarrega de criar o processo.

case '>':

case '<':

```
int file = open(rcmd->file, rcmd->mode, S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP |  
S_IROTH | S_IWOTH);
```

```
dup2(file, rcmd->fd);
```

```
close(file);
```

Esta primitiva (dup2) cria uma cópia de um descritor de arquivo existente (oldfd) e fornece um novo descritor (newfd) tendo exatamente as mesmas características que aquele passado como argumento na chamada. Dup2 determina que newfd será a cópia de oldfd, fechando antes newfd se ele já estiver aberto.

case '|':

```
int pipefd[2];

if (pipe(pipefd) == -1) {
    perror("pipe");
    exit(EXIT_FAILURE);
}

int pid = fork1();

if(pid == -1){
    perror("fork");
    exit(EXIT_FAILURE);
}

if(pid == 0){
    close(pipefd[0]);

    close(1);

    dup(pipefd[1]);

    close(pipefd[1]);

    runcmd(pcmd->left);
}

else if(pid != 0){
    close(pipefd[1]);

    close(0);

    dup(pipefd[0]);

    close(pipefd[0]);

    wait(NULL);
}
```

```
    runcmd(pcmd->right);  
}
```

O comando acima descrito cria um pipe, em seguida, `fork1()` cria um processo filho. Após o `fork1()`, cada processo fecha os descritores que não são necessários para o pipe. O pai então utiliza o `dup(pipefd[1])` para se comunicar com o filho. Já o filho utiliza o `dup(pipefd[0])` para se comunicar com pai.

## Parte 2: Lendo o /proc/ para fazer um PS-Tree

Para resolver essa parte do trabalho prático, algumas funções foram criadas. Abaixo seguem breves explicações destas:

`void retornarNomeProcesso(char str[], char *str2)`: função responsável por retornar o nome do processo e armazena-lo na `str2`. Como referência, passa-se uma string `str`, com o formato `/proc/[pid]/task/[pid]/comm`, onde o `[pid]` é o id do processo que se quer descobrir o nome.

`int func(char str[], int espaco)`: função responsável por caminhar na árvore de processos a partir do pai `init` (`pid = 1`), passado como primeiro parâmetro. Para se descobrir os filhos de um processo, faz-se uso do arquivo `/proc/[pid]/task/[pid]/children`. Esse arquivo contém todo os filhos dos processos `[pid]` que se deseja buscar. Para cada processo, a `func` é novamente chamada recursivamente. Para controlar os tabs, a cada chamada da `func` é passado como argumento a quantidade de tabs de acordo com o nível hierárquico que o processo se encontra, através do parâmetro `espaco`.

## Parte 3: Uma TOP Simples (topzera)

Para construção da terceira parte do trabalho, a segunda parte foi utilizada como base. A identificação do programa foi removida e o número de `[pid]` encontrado para construir a árvore foi utilizado como base para encontrar as outras informações especificadas no trabalho.

A seguir serão detalhados o funcionamento de cada função considerada importante para esta parte do trabalho.

`void arquivoNomeProcesso(char *str, char processo[])`: constrói a string responsável em encontrar o arquivo certo para o nome de cada processo utilizando o `[pid]`. A string tem o formato `"/proc/[pid]/task/[pid]/comm"`.

`void arquivoNomeEstado(char *str, char processo[])`: Esta função constrói a string `"/proc/[pid]/status"` que contém diversas informações de um processo. Em específico, nesse arquivo buscamos o estado do processo, que se for lido de forma linear, encontra-se na sexta posição do arquivo. Em seguida a função imprime na tela o estado do processo através do `[pid]` em questão.

`void imprimiOwner(char *str, char processo[], char *owner)`: Para chegarmos ao nome do dono do processo, podemos obter o UID real do programa e traduzir esse número para o login, com ajuda

do arquivo password. Usando a função: `struct passwd *getpwuid(uid_t uid);` que retorna o endereço de uma struct password na qual contém o campo `pw_name` que aponta para o nome do usuário. A função também ficou responsável em imprimir esse nome na tabela do top.

`void arquivoFilhos(char *str, char processo[]):` função já havia sido utilizada para construir a árvore da parte dois do trabalho. Os pids filhos de cada processo foram retirados do arquivo children. Esta função contrai o caminho para esse arquivo. `"/proc/[pid_pai]/task/[pid_pai]/children"`. O primeiro `pid_pai` é o 1.

A função `"int func(char str[], int *numProc)"` e `"void retornarNomeProcesso(char str[], char *str2)"` são omitidas nesta parte da documentação porque já foram abordadas na parte do dois do trabalho.

## Parte 4: Sinais

Para desenvolver esta parte do trabalho, utilizamos a biblioteca `signal.h` dentre as bibliotecas `c`. Com ela foi possível ler uma variável do tipo `pid_t` e utilizar a função `kill(pid_t pid, int sinal)` da biblioteca que faz o que foi requisitado na especificação. A função enviar um sinal `int` para um programa com `pid` determinado.

## Decisões de implementação

Preferimos não fazer a parte 5, que no caso era ponto extra. Além disso, optamos por não implementar as atualizações de 1 em 1 segundo para simplificar a quarta parte do trabalho.

## Testes

Parte 2:

```
cesar@cesar-Inspiron-7437:~/Documentos/so-tp1$ ./mysys
systemd
systemd-journal
systemd-udev
systemd-timesyn
avahi-daemon
avahi-daemon
snapd
accounts-daemon
cron
rsyslogd
thermald
whoopsie
bluetoothd
ModemManager
cgmanager
acpid
dbus-daemon
NetworkManager
dhclient
dnsmasq
systemd-logind
cupsd
cups-browsed
polkitd
irqbalance
systemd-resolve
lightdm
Xorg
lightdm
upstart
xbrlapi
upstart-udev-br
sleep
sh
url-dispatcher
window-stack-br
upstart-dbus-br
upstart-dbus-br
upstart-file-br
hud-service
sleep
indicator-messa
indicator-bluet
indicator-power
indicator-datet
indicator-keybo
indicator-sound
indicator-print
indicator-sessi
agetty
wpa_supplicant
upowerd
rtkit-daemon
colord
systemd
(sd-pam)
dbus-daemon
dconf-service
gvfsd
gvfsd-fuse
bamfdaemon
at-spi-bus-laun
dbus-daemon
at-spi2-registr
unity-settings-
syndaeon
gnome-session-b
nautilus
gnome-software
nm-applet
indicator-appli
unity-fallback-
gnome-do
polkit-gnome-au
zeitgeist-datah
update-notifier
deja-dup-monito
ibus-daemon
ibus-dconf
ibus-ui-gtk3
ibus-engine-sim
ibus-x11
pulseaudio
compiz
unity-panel-ser
gvfs-udisks2-vo
evolution-sourc
gvfs-gphoto2-vo
gvfs-mtp-volume
gvfs-afc-volume
gvfs-goa-volume
evolution-calen
evolution-calen
evolution-calen
gvfsd-trash
evolution-addre
evolution-addre
gvfsd-metadata
gconfd-2
sh
zeitgeist-daemo
zeitgeist-fts
gnome-terminal-
bash
bash
mysys
update-manager
oosplash
gnome-keyring-d
udisksd
unity-greeter-s
fwupd
aptd
packagekitd
cesar@cesar-Inspiron-7437:~/Documentos/so-tp1$
```

Parte 3: Impressão dos 20 primeiros processos em execução

```
maxaraujo@maxaraujo:~/so-tp1$ ./topzera
```

PID	User	PROCNAME	Estado
274	root	systemd-journal	S
285	root	systemd-udev	S
724	root	systemd-timesyn	S
861	root	rsyslogd	S
866	root	cron	S
869	root	systemd-logind	S
870	root	thermald	S
874	root	acpid	S
880	root	avahi-daemon	S
912	root	avahi-daemon	S
891	root	ModemManager	S
893	root	dbus-daemon	S
945	root	NetworkManager	S
1331	root	dhclient	S
1343	root	dnsmasq	S
947	whoopsie	whoopsie	S
948	root	accounts-daemon	S
953	root	snapd	S
1018	root	polkitd	S
1028	root	irqbalance	S

O campo no final da tabela é a espera pela leitura do pid e um sinal que é especificado na quarta parte do trabalho.

#### Parte 4:

Para imprimir este teste, utilizamos o TOP original para visualizar o PID do processo SIGNALTESTER.

```
maxaraujo@maxaraujo: ~/so-tp1
maxaraujo@maxaraujo:~/so-tp1$ ./signaltester
getmarks.py myps myps.x myshell READONLY
topzera.x TPI.docx

maxaraujo@maxaraujo:~/so-tp1$ delete mode 100644 x.txt
maxaraujo@maxaraujo:~/so-tp1$ ./topzera
PID | User | PROCNAME | Estado |
-----|-----|-----|-----|
1274 | root | systemd-journal | S
1285 | root | systemd-udev | S
1724 | root | systemd-timesyn | S
1861 | root | rsyslogd | S
1866 | root | crcon | S
1869 | root | systemd-logind | S
1870 | root | thermald | S
1874 | root | acpid | S
1880 | root | avahi-daemon | S
1912 | root | avahi-daemon | S
1891 | root | ModemManager | S
1893 | root | dbus-daemon | S
1945 | root | NetworkManager | S
11331 | root | dhclient | S
11343 | root | dnsmasq | S
1947 | root | whoopsie | S
1948 | root | accounts-daemon | S
1953 | root | snapd | S
11018 | root | polkitd | S
11028 | root | irqbalance | S

maxaraujo@maxaraujo:~/so-tp1$ top - 20:47:53 up 1 day, 1:41, 1 user, load average: 0,30, 0,34, 0,41
Tarefas: 235 total, 1 executando, 234 dormindo, 0 parado, 0 zumbi
NCPU(s): 8,5 us, 4,1 sy, 0,0 ni, 87,5 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
KlB Mem : 3909588 total, 440596 free, 2036604 used, 1432388 buff/cache
KlB Swap: 6070268 total, 6056336 free, 13932 used, 974116 avail Mem

PID USUARIO PR NI VIRT RES SHR %CPU %MEM TIME+ COMMAND
2160 maxarau+ 20 0 364196 9072 8016 S 0,0 0,2 0:00.05 gvfsd-trash
2205 maxarau+ 20 0 196480 5492 4820 S 0,0 0,1 0:00.02 gvfsd-metadatas
2224 maxarau+ 20 0 442080 20848 14008 S 0,0 0,5 0:02.36 zeitgeist-datas
2231 maxarau+ 20 0 4508 712 632 S 0,0 0,0 0:00.00 sh
2235 maxarau+ 20 0 423656 9408 7852 S 0,0 0,2 0:00.20 zeitgeist-daemo
2242 maxarau+ 20 0 324312 18576 13152 S 0,0 0,5 0:00.25 zeitgeist-fts
2303 maxarau+ 20 0 520088 25888 21920 S 0,0 0,7 0:02.31 update-notifier
2333 maxarau+ 20 0 441816 9080 8096 S 0,0 0,2 0:04.07 deja-dup-monito
2446 maxarau+ 20 0 61600 5748 5256 S 0,0 0,1 0:00.12 gconfd-2
7500 maxarau+ 20 0 364636 8892 7932 S 0,0 0,2 0:00.03 gvfsd-network
7525 maxarau+ 20 0 362012 6860 6104 S 0,0 0,2 0:00.03 gvfsd-dnssd
9622 www-data 20 0 325536 8868 1844 S 0,0 0,2 0:00.00 apache2
9624 www-data 20 0 325536 8868 1844 S 0,0 0,2 0:00.00 apache2
9627 www-data 20 0 325536 8868 1844 S 0,0 0,2 0:00.00 apache2
9629 www-data 20 0 325536 8868 1844 S 0,0 0,2 0:00.00 apache2
9651 root 20 0 274820 9496 8268 S 0,0 0,2 0:00.02 cups-browsed
14409 root 20 0 0 0 0 S 0,0 0,0 0:00.50 kworker/3:1
14446 maxarau+ 20 0 659492 40452 28500 S 0,0 1,0 0:05.76 gnome-terminal-
14453 maxarau+ 20 0 22796 5264 3300 S 0,0 0,1 0:00.05 bash
14472 root 20 0 0 0 0 S 0,0 0,0 0:00.58 kworker/0:2
14509 maxarau+ 20 0 134136 5740 4652 S 0,0 0,1 0:00.08 oosplash
14528 maxarau+ 20 0 1566980 157844 88392 S 0,0 4,0 2:05.33 soffice.bin
14823 maxarau+ 20 0 652316 39184 29532 S 0,0 1,0 0:25.53 gedit
15476 root 20 0 0 0 0 S 0,0 0,0 0:00.95 kworker/2:0
15690 root 20 0 0 0 0 S 0,0 0,0 0:00.37 kworker/1:1
16016 root 20 0 0 0 0 S 0,0 0,0 0:01.49 kworker/u8:0
16037 root 20 0 0 0 0 S 0,0 0,0 0:00.35 kworker/2:2
16163 maxarau+ 20 0 647840 18284 14012 S 0,0 0,5 0:00.34 unity-scope-ho
16175 maxarau+ 20 0 587644 25988 17824 S 0,0 0,7 0:00.29 unity-scope-loa
16176 maxarau+ 20 0 647604 14716 12308 S 0,0 0,4 0:00.15 unity-files-dae
16555 root 20 0 0 0 0 S 0,0 0,0 0:00.00 kworker/1:2
16721 root 20 0 0 0 0 S 0,0 0,0 0:00.37 kworker/u8:2
16737 root 20 0 0 0 0 S 0,0 0,0 0:00.00 kworker/3:2
16773 maxarau+ 20 0 12808 1840 1700 S 0,0 0,0 0:00.00 topzera
16776 root 20 0 0 0 0 S 0,0 0,0 0:00.22 kworker/u8:1
16814 root 20 0 0 0 0 S 0,0 0,0 0:00.04 kworker/0:0
16869 root 20 0 0 0 0 S 0,0 0,0 0:00.00 kworker/2:3
16880 maxarau+ 20 0 22796 5360 3392 S 0,0 0,1 0:00.05 bash
16893 maxarau+ 20 0 4224 632 556 S 0,0 0,0 0:00.00 signaltester
16901 maxarau+ 20 0 22796 5220 3252 S 0,0 0,1 0:00.05 bash
16945 root 20 0 0 0 0 S 0,0 0,0 0:00.00 kworker/2:1
16956 maxarau+ 20 0 41988 3836 3124 R 0,0 0,1 0:02.56 top
16960 root 20 0 0 0 0 S 0,0 0,0 0:00.02 kworker/u8:3
```

```
maxaraujo@maxaraujo:~/so-tp1
maxaraujo@maxaraujo:~/so-tp1$ ./signaltester
SIG 1
maxaraujo@maxaraujo:~/so-tp1$ top - 20:49:45 up 1 day, 1:43, 1 user, load average: 0,08, 0,25, 0,36
Tarefas: 235 total, 2 executando, 233 dormindo, 0 parado, 0 zumbi
NCPU(s): 7,3 us, 4,4 sy, 0,0 ni, 88,3 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
KlB Mem : 3909588 total, 466756 free, 2058516 used, 1384316 buff/cache
KlB Swap: 6070268 total, 6056344 free, 13924 used, 1001436 avail Mem

PID USUARIO PR NI VIRT RES SHR %CPU %MEM TIME+ COMMAND
2128 maxarau+ 20 0 697976 19860 17076 S 0,0 0,5 0:00.05 evolution-adre
2131 maxarau+ 20 0 807736 50944 14416 S 0,0 1,3 0:04.39 evolution-calen
2138 maxarau+ 20 0 912752 21024 16392 S 0,0 0,5 0:04.40 evolution-adre
2160 maxarau+ 20 0 364196 9080 8016 S 0,0 0,2 0:00.07 gvfsd-trash
2205 maxarau+ 20 0 196480 5492 4820 S 0,0 0,1 0:00.03 gvfsd-metadatas
2224 maxarau+ 20 0 442080 20848 14008 S 0,0 0,5 0:02.42 zeitgeist-datas
2231 maxarau+ 20 0 4508 712 632 S 0,0 0,0 0:00.00 sh
2235 maxarau+ 20 0 423656 9408 7852 S 0,0 0,2 0:00.21 zeitgeist-daemo
2242 maxarau+ 20 0 324312 18576 13152 S 0,0 0,5 0:00.26 zeitgeist-fts
2303 maxarau+ 20 0 520088 25888 21920 S 0,0 0,7 0:02.32 update-notifier
2333 maxarau+ 20 0 441816 9080 8096 S 0,0 0,2 0:04.08 deja-dup-monito
2446 maxarau+ 20 0 61600 5748 5256 S 0,0 0,1 0:00.12 gconfd-2
7500 maxarau+ 20 0 364636 8892 7932 S 0,0 0,2 0:00.04 gvfsd-network
7525 maxarau+ 20 0 362012 6860 6104 S 0,0 0,2 0:00.03 gvfsd-dnssd
9622 www-data 20 0 325536 8868 1844 S 0,0 0,2 0:00.00 apache2
9624 www-data 20 0 325536 8868 1844 S 0,0 0,2 0:00.00 apache2
9627 www-data 20 0 325536 8868 1844 S 0,0 0,2 0:00.00 apache2
9629 www-data 20 0 325536 8868 1844 S 0,0 0,2 0:00.00 apache2
9651 root 20 0 274820 9496 8268 S 0,0 0,2 0:00.03 cups-browsed
14409 root 20 0 0 0 0 S 0,0 0,0 0:00.51 kworker/3:1
14453 maxarau+ 20 0 22796 5328 3364 S 0,0 0,1 0:00.05 bash
14472 root 20 0 0 0 0 S 0,0 0,0 0:00.58 kworker/0:2
14509 maxarau+ 20 0 134136 5740 4652 S 0,0 0,1 0:00.08 oosplash
14528 maxarau+ 20 0 1566980 157844 88392 S 0,0 4,0 2:05.37 soffice.bin
14823 maxarau+ 20 0 652316 39184 29532 S 0,0 1,0 0:25.57 gedit
15476 root 20 0 0 0 0 S 0,0 0,0 0:00.95 kworker/2:0
15690 root 20 0 0 0 0 S 0,0 0,0 0:00.38 kworker/1:1
16016 root 20 0 0 0 0 S 0,0 0,0 0:01.49 kworker/u8:0
16163 maxarau+ 20 0 647840 18284 14012 S 0,0 0,5 0:00.36 unity-scope-ho
16175 maxarau+ 20 0 587644 25988 17824 S 0,0 0,7 0:00.30 unity-scope-loa
16176 maxarau+ 20 0 647604 14716 12308 S 0,0 0,4 0:00.15 unity-files-dae
16555 root 20 0 0 0 0 S 0,0 0,0 0:00.00 kworker/1:2
16721 root 20 0 0 0 0 S 0,0 0,0 0:00.51 kworker/u8:2
16737 root 20 0 0 0 0 S 0,0 0,0 0:00.00 kworker/3:2
16776 root 20 0 0 0 0 S 0,0 0,0 0:00.22 kworker/u8:1
16814 root 20 0 0 0 0 S 0,0 0,0 0:00.05 kworker/0:0
16869 root 20 0 0 0 0 S 0,0 0,0 0:00.00 kworker/2:3
16880 maxarau+ 20 0 22796 5360 3392 S 0,0 0,1 0:00.05 bash
16901 maxarau+ 20 0 22796 5220 3252 S 0,0 0,1 0:00.05 bash
16945 root 20 0 0 0 0 S 0,0 0,0 0:00.00 kworker/2:1
16956 root 20 0 0 0 0 S 0,0 0,0 0:00.06 kworker/u8:3
17008 root 20 0 0 0 0 S 0,0 0,0 0:00.00 kworker/3:0
17030 root 20 0 15676 1136 1000 S 0,0 0,0 0:00.01 systemd-hostnam
```

O processo signaltester é o quarto processo de baixo para cima no terminal a direita. Enquanto isso o topzera aguarda a leitura de um pid e um sinal



## Conclusão

Foi possível implementar todas as especificações mais importantes do trabalho, possibilitando assim o ganho de conhecimento sobre como funciona algumas funções do Linux.

## Referências bibliográficas

<https://linux.die.net/man/3/execvp>

<http://www.dca.ufrn.br/~adelardo/cursos/DCA409/node22.html>

<https://linux.die.net/man/2/pipe>

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