Comunicação entre processos:

O Linux possui diversas formas de realizar comunicação entre processos, também conhecido com IPC. Estas formas de comunicação se referem a processos em execução na mesma máquina, o que não inclui sockets ou RPC (são para comunicação entre máquinas distintas).
Dentre as técnicas de comunicação entre processos fornecidas pelo Linux, convém citar:
a) por memória compartilhada: através de diretivas shm* pode-se disponibilizar uma área de memória comum à vários processos. Simples: quando um processo quer enviar dados ao outro, põe na memória.
b) pipes: um método muito interessante onde se cria um canal bidirecional na tabela de descritores, como se fosse arquivo, mas não é (e não tem I/O). Quando você faz um ls -la | sort está usando pipe!
c) sinais: quando um processo quer sinalizar algo a outro processo. Ao contrário das demais nesta não se envia dados mas apenas um valor numérico.
Existem outras formas de comunicação como semáforos, fila de mensagens e até mesmo morte de filho, que é considerada uma comunicação. Se você digitar ipcs na linha de comando irá ver os canais de comunicação atualmente em uso (não sinais).
Texto retirado do Artigo: http://www.vivaolinux.com.br/artigo/Sinais-em-Linux/?pagina=1

Comandos:

Listar os processos do terminal

```
# ps
```

Listar os processos de todos os terminais

```
# ps -a
# ps a (exibe os processos mais detalhes)
```

LIstar os processos dos usuários

```
# ps -u
# ps -au
```

Lista todos os processos do Sistema

```
# ps -A
# ps -ef
# ps -eF
# ps -ely
# ps ax
# ps aux (exibe uma coluna com o nome do usuário que iniciou o processo)
```

Exibe todos os processos em forma de árvore

```
# pstree
# ps -ejH
# ps axfj
```

Obs: teste os vários parâmetros do comando pstree:

-a
Mostra opções passadas na linha de comando.

-c
Mostra toda a estrutura (inclusive sub-processos do processo pai).

-G
Usa caracteres gráficos no desenho da árvore de processos.

-h
Destaca o processo atual e seus antecessores.

-H [pid]
Destaca o processo especificado.
-l
Não faz quebra de linha
-n
Classifica pelo número PID ao invés do nome.
-p
Mostra o número PID entre parênteses após o nome do processo.
-u
Mostra também o dono do processo.
-U
Usa o conjunto de caracteres Unicode para o desenho da árvore.

Exibe informações sobre Threads

```
# ps -eLf
# ps axms
```

Exibe informações de segurança dos processos

```
# ps -eo euser,ruser,suser,fuser,f,comm,label
# ps axZ
# ps -eM
```

Exibe todos os processos que estão sendo executados com permissões de root (real e com ID)

```
#  ps -U root -u root u
```

Especificando as colunas para visualização:
Exemplos:
Apenas o Numero do processo e o comando que o gerou

```
# ps ao PID,COMMAND
```

Apenas o numero do processo, o comando que gerou o pid, o uso da CPU e da memória

```
# ps ao pid,%cpu,%mem,command
```

Outros exemplos tirados do manual do ps:

```
# ps -eo pid,tid,class,rtprio,ni,pri,psr,pcpu,stat,wchan:14,comm
# ps axo stat,euid,ruid,tty,tpgid,sess,pgrp,ppid,pid,pcpu,comm
# ps -eo pid,tt,user,fname,tmout,f,wchan
```

Exibir processos e prioridade do processo (Coluna NI)

```
# ps l
# ps -l
# ps -Al
# ps -el
```

Significado das Principais colunas do comando TOP:
USER - Nome do usuário dono do processo.
UID - Número de identificação do usuário dono do processo.
PID - Número de identificação do processo.
PPID - Número de identificação do processo pai de cada tarefa.
PRI - Número de prioridade da tarefa. (Números altos são prioridades baixas).
NI - Valor preciso da prioridade da tarefa.
%CPU - O consumo de processamento do processo.

%MEM - O consumo de memória do processo.
SIZE - Tamanho do código da tarefa em kilobytes.
RSS - Soma total da memória física usada pelo processo, em kilobytes.
WCHAN - Endereço ou nome da função do kernel da tarefa que está atualmente suspensa.
STAT - Estado do processo: S - Suspenso, R - em Execução, T - Interrompido, Z - Terminado, etc.
TTY - Terminal onde são executados os processos.
TIME - Tempo total da CPU usado pelo processo desde que foi iniciado.
COMMAND - Nome do comando do processo.

Entendendo os Sinais:

Sinais são usados quando se deseja enviar uma mensagem para um processo, mas esta mensagem não é um texto, mas sim um código cujo significado é pré determinado. Antes de mais nada, para dar o melhor exemplo possível, quando se executa um kill 7000 está se enviando para o processo 7000 o sinal de código 15 que significa algo como "Caro processo, queria gentilmente terminar a sua execução".
Como o sinal 15 significa terminar, o processo irá terminar a sua execução. Como todos já devem saber, se o "gentil" processo recusar-se a terminar, posso forçar com um kill -9 7000, onde envio para ele o sinal 9 que significa algo como: "Morra!" (uma vez disse em um curso que com sinal 15 eu dou a arma para o processo esperando que ele aperte o gatilho, depois de escrever o seu testamento, fechar suas finanças etc. Com o -9 é um tiro na testa sem aviso).

Principais Sinais:

Sinal 1: Significa reinício do programa. Este sinal é chamado de SIGHUP.

Sinal 2: Sinal chamado de SIGINT. Causa uma interrupção no programa. Falando em termos práticos, é um sinal 2 que o programa recebe quando se pressiona Control+C.

Sinal 15: esta é a solicitação de morte, chamada de SIGTERM. Ao receber um sinal 15 o processo deveria preparar-se para terminar, fazendo "seus últimos pedidos" e ele mesmo encerrando normalmente sua execução.

Sinal 9: SIGKILL. Este é a morte indefensável. Não pode ser mascarado, ou seja, o programador não consegue substituir a rotina de tratamento do Sistema Operacional que simplesmente tira o processo da fila de prontos. Deve ser usado em último caso, pois um sinal 15 é mais elegante por dar a chance ao processo de se preparar para sua morte.

Sinal 20: SIGTSTOP. Este sinal de STOP faz com que o processo interrompa a sua execução. Veja, ele não termina, apenas interrompe.

Sinal 18: SIGCONT. Este sinal de CONTINUAR, faz com que o processo que foi interrompido pelo Sinal 20, continue seu processamento.

Fonte: http://www.vivaolinux.com.br/artigo/Sinais-em-Linux/?pagina=2

Enviando Sinais para os Processos:

Vamos utilizar dois comandos. O KILL e o KILLALL.
A diferença entre estes dois comandos é que o KILL envia sinais para os números dos processos e o

KILLALL envia para nomes.

Exemplo:
Abro o vim em um terminal. E suponho que ele está travado. Quero enviar um sinal para este processo. Com o comando KILL precisariamos de dois passos:

1º Vou para outro terminal e executo o comando ps a
2º Anoto o PID do VIM
3º kill -15 o número do PID do vim.
Exemplo: kill -15 4567

Usando o KILLALL, iríamos para outro terminal e fariamos apenas:
```
# killall -15 vim
```
Obs: Porém se tiver mais programas vim abertos, todos receberão o mesmo sinal.

Para os exemplos abaixo, vamos supor que estamos enviando sinais para o processo 4567 - backup.sh

Reiniciando o processo
```
# kill -SIGHUP 4567
# kill -1 4567
```

Interrompendo o programa
```
# kill -SIGINT 4567
# kill -2 4567
```

Encerrando um programa (educadamente)
```
# kill -SIGTERM 4567
# kill -15 4567
```

Matando um processo (LUNGA MODE=ON)
```
#  kill -SIGKILL 4567
# kill -9 4567
```

Parando (congelando) um processo na memória
```
# kill -SIGSTOP 4567
# kill -20 4567
```

Retomando o processo (congelado no tópico anterior)
```
# kill -SIGCONT 4567
# kill -18 4567
```

Primeiro Plano (FOREGROUND) e Segundo Plano (BACKGROUND)
Você sabia que no mesmo terminal posso iniciar vários processos e deixá-lo em background. Ainda tenho a opção de deixá-lo parado ou executando.

Para termos um bom exemplo de programas executando em segundo plano, vamos criar um pequeno script que terá a função apenas de contar de 1 até infinito.

Passos para escrever o script:
```
vim conta.sh
#!/bin/bash
while :  ;
```

```
do
   let i++
done
```

Pressione a tecla ESC e digite :x

Tornando o script executável
```
# chmod +x conta.sh
```

Executando o script:
```
# ./conta.sh
```

Parece que ele não está fazendo nada, mas ele está contando de 1 até infinito. Para cancelar o programa, basta digitar CTRL+C
OBs: CTRL+C é o mesmo que kill -SIGINT ou kill -2

Iniciando um processo.
Vamos colocar em segundo plano os programas vim /etc/services, nano, man ls, cat /etc/services | more,

```
# vim /etc/services
```
Pressione CTRL+Z
O processo vai para segundo plano.

```
# man ls
```
Pressione CTRL+Z
O processo vai para segundo plano.

```
# cat /etc/services | less
```
Pressione CTRL+Z
O processo vai para segundo plano.

```
# cat /etc/services | more
```
Pressione CTRL+Z
O processo vai para segundo plano.

E por ultimo vamos chamar o nosso programa, o conta.sh
```
# ./conta.sh
```

Exibindo os processos que estão em segundo plano (background):
```
# jobs
```
Se você seguiu meus exemplos acima, você terá esta lista:

```
[1]   Stopped                 vim /etc/services
[2]   Stopped                 man ls
[3]   Stopped                 cat /etc/services | less
[4]-  Stopped                 cat /etc/services | more
[5]+  Stopped                 ./conta.sh
```

Ativando programas que estão em Segundo Plano
Na listagem acima, o processo conta.sh é um programa que está com o status Stopped. Mas sabemos que este programa tem a função de contar. Para ativar este programa você tem duas

maneiras possíveis:
A primeira é mandar o sinal SIGCONT para ele. A outra, bem mais fácil é fazer o seguite:

```
# bg 5
```

Ao executar o comando jobs, veja como ficou nossa lista:

```
[1]   Stopped                 vim /etc/services
[2]   Stopped                 man ls
[3]   Stopped                 cat /etc/services | less
[4]-  Stopped                 cat /etc/services | more
[5]+  Running                 ./conta.sh &
```

Note que agora o conta.sh status Running (executando) e apareceu um novo símbolo ao lado do nome do programa: &

O Símbolo & avisa que o programa está sendo executado em segundo plano.

Se você desejar parar novamente o conta.sh, basta digitar:

```
# killall -SIGSTOP conta.sh
```

Mais uma vez, vamos ver como ficou através do comando jobs

```
[1]   Stopped                 vim /etc/services
[2]   Stopped                 man ls
[3]   Stopped                 cat /etc/services | less
[4]-  Stopped                 cat /etc/services | more
[5]+  Running                 ./conta.sh &
```

Outra maneira de ativá-lo novamnete é enviando um sinal através do comando KILL

```
# killall -SIGCONT conta.sh
```

Trazendo para primeiro plano (foreground) programas que estão em segundo plano (background).

De posse da lista de processos que estão em segundo plano, basta você digitar o comando fg e número do programa nesta lista e você terá o programa de volta em primeiro plano.
Observe a lista do comando jobs:

```
[1]   Stopped                 vim /etc/services
[2]   Stopped                 man ls
[3]   Stopped                 cat /etc/services | less
[4]-  Stopped                 cat /etc/services | more
[5]+  Running                 ./conta.sh &
```

```
# fg 1 (trará o vim /etc/services para primeiro plano)
```

Para voltar para background digite mais uma vez CTRL+Z.

Emita o comando jobs mais uma vez:

```
# jobs
```

Veja o resultado da lista:

```
[1]+  Stopped                 vim /etc/services
[2]   Stopped                 man ls
[3]   Stopped                 cat /etc/services | less
[4]   Stopped                 cat /etc/services | more
[5]-  Stopped                 ./conta.sh
```

Notou que agora o sinal de + saiu da posição 5 e foi para a posição 1? E o sinal de menos da posição 4 para a posição 5?
Estes sinais demonstram os dois últimos programas que foram colocados em backgroud. Caso você digite fg sem parametros, o programa que virá para primeiro plano é o que tem o

sinal + e o próximo da fila que era o - torna-se + e assim por diante.

Colocando programas em segundo plano diretamente na linha de comando.
./conta.sh &
O Uso do & já coloca o programa em segundo plano em execução.
Repita este comando várias vezes, e veja um resultado semelhante a este:

```
[1]+  Stopped                 vim /etc/services
[2]   Stopped                 man ls
[3]   Stopped                 cat /etc/services | less
[4]   Stopped                 cat /etc/services | more
[5]-  Stopped                 ./conta.sh
[6]   Running                 ./conta.sh &
[7]   Running                 ./conta.sh &
[8]   Running                 ./conta.sh &
[9]   Running                 ./conta.sh &
[10]   Running                 ./conta.sh &
[11]   Running                 ./conta.sh &
```

Obs: Toda vez que você chama um programa já em segundo plano, o Shell exibe na linha abaixo o número do processo no qual ele está submetido.

[dailson@server tmp]$ ./conta.sh &
[11] 12642

Outra observação importante é que quando executamos um programa em segundo plano e ele acaba sua execução o Shell avisa emitindo uma mensagem na própria linha de comando.
Teste você mesmo. Execute o comando que atualiza o banco de dados do comando locate.
 # updatedb &
[12] 12649

Fique pressionando enter algumas vezes. Em determinado momento o Shell emite a seguinte mensagem:
Done [12] 12649
Informando que o processo já foi concluido.

Prioridade de Processos no Linux

Os processos no Linux (no espaço do usuário) vão de -20 a +19.
A prioridade padrão de qualquer processo iniciado pelo usuário é 0.
Numeros negativos, indicam maior prioridade e numeros positivos, prioridades baixas.

Para visualizar a prioridade dos processos do sistema, utilize o seguinte comando:

 # ps al

O Resultado é parecido com este:

```
F   UID   PID  PPID PRI  NI    VSZ  RSS WCHAN  STAT TTY        TIME
COMMAND
4     0  3988     1  17   0   3016  400 ?      Ss+  tty1       0:00
/sbin/mingetty tty1
4     0  3989     1  16   0   2708  400 ?      Ss+  tty2       0:00
/sbin/mingetty tty2
4     0  3990     1  17   0   2156  400 ?      Ss+  tty3       0:00
/sbin/mingetty tty3
4     0  3991     1  17   0   2204  400 ?      Ss+  tty4       0:00
/sbin/mingetty tty4
4     0  3992     1  17   0   2044  400 ?      Ss+  tty5       0:00
/sbin/mingetty tty5
4     0  3993     1  16   0   2884  400 ?      Ss+  tty6       0:00
/sbin/mingetty tty6
0   525 16849 21405  17   0   3724  636 -      R+   pts/0      0:00
ps al
0   525 21405 21404  15   0   6168 1472 wait   Ss   pts/0      0:00
-bash
```

Ou se preferir, você pode filtrar com o comando que aprendemos na aula de ps:
`# ps ao pid,ni,command` (assim só aparece o pid, a coluna NICE e o comando que gerou).

Observe a coluna: NI. Ela tem os valores que falamos.
Note que todo processo iniciado pelo usuário tem prioridade 0.
Vamos ver agora a listagem completo com os processos do sitema.

```
# ps -axl | more
```

```
F   UID   PID  PPID PRI  NI    VSZ  RSS WCHAN  STAT TTY        TIME
COMMAND
4     0     1     0  16   0   2512  548 ?      S    ?          0:00
init [3]
1     0     2     1 -100  -     0    0 migrat S    ?          0:03
[migration/0]
1     0     3     1  34  19     0    0 ksofti SN   ?          0:00
[ksoftirqd/0]
1     0     4     1 -100  -     0    0 migrat S    ?          0:02
[migration/1]
1     0     5     1  34  19     0    0 ksofti SN   ?          0:00
[ksoftirqd/1]
1     0     6     1 -100  -     0    0 migrat S    ?          0:02
[migration/2]
1     0     7     1  34  19     0    0 ksofti SN   ?          0:00
[ksoftirqd/2]
1     0     8     1 -100  -     0    0 migrat S    ?          0:02
[migration/3]
1     0     9     1  34  19     0    0 ksofti SN   ?          0:00
[ksoftirqd/3]
1     0    10     1   5 -10     0    0 worker S<   ?          0:00
[events/0]
1     0    11     1   5 -10     0    0 worker S<   ?          0:00
[events/1]
```

```
1     0    12     1   5 -10     0    0 worker S<   ?          0:00
[events/2]
1     0    13     1   5 -10     0    0 worker S<   ?          0:00
[events/3]
1     0    14     1   6 -10     0    0 worker S<   ?          0:00
[khelper]
1     0    15     1   5 -10     0    0 worker S<   ?          0:00
[kthread]
1     0    16    15  15 -10     0    0 worker S<   ?          0:00
[kacpid]
1     0    94    15   5 -10     0    0 worker S<   ?          0:00
[kblockd/0]
1     0    95    15   5 -10     0    0 worker S<   ?          0:00
[kblockd/1]
1     0    96    15   5 -10     0    0 worker S<   ?          0:00
[kblockd/2]
1     0    97    15   5 -10     0    0 worker S<   ?          0:00
[kblockd/3]
```

Note na coluna NI que existem processos com prioridades baixíssimas (19) e outros com prioridade -10.

Iniciando processos fora da prioridade Padrão

Vamos aprender a usar o comando nice

```
# nice -n prioridade processo
```

Exemplo: Vamos usar o nosso programa conta.sh como exemplo. Vamos iniciá-lo com a menor prioridade possível e já em segundo plano:

```
# nice -n +19 ./conta.sh &
```

Veja o resultado:

```
# ps -al
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME
CMD
0 R   525 26483 21405 99  99  19 -  1234 -      pts/0    00:00:02
conta.sh
0 R   525 26484 21405  0  76   0 -   878 -      pts/0    00:00:00
ps
```

Agora vamos iniciá-lo com a prioridade -10

```
# nice -n -10 ./conta.sh &
```

Veja o resultado:

```
# ps -al
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME
CMD
0 R   525 26483 21405 99  99 -10 -  1234 -      pts/0    00:00:02
conta.sh
0 R   525 26484 21405  0  76   0 -   878 -      pts/0    00:00:00
```

```
ps
```

Alterando a prioridade de um processo que já está em execução.

Desta vez vamos utilizar o comando renice.

```
# renice -n prioridade processo
```

Vamos usar o exemplo anterior. O conta.sh tem o pid 26483 e o processo tem prioridade -10.
Vamos mudar para -5 (lembre-se que o processo já está em execução)

```
# renice -5 -p 26483
```

Veja o resultado:

```
# ps -al
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 R   525 26483 21405 99  99  -5 -  1234 -      pts/0    00:00:02 conta.sh
0 R   525 26484 21405  0  76   0 -   878 -      pts/0    00:00:00 ps
```

Vamos mudar para 0

```
# renice 0 -p 26483
```

Veja o resultado:

```
# ps -al
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 R   525 26483 21405 99  99   0 -  1234 -      pts/0    00:00:02 conta.sh
0 R   525 26484 21405  0  76   0 -   878 -      pts/0    00:00:00 ps
```

NOHUP

O nohup é um comando que faz com que os processos ignorem o sinal hangup (sinal 1 - interrupção).
Vamos executar um processo que não poderá ser finalizado, veja o exemplo:
nohup ./conta.sh
nohup: appending output to `nohup.out'

Este recurso é utilizado como recurso de iniciar programas e fazer lougout. Com isso o programa fica rodando em background mesmo que o terminal (ou janela do ambiente gráfico)

seja encerrado. O Resultado do comando é direcionado para o arquivo nohup.out

Veja este exemplo:

```
# nohup vim
# killall -1 vi
# ps -aux | grep vi
```

```
root       988  0.6 11.0 22120 13152 ?       S    08:27   0:06
kghostview /mnt/w
root      1001 14.3  3.0  8500 3572 pts/3    RN   08:38   0:50 vi
root      1026  0.0  0.5  1724  668 pts/4    S    08:44   0:00
grep vi
```

Observe que o processo não foi finalizado, então agora vamos finalizá-lo:

Monitorando processos em Tempo Real

Para monitorar processos em tempo real poderemos utilizar duas ferramentas: o top e o htop.
O comando top já vem por padrão nas distribuições linux.
Basta digitar:

```
# top
```

Este programa atualiza a tela a cada 5s. Se quiser uma monitoração segundo a segundo, digite:

```
# top -d 1
```

O Top também aceita comandos quando está em execução:
[Space] Atualizar a tela imediatamente
[h] Exibir uma tela de ajuda
[k] Matar (kill) um processo. Você deverá indicar o ID do processo e o sinal a ser enviado para ele.
[n] Alterar o número de processos exibidos. Você deverá indicar o número.
[u] Ordenar por usuário.
[M] Ordenar por uso da memória.
[P] Ordenar por uso da CPU.

Dica:
Aplicações como o Apache, Mozilla e o Nautilus são thread-aware — são criados threads múltiplos para lidar com usuários múltiplos ou pedidos múltiplos, e cada thread recebe um

ID de processo. Por default, o ps e o top exibem somente o thread principal (inicial). Para visualizar todos os threads, use o comando ps -m ou pressione [Shift]-[H] no top.
Fonte: http://web.mit.edu/rhel-doc/3/rhel-sag-pt_br-3/ch-sysinfo.html

HTOP

Este programa não vem por padrão nas distribuições, mas é o melhor de todos! Usa cores, gráficos e teclas de atalhos bem mais fáceis do que o engessado TOP.

Digite:

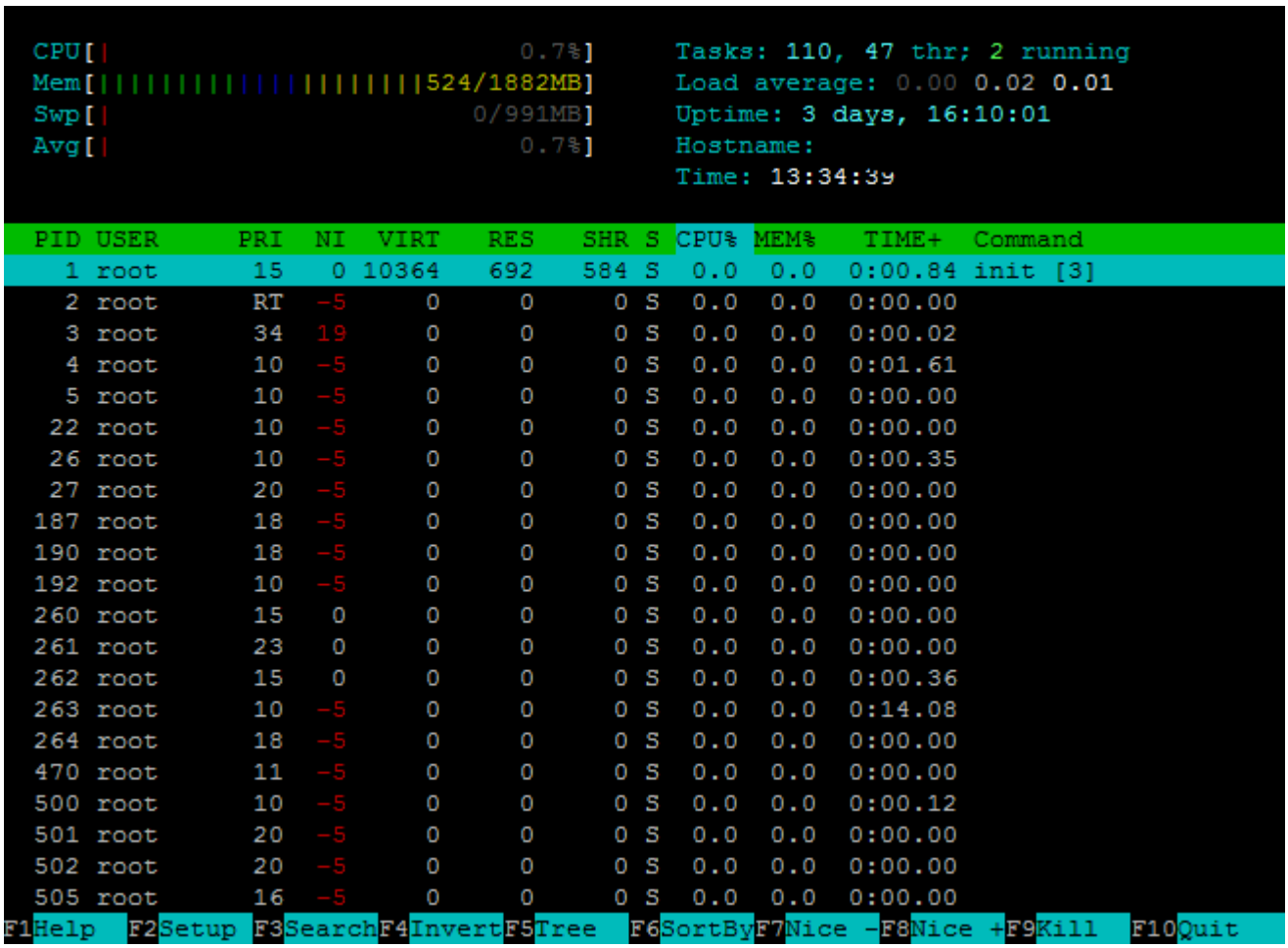
```
# htop
```

Caso você não tenha este programa, instale na sua distribuição.
Debian e derivados:

```
# apt-get install htop
```

Redhat, Fedora, CentOS...

```
# yum install htop
```



Note que as barras já dão uma idéia melhor de poder de processamento, memória RAM e Swap. Na parte inferior, tem as teclas de atalho (sempre teclas de função) que dão acesso rápido aos recursos:
F1 - Ajuda
F2 - Altera cores, graficos e personaliza o htop
F3 - Procura processos (ps aux | grep ...)
F4 - Inverte a ordem da ordenação escolhida no F6
F5 - Exibe os processos em forma de árvore (pstree)
F6 - Ordena a visualização do programa (Por consumo de cpu, memoria...)
F7 - Diminui a prioridade de um processo (renice -1)
F8 - Aumenta a prioridade de um processo (renice +1)
F9 - Envia sinais para os processos (kill e killall)
F10 - Sai do programa

Fontes:
http://www.guiafoca.org/cgs/guia/inic_interm/ch-run.html#s-run-controle
http://dainf.ct.utfpr.edu.br/~maziero/doku.php/unix:gestao_de_processos
http://web.mit.edu/rhel-doc/3/rhel-sag-pt_br-3/ch-sysinfo.html
http://www.vivaolinux.com.br/artigo/Criando-monitorando-e-terminando-processos?pagina=3