25D Linux Foundation Course

08 – Managing Linux Processes

Overview

- ❑ Understanding Linux processes
- ❑ Managing processes
- ❑ Scheduling processes

Understanding Linux Processes

- ❑ In the last section we installed applications and executables, in this section we will learn some basics to manage them

- ❑ The key to managing Linux processes is to know what a process is and how they function
  - – A process is a program that has been loaded from a long-term storage device, usually a hard disk drive, into system RAM and is currently being processed by the CPU on the motherboard

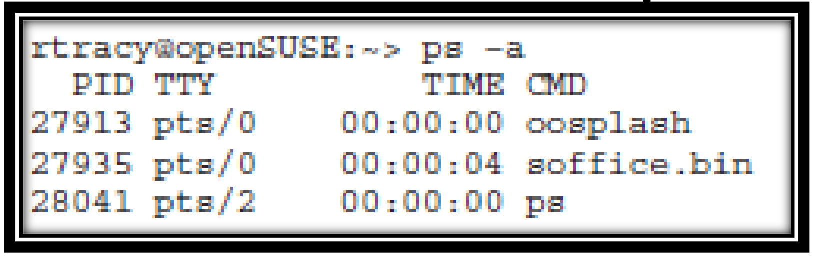
- ❑ Many different types of programs can be executed to create a process:
  - – Binary executables
  - – Internal shell commands
  - – Shell scripts

Understanding Linux Processes

Type of Program	Description
Binary executables	These are programs that were originally created as a text file using a programming language such as C or C++. The text file was then run through a compiler to create a binary file that can be processed by the CPU.
Internal shell commands	Some of the commands you enter at the shell prompt are actual binary files in the file system that are loaded and run by the CPU. For example, when you enter **top** at the shell prompt, you load the top binary file into memory. Other commands, however, are not binary executables. Instead, they are commands that are rolled into the shell program itself. For example, if you enter **exit** at a shell prompt, you are actually running an internal shell command. There is no executable file in the file system named “exit.” Instead, the computer code associated with the exit function is stored within the shell program code itself.
Shell scripts	These are text files that are executed through the shell itself. You can include commands to run binary executables within the text of any shell script. You will learn how to create shell scripts in a later chapter.

Understanding Linux Processes

- ❑ Some programs create more than one process:



```
rtracy@openSUSE:~> ps -a
  PID TTY          TIME CMD
27913 pts/0    00:00:00 oosplash
27935 pts/0    00:00:04 soffice.bin
28041 pts/2    00:00:00 ps
```

- ❑ In the above example LibreOffice suite was run and two user processes were created, oosplash and soffice.bin
- ❑ User processes called within a shell are associated with that shell session
- ❑ Not all processes running on your system are user processes
  - – most processes executing on Linux system will probably be of a different type:
    - system processes
    - daemons

User Processes vs System Processes (Daemons)

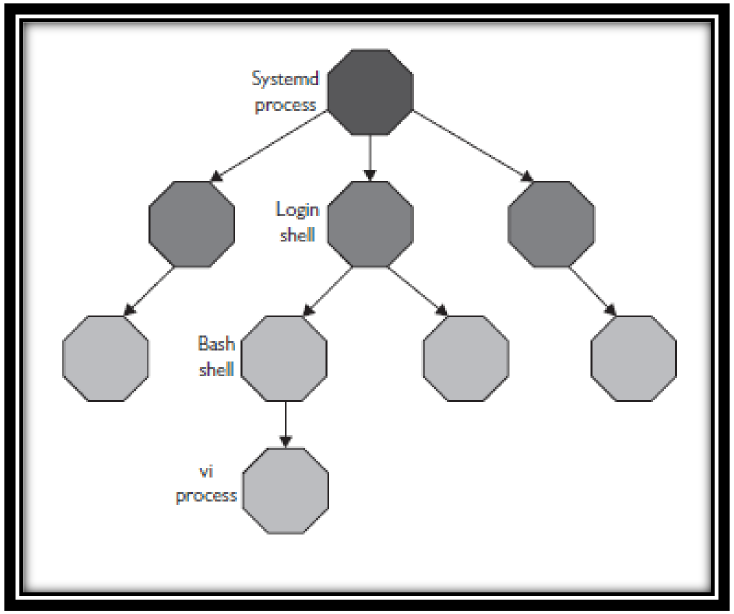
- System processes do not provide an application or interface to users
  - Usually provide a service (web server, FTP server, logging service, etc.)
  - These usually run in the background and are transparent to most users (until you stop one anyways)
  - Can be loaded from startup by the system itself (not tied to a shell like we saw with user processes)
  - Many distributions boot with many system processes and daemons running automatically on bootup
    - Some are critical, some are not
    - Running unnecessary system processes is inefficient and can be security issues

Systemd/init Process

- ❑ All Linux processes are directly or indirectly loaded by a single process, init or systemd
  - Started by the Linux kernel when it boots
  - Any process (Parent) can launch another process (Child)
- ❑ All processes on a Linux system need to be uniquely identified
  - So when a process is created it is assigned two resources:
    - ➢ Process ID (PID) number:
      - ✓ Number assigned to process that uniquely identifies it
    - ➢ Parent Process ID (PPID) number:
      - ✓ PID of the processes parent

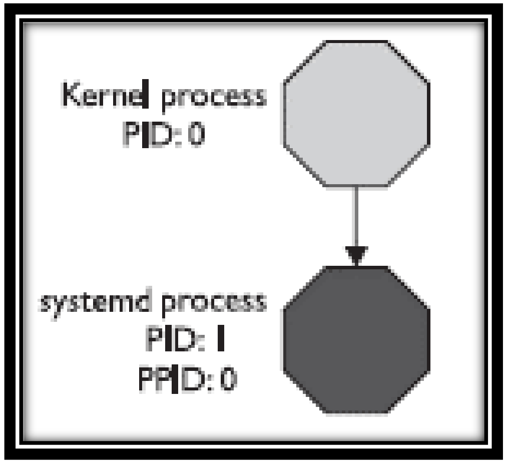
Systemd/init Process

- ❑ In this example the kernel loads the systemd or init process
- ❑ The systemd or init process then launches a login shell
- ❑ From the login shell a subshell (new PID) is launched (when running a command)
- ❑ The command is run from the subshell (vi in this example)
- ❑ When done the subshell is terminated and you are back to the original shell and PID

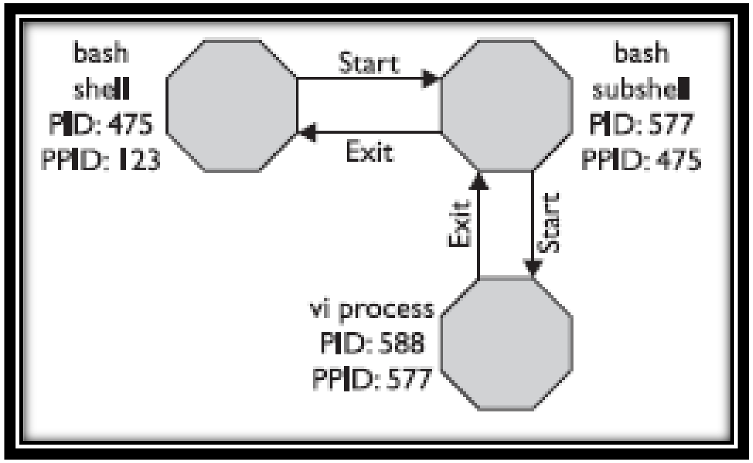


Process Forking

- ❑ Many processes are assigned a PID randomly
- ❑ Not the case with the kernel (always 0) and the systemd/init process (1)

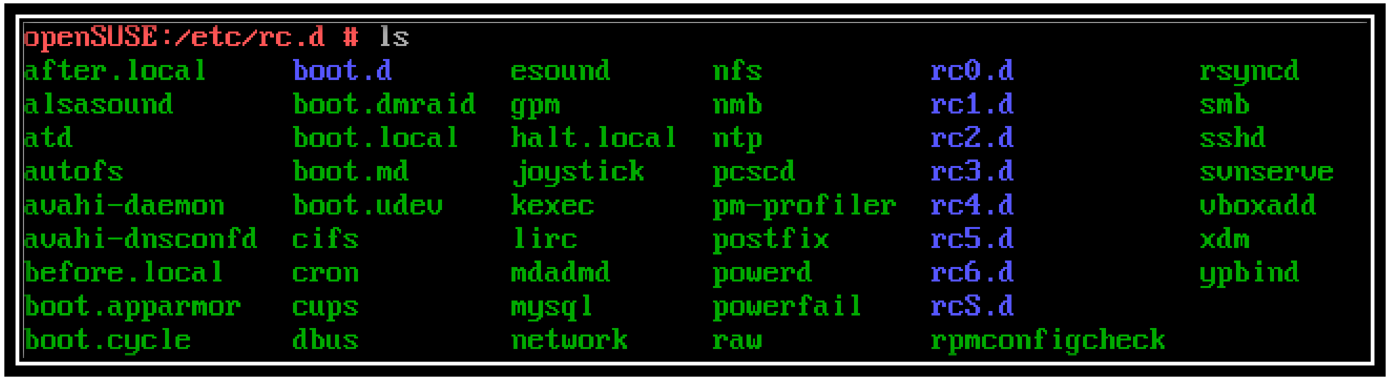


- The user creates a new vi process under the bash subshell
- Notice the PPID of the vi process
- Once the vi process is destroyed, control is sent back to the subshell



Starting System Processes

- Two types of init scripts:
  - System V and BSD
  - System V
    - stored them in the /etc/rc.d directory
    - within /etc/rc.d are a series of subdirectories named rc0.d through rc6.d
    - each of these directories is associated with a particular runlevel
    - inside these are symbolic links that point to the init scripts for your system daemons which reside in /etc/init.d.

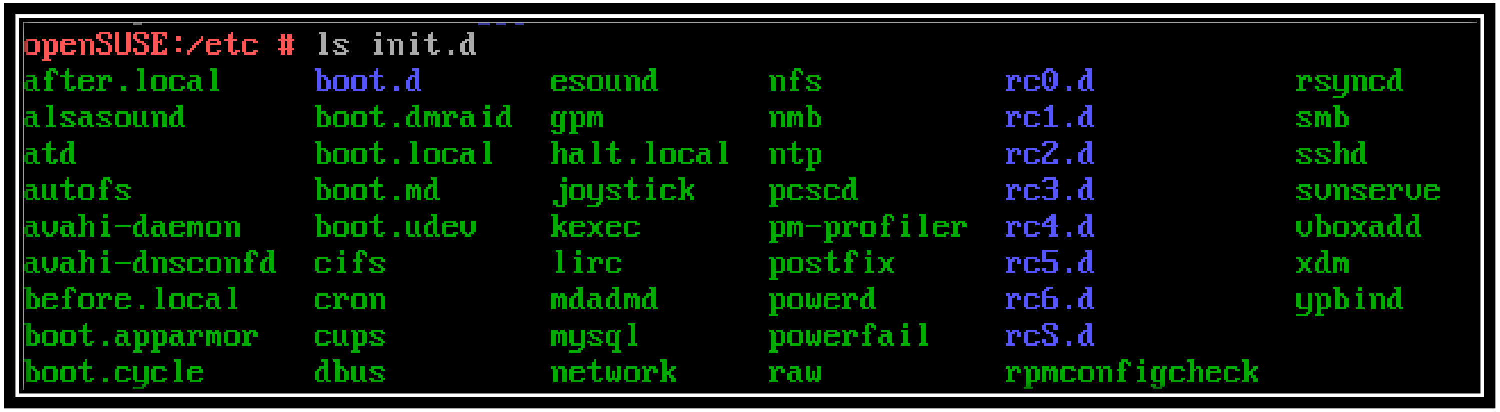


```
openSUSE:/etc/rc.d # ls
after.local      boot.d        esound       nfs          rc0.d           rsyncd
alsasound        boot.dmraid   gpm          nmb          rc1.d           smb
atd              boot.local    halt.local   ntp          rc2.d           sshd
autofs           boot.md       joystick     pcscd        rc3.d           svnserve
avahi-daemon     boot.udev     kexec        pm-profiler  rc4.d           vboxadd
avahi-dnsconfd   cifs          lirc         postfix      rc5.d           xdm
before.local     cron          mdadmd       powerd       rc6.d           ypbind
boot.apparmor    cups          mysql        powerfail    rcS.d
boot.cycle       dbus          network      raw          rpmconfigcheck
```

Starting System Processes

❑ BSD

- other Linux distributions use BSD-style init scripts
- these scripts reside in the /etc/init.d directory
- within /etc/init.d are a series of directories named rc0.d through rc6.d.
- as with System V init scripts, these directories are associated with specific runlevels
- these directories contain links that point to the init scripts in /etc/init.d



```
openSUSE:/etc # ls init.d
after.local     boot.d       esound      nfs          rc0.d           rsyncd
alsasound       boot.dmraid  gpm         nmb          rc1.d           smb
atd             boot.local   halt.local  ntp          rc2.d           sshd
autofs          boot.md      joystick    pcscd        rc3.d           svnserve
avahi-daemon    boot.udev    kexec       pm-profiler  rc4.d           vboxadd
avahi-dnsconfd  cifs         lirc        postfix      rc5.d           xdm
before.local    cron         mdadmd      powerd       rc6.d           ypbind
boot.apparmor   cups         mysql       powerfail    rcS.d
boot.cycle      dbus         network     raw          rpmconfigcheck
```

Starting System Processes

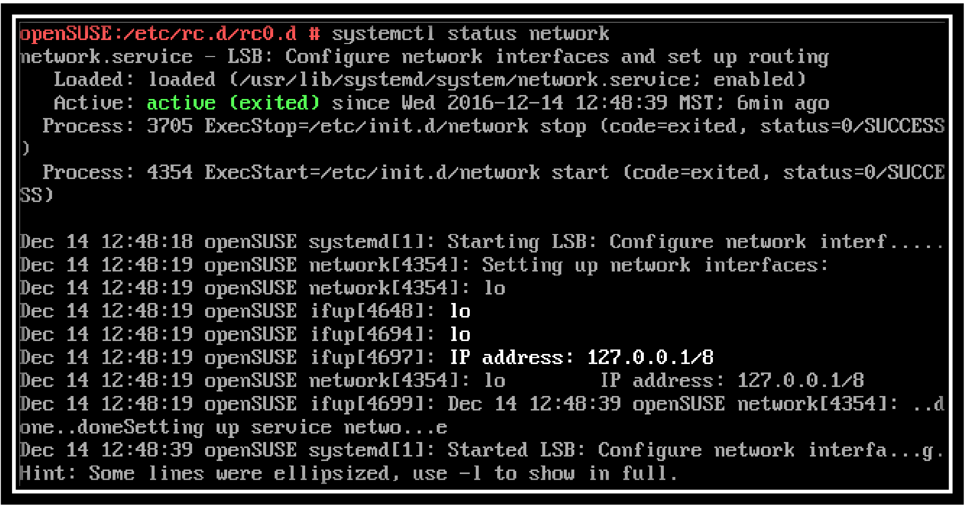
❑ Executing scripts from command prompt

- BSD-style
  - /etc/init.d/script_name
    - /etc/init.d/smb start
- System V-style
  - /etc/rc.d/init.d/script_name
    - /etc/rc.d/init.d/gpm
- Some distributions rcscript can be used
  - rcscript_name start | stop | restart
    - rcsmb start (starts smb service)

Starting System Processes

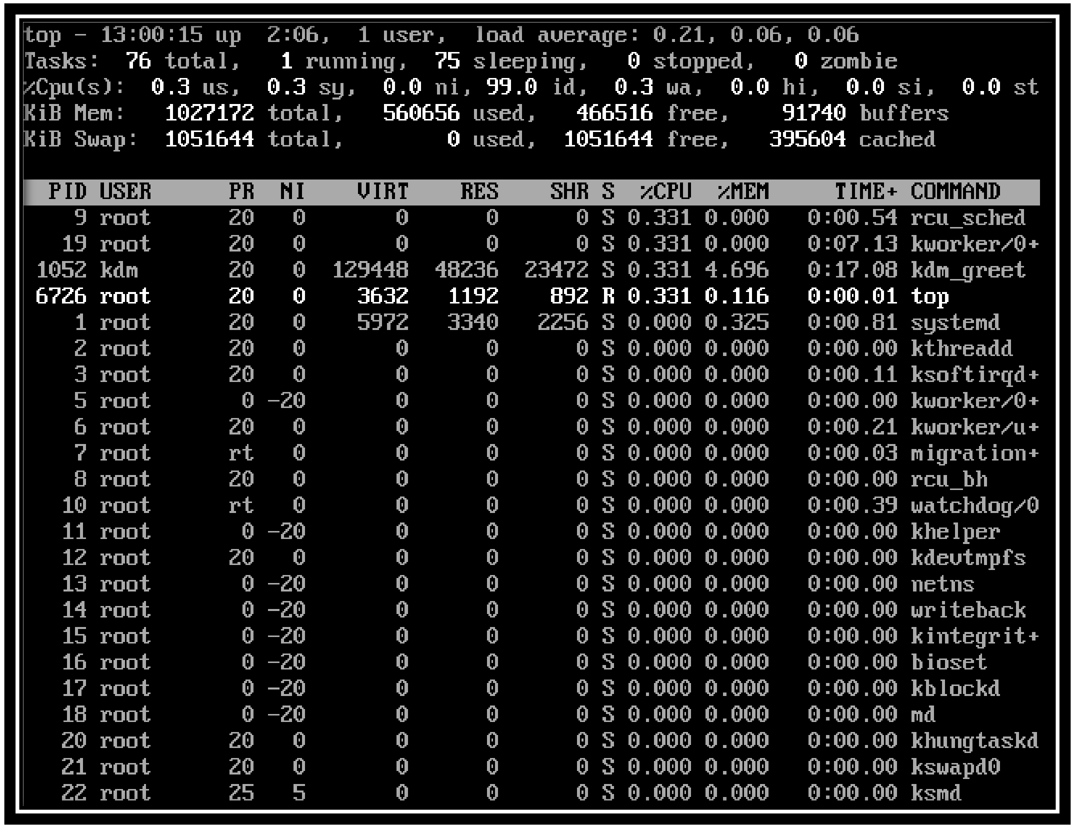
❑ **systemd system services are managed using service files**

- **systemctl start service_name**
  - **systemctl start network**
- **systemctl stop service_name**
  - **systemctl stop network**
- **systemctl restart service_name**
  - **systemctl restart network**
- **systemctl status service_name**



```
openSUSE:/etc/rc.d/rc0.d # systemctl status network
network.service - LSB: Configure network interfaces and set up routing
   Loaded: loaded (/usr/lib/systemd/system/network.service; enabled)
   Active: active (exited) since Wed 2016-12-14 12:48:39 MST; 6min ago
  Process: 3705 ExecStop=/etc/init.d/network stop (code=exited, status=0/SUCCESS
)
  Process: 4354 ExecStart=/etc/init.d/network start (code=exited, status=0/SUCCE
SS)

Dec 14 12:48:18 openSUSE systemd[1]: Starting LSB: Configure network interf.....
Dec 14 12:48:19 openSUSE network[4354]: Setting up network interfaces:
Dec 14 12:48:19 openSUSE network[4354]: lo
Dec 14 12:48:19 openSUSE ifup[4648]: lo
Dec 14 12:48:19 openSUSE ifup[4694]: lo
Dec 14 12:48:19 openSUSE ifup[4697]: IP address: 127.0.0.1/8
Dec 14 12:48:19 openSUSE network[4354]: lo        IP address: 127.0.0.1/8
Dec 14 12:48:19 openSUSE ifup[4699]: Dec 14 12:48:39 openSUSE network[4354]: ..d
one..doneSetting up service netwo...e
Dec 14 12:48:39 openSUSE systemd[1]: Started LSB: Configure network interfa...g.
Hint: Some lines were ellipsized, use -l to show in full.
```

Using top

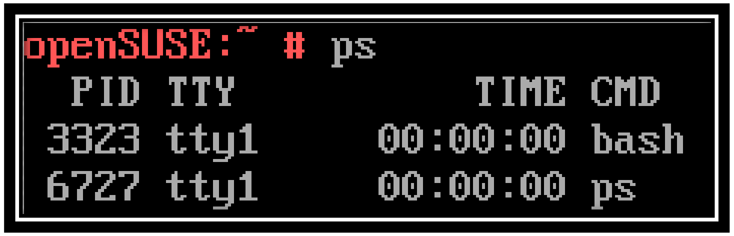


```
top - 13:00:15 up  2:06,  1 user,  load average: 0.21, 0.06, 0.06
Tasks:  76 total,   1 running,  75 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.3 us,  0.3 sy,  0.0 ni, 99.0 id,  0.3 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:   1027172 total,   560656 used,   466516 free,    91740 buffers
KiB Swap:  1051644 total,        0 used,  1051644 free,   395604 cached

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
    9 root      20   0       0      0      0 S 0.331 0.000   0:00.54 rcu_sched
   19 root      20   0       0      0      0 S 0.331 0.000   0:07.13 kworker/0+
 1052 kdm       20   0  129448  48236  23472 S 0.331 4.696   0:17.08 kdm_greet
 6726 root      20   0    3632   1192    892 R 0.331 0.116   0:00.01 top
    1 root      20   0    5972   3340   2256 S 0.000 0.325   0:00.81 systemd
    2 root      20   0       0      0      0 S 0.000 0.000   0:00.00 kthreadd
    3 root      20   0       0      0      0 S 0.000 0.000   0:00.11 ksoftirqd+
    5 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 kworker/0+
    6 root      20   0       0      0      0 S 0.000 0.000   0:00.21 kworker/u+
    7 root      rt   0       0      0      0 S 0.000 0.000   0:00.03 migration+
    8 root      20   0       0      0      0 S 0.000 0.000   0:00.00 rcu_bh
   10 root      rt   0       0      0      0 S 0.000 0.000   0:00.39 watchdog/0
   11 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 khelper
   12 root      20   0       0      0      0 S 0.000 0.000   0:00.00 kdevtmpfs
   13 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 netns
   14 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 writeback
   15 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 kintegrit+
   16 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 bioset
   17 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 kblockd
   18 root       0 -20       0      0      0 S 0.000 0.000   0:00.00 md
   20 root      20   0       0      0      0 S 0.000 0.000   0:00.00 khungtaskd
   21 root      20   0       0      0      0 S 0.000 0.000   0:00.00 kswapd0
   22 root      25   5       0      0      0 S 0.000 0.000   0:00.00 ksmd
```

top utility columns

- PID (process ID)
- USER (process owner)
- PR (priority assigned)
- NI (nice value)
- VIRT (virtual memory)
- RES (physical RAM/resident size)
- SHR (shared memory)
- S (status)
  - D (uninterruptibly sleeping)
  - R (running)
  - S (sleeping)
  - T (traced or stopped)
  - Z (zombied)
- %CPU (% of cpu time)
- %MEM (% of available RAM)
- TIME+ (cpu time consumed)
- COMMAND (command that started process)

Using ps



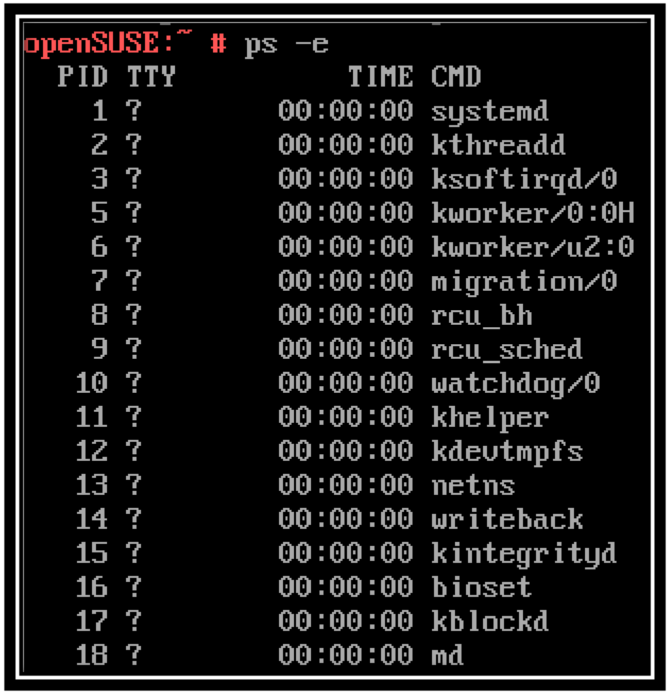
```
openSUSE:~ # ps
  PID TTY          TIME CMD
 3323 tty1     00:00:00 bash
 6727 tty1     00:00:00 ps
```

- ❑ In this example, the following processes are displayed by ps:
  - bash: the current bash shell session
  - ps: because ps is in use to list current processes, its process is also displayed
- ❑ Notice that the following information is displayed by default:
  - PID: the process ID of the process
  - TTY: the name of the terminal session (shell) that the process is running within
  - TIME: the amount of CPU time used by the process
  - CMD: the name of the command that was entered to create the process

Using ps -e

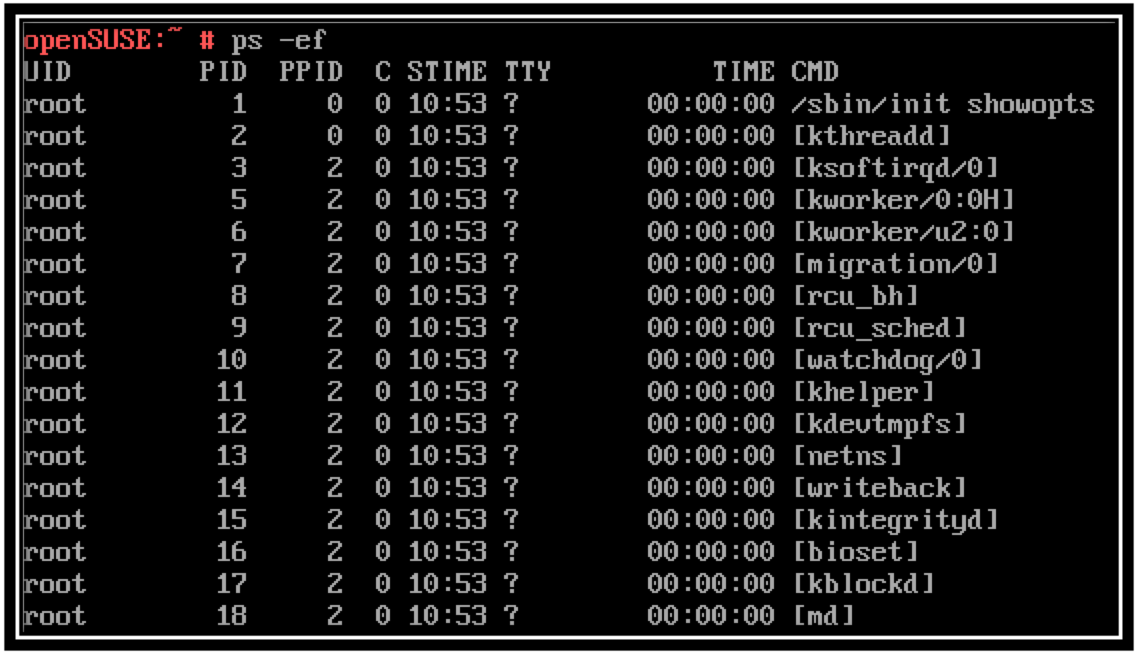
- ❑ The -e (select all processes, could use a –A as well) option results in many more processes

- ❑ The ? indicates the process is a system process



```
openSUSE:~ # ps -e
  PID TTY          TIME CMD
    1 ?        00:00:00 systemd
    2 ?        00:00:00 kthreadd
    3 ?        00:00:00 ksoftirqd/0
    5 ?        00:00:00 kworker/0:0H
    6 ?        00:00:00 kworker/u2:0
    7 ?        00:00:00 migration/0
    8 ?        00:00:00 rcu_bh
    9 ?        00:00:00 rcu_sched
   10 ?        00:00:00 watchdog/0
   11 ?        00:00:00 khelper
   12 ?        00:00:00 kdevtmpfs
   13 ?        00:00:00 netns
   14 ?        00:00:00 writeback
   15 ?        00:00:00 kintegrityd
   16 ?        00:00:00 bioset
   17 ?        00:00:00 kblockd
   18 ?        00:00:00 md
```

Using ps -ef

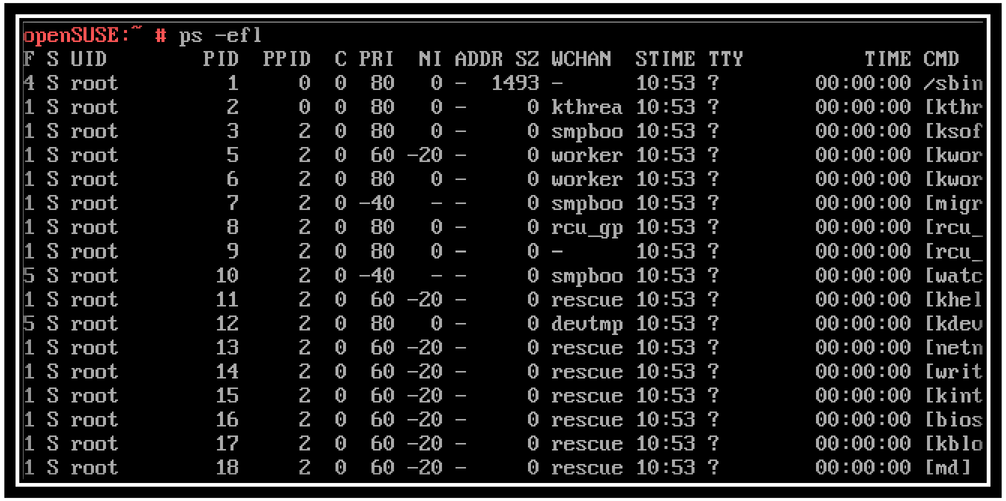


```
openSUSE:~ # ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 10:53 ?        00:00:00 /sbin/init showopts
root         2     0  0 10:53 ?        00:00:00 [kthreadd]
root         3     2  0 10:53 ?        00:00:00 [ksoftirqd/0]
root         5     2  0 10:53 ?        00:00:00 [kworker/0:0H]
root         6     2  0 10:53 ?        00:00:00 [kworker/u2:0]
root         7     2  0 10:53 ?        00:00:00 [migration/0]
root         8     2  0 10:53 ?        00:00:00 [rcu_bh]
root         9     2  0 10:53 ?        00:00:00 [rcu_sched]
root        10     2  0 10:53 ?        00:00:00 [watchdog/0]
root        11     2  0 10:53 ?        00:00:00 [khelper]
root        12     2  0 10:53 ?        00:00:00 [kdevtmpfs]
root        13     2  0 10:53 ?        00:00:00 [netns]
root        14     2  0 10:53 ?        00:00:00 [writeback]
root        15     2  0 10:53 ?        00:00:00 [kintegrityd]
root        16     2  0 10:53 ?        00:00:00 [bioset]
root        17     2  0 10:53 ?        00:00:00 [kblockd]
root        18     2  0 10:53 ?        00:00:00 [md]
```

❑ With the –f option (full format listing), you can now view additional information, including the following:

- UID: the user ID of the process’s owner
- PPID: the PID of the process’s parent process
- C: the amount of processor time utilized by the process
- STIME: the time that the process started

Using ps -efl



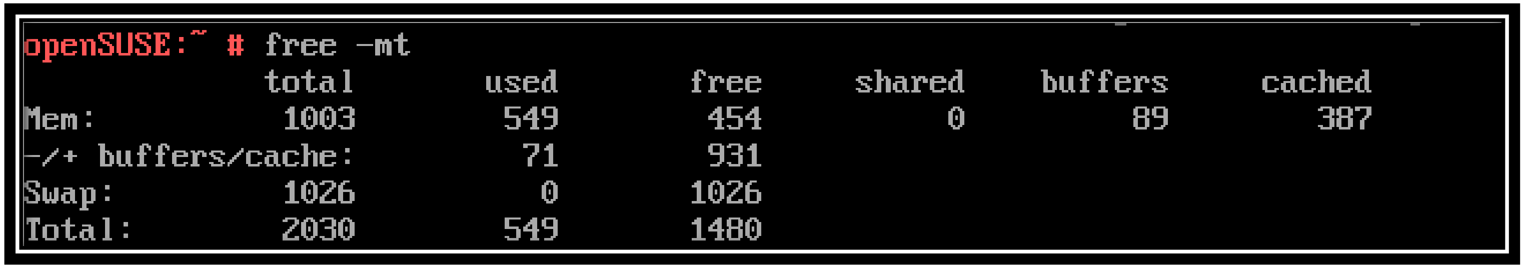
```
openSUSE:~ # ps -efl
F S UID        PID  PPID  C PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
4 S root         1     0  0  80   0 -  1493 -      10:53 ?        00:00:00 /sbin
1 S root         2     0  0  80   0 -     0 kthrea 10:53 ?        00:00:00 [kthr
1 S root         3     2  0  80   0 -     0 smpboo 10:53 ?        00:00:00 [ksof
1 S root         5     2  0  60 -20 -     0 worker 10:53 ?        00:00:00 [kwor
1 S root         6     2  0  80   0 -     0 worker 10:53 ?        00:00:00 [kwor
1 S root         7     2  0 -40   - -     0 smpboo 10:53 ?        00:00:00 [migr
1 S root         8     2  0  80   0 -     0 rcu_gp 10:53 ?        00:00:00 [rcu_
1 S root         9     2  0  80   0 -     0 -      10:53 ?        00:00:00 [rcu_
5 S root        10     2  0 -40   - -     0 smpboo 10:53 ?        00:00:00 [watc
1 S root        11     2  0  60 -20 -     0 rescue 10:53 ?        00:00:00 [khel
5 S root        12     2  0  80   0 -     0 devtmp 10:53 ?        00:00:00 [kdev
1 S root        13     2  0  60 -20 -     0 rescue 10:53 ?        00:00:00 [netn
1 S root        14     2  0  60 -20 -     0 rescue 10:53 ?        00:00:00 [writ
1 S root        15     2  0  60 -20 -     0 rescue 10:53 ?        00:00:00 [kint
1 S root        16     2  0  60 -20 -     0 rescue 10:53 ?        00:00:00 [bios
1 S root        17     2  0  60 -20 -     0 rescue 10:53 ?        00:00:00 [kblo
1 S root        18     2  0  60 -20 -     0 rescue 10:53 ?        00:00:00 [md]
```

- **With the –l option, you can view the following information about processes running on your system:**

- **F**: The flags associated with the process. This column uses the following codes:
  - 1: forked, but didn’t execute
  - 4: used root privileges
  - 5: both flags applied
- **S**: The state of the process. This column uses the following codes:
  - D: uninterruptible sleep
  - R: running
  - S: interruptible sleep
  - T: stopped or traced
  - Z: zombied
- **PRI**: priority
- **NI**: nice value
- **ADDR**: memory address
- **SZ**: size of the process
- **WCHAN**: name of the kernel function in which the process is sleeping

Using free

- □ The free command displays the amount of free and allocated RAM and swap memory on the Linux system
- □ The –m option is used to display memory statistics in megabytes
- □ The –t option is used to display totals for each category of information



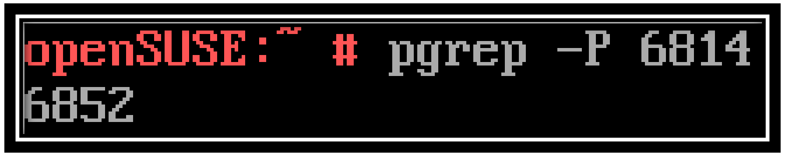
```
openSUSE:~ # free -mt
             total       used       free     shared    buffers     cached
Mem:          1003        549        454          0         89        387
-/+ buffers/cache:         71        931
Swap:         1026          0       1026
Total:        2030        549       1480
```

Using pgrep

❑ pgrep command

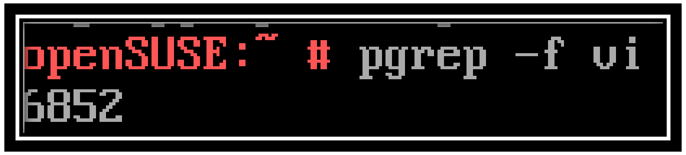
– Combines the functionality of the ps and grep commands

- -P: ppid matches on the specified parent process ID (in this example bash (6814) is the parent of vi (6852))



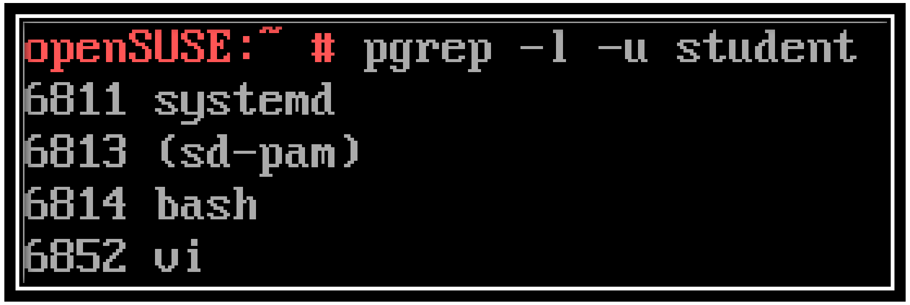
```
openSUSE:~ # pgrep -P 6814
6852
```

- -f: name matches on the specified process name



```
openSUSE:~ # pgrep -f vi
6852
```

- -u: user_name matches on the specified process owner



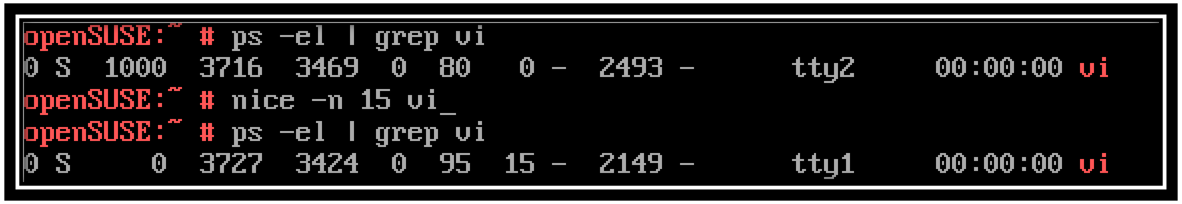
```
openSUSE:~ # pgrep -l -u student
6811 systemd
6813 (sd-pam)
6814 bash
6852 vi
```

Prioritizing Processes

- ❑ Remember that Linux is a multitasking multiuser operating system
- ❑ Because Linux is multitasking priority levels can be specified for each process
- ❑ Linux will attempt to “balance” the amount of CPU time given to processes running
- ❑ You may want to have a process to have a higher priority
- ❑ Several Linux utilities can assist you with prioritizing processes, we will look at the nice and renice utilities

Setting Priorities with nice

- PR value is process's kernel priority
- The higher the number, the lower the priority
- Nice value is factored into the kernel calculations that determine the priority of the process
- Nice value ranges between -20 and +19
- Nice value is typically set when a command is initially launched

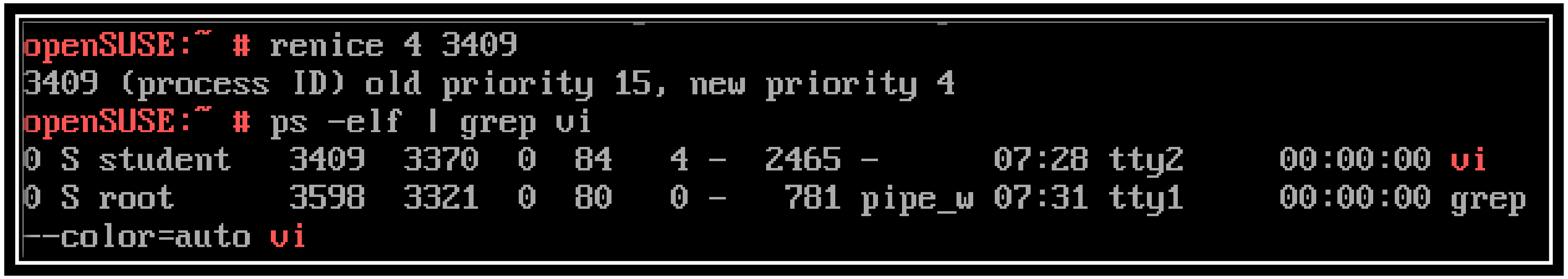


```
openSUSE:~ # ps -el | grep vi
0 S  1000  3716  3469  0  80   0 -  2493 -      tty2     00:00:00 vi
openSUSE:~ # nice -n 15 vi_
openSUSE:~ # ps -el | grep vi
0 S     0  3727  3424  0  95  15 -  2149 -      tty1     00:00:00 vi
```

- In the above example vi is launched and the nice utility is used to set the nice value +15
- This changes the overall kernel calculation from 80 (default) to 95, lowering the priority

Setting Priorities of Running Processes with renice

- Rather than killing and restarting a process, the renice command can be used to adjust the priority of a process that is currently running



```
openSUSE:~ # renice 4 3409
3409 (process ID) old priority 15, new priority 4
openSUSE:~ # ps -elf | grep vi
0 S student   3409  3370  0  84   4 -  2465 -      07:28 tty2     00:00:00 vi
0 S root      3598  3321  0  80   0 -   781 pipe_w 07:31 tty1     00:00:00 grep
--color=auto vi
```

- In the above example renice is used to set the nice value from 15 to 4 thus adjusting the kernel calculation from 95 to 84

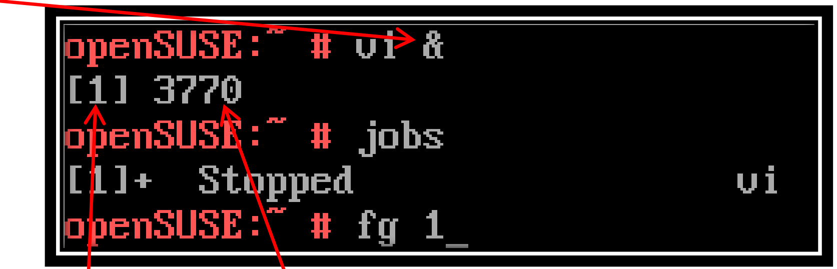
Running Processes in the Background

- ❑ **Example of a process running in the foreground**
  - – **When you enter any command at the shell prompt, a subshell is created and the process is run within it. As soon as the process exits, the subshell is destroyed.**
  - – **During the time that the process is running, the shell prompt of the parent shell disappears. You can't do anything at the shell prompt unless you open a new terminal session.**
  - – **Running the vi text editor is another good example. Entering vi at the prompt will cause vi to run in the foreground and the bash shell in the background.**

Running Processes in the Background

❑ Example of a process running in the background

– This tells the shell to run the program in the background.



```
openSUSE:~ # vi &
[1] 3770
openSUSE:~ # jobs
[1]+  Stopped                 vi
openSUSE:~ # fg 1
```

– Notice the two values displayed on the screen after the process was run in the background

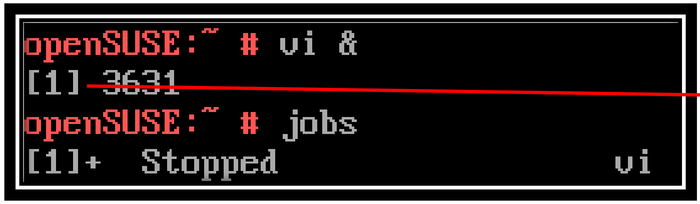
- The first value [1] is the background job ID that was assigned to the background job.
- The second value is the PID of the process.

– The jobs command will list the background jobs and status

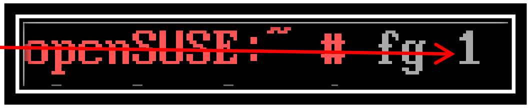
– The last command, fg 1, would move the job with the background job id of 1 to the foreground (vi in this case)

Switching Processes Between the Background and the Foreground

- **To switch a process between the background and the foreground while it is still running**
  - **fg**
    - **This command will move a background process to the foreground. The syntax is fg job_ID.**

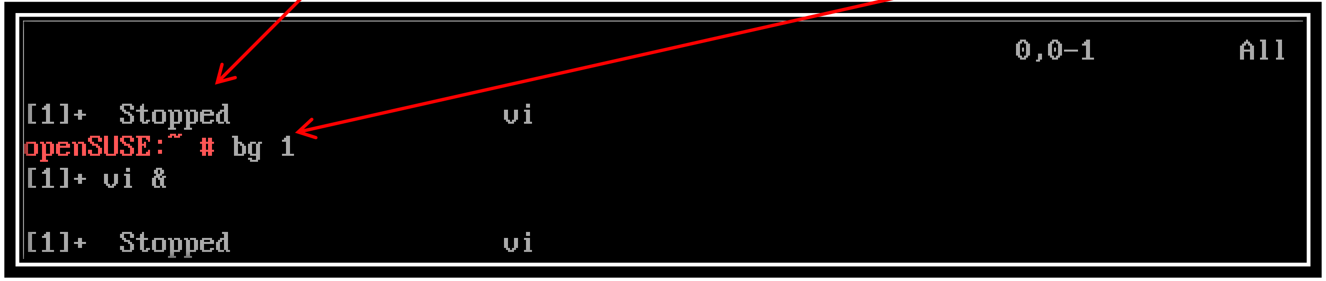


```
openSUSE:~ # vi &
[1] 3631
openSUSE:~ # jobs
[1]+  Stopped                    vi
```



```
openSUSE:~ # fg 1
```

  - **bg**
    - **This command will move a foreground process to the background. To use this utility, you must first assign the foreground job a background job ID. This is done by pressing ctrl-z. When you do, you'll see the process stop and a background job ID assigned to the process. You can then enter bg job_ID to move the process to the background.**



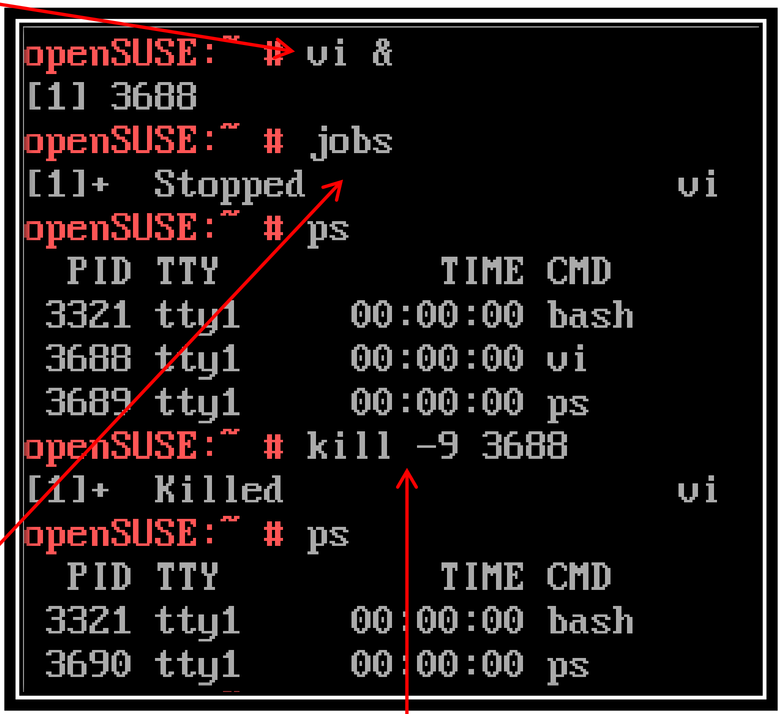
```
                                                          0,0-1          All

[1]+  Stopped                    vi
openSUSE:~ # bg 1
[1]+ vi &

[1]+  Stopped                    vi
```

Using kill, killall, and pkill

- ❑ The kill command is used to terminate a process
  - – kill –signal PID (kill 3631 (vi PID in last slide))
- ❑ Kill signals
  - – SIGHUP – kill signal 1 (restarts the process with the same PID)
  - – SIGINT – kill signal 2 (sends a CTRL-C)
  - – SIGKILL – kill signal 9 (brute-force signal)
  - – SIGTERM – kill signal 15 (default kill signal)
- ❑ The killall command uses the command name rather than the PID
  - – killall –signal command
- ❑ The pkill command is used just like pgrep
  - – pkill –signal –f pattern

Using kill, killall, and pkill

- ❑ In the below example we started vi in the background



```
openSUSE:~ # vi &
[1] 3688
openSUSE:~ # jobs
[1]+  Stopped                 vi
openSUSE:~ # ps
  PID TTY          TIME CMD
 3321 tty1     00:00:00 bash
 3688 tty1     00:00:00 vi
 3689 tty1     00:00:00 ps
openSUSE:~ # kill -9 3688
[1]+  Killed                  vi
openSUSE:~ # ps
  PID TTY          TIME CMD
 3321 tty1     00:00:00 bash
 3690 tty1     00:00:00 ps
```

- ❑ Verified it was a job and given a job number (1)
- ❑ Then used the brute force (9) kill command to stop the vi process (PID 3688)

Keeping a Process Running After Logout

- ❑ To keep a process running after logout, use the nohup command
  - – nohup program &
    - If the command generates output that is usually sent to the stdout, nohup will redirect the output to the ~/nohup.out file
- ❑ Another way to keep a process running after logout is by using the screen command
  - – Must be installed in order to use
  - – Allows for multiple shell windows to be used from a single SSH session (really helpful for remote connections)
  - – To use screen, simply enter the screen command at the prompt

screen Examples

❑ Examples of using screen

- ctrl-a and then ? causes the screen help to be displayed
- ctrl-a and then c causes a new screen window to be created. The old window you were working in remains active along with any processes that were running within it
  - top for example
- ctrl-a and then n toggles between open windows in screen
- ctrl-a and then d detaches your screen window and drops you back at your original shell prompt. However, whatever you had running in the window remains running. In fact, you can log completely out of the server and everything will keep working within the detached window.
- screen –r reattaches you to a detached screen window. If you have multiple detached screen windows, you'll be prompted to specify which one you want to reattach to.

Exercise 8-1: Working with Linux Processes

Please open your Practical Exercise book to Exercise 8-1.

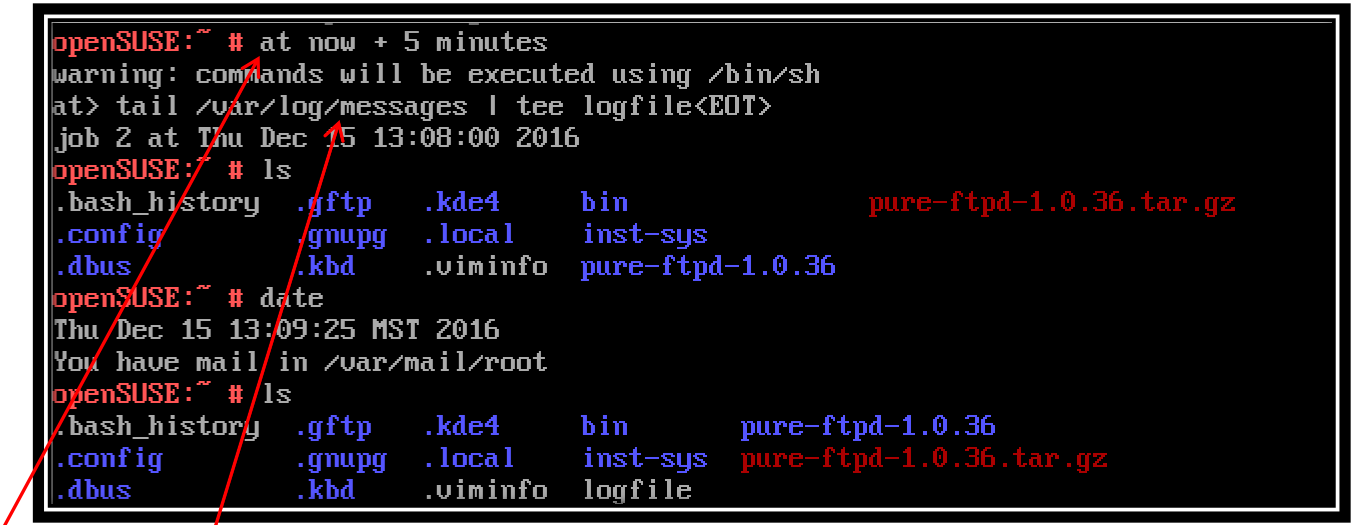
Time to Complete: 5 Minutes

Using the at daemon

- ❑ The at daemon can be used to schedule a process to run once sometime in the future
- ❑ Runs in the background
- ❑ Typically is installed by default; might need to manually download and install
- ❑ Ensure the atd daemon is running by entering rcatd start
- ❑ Can also use the chkconfig command to see if atd starts automatically upon system booting
- ❑ Systems that use systemd can start the atd daemon by entering systemctl start atd
- ❑ To specify which users can or cannot create at jobs edit the following files:
  - /etc/at.allow
  - /etc/at.deny

Using the at daemon

Here is an example of using the at utility to schedule a job/task:



```
openSUSE:~ # at now + 5 minutes
warning: commands will be executed using /bin/sh
at> tail /var/log/messages | tee logfile<EOT>
job 2 at Thu Dec 15 13:08:00 2016
openSUSE:~ # ls
.bash_history  .gftp   .kde4     bin                pure-ftpd-1.0.36.tar.gz
.config        .gnupg  .local    inst-sys
.dbus          .kbd    .viminfo  pure-ftpd-1.0.36
openSUSE:~ # date
Thu Dec 15 13:09:25 MST 2016
You have mail in /var/mail/root
openSUSE:~ # ls
.bash_history  .gftp   .kde4     bin        pure-ftpd-1.0.36
.config        .gnupg  .local    inst-sys   pure-ftpd-1.0.36.tar.gz
.dbus          .kbd    .viminfo  logfile
```

1. Enter at with a time (in this case we used now + 5 minutes)
2. Enter the command(s) you want at to run and press CTRL-D
   a) In this example tail /var/log/messages | tee logfile was entered. So in 5 minutes from now at will run tail and collect the last few lines from the messages file in /var/log and save that information to a file named logfile in the current directory.
3. Sure enough 5 minutes later there is the logfile file, entering atq will list jobs

Using the at daemon

Type of Reference	Syntax	Description
Fixed	HH:MM	Specifies the exact hour and minute when the commands should be run. The at daemon assumes that the hour and minute specified is today unless that time has already passed; then it assumes it is tomorrow. You can also add am or pm to the value to specify morning or afternoon.
	Noon	Specifies that a command be run at 12:00 p.m.
	Midnight	Specifies that a command be run at 12:00 a.m.
	Teatime	Specifies that a command be run at 4:00 p.m.
	MMDDYY or MM/DD/YY or MM.DD.YY	Specifies the exact month, date, and year when a command is to be run.

Using the at daemon

Type of Reference	Syntax	Description
	HH:MM MMDDYY	Specifies the exact month, date, year, and time when a command is to be run.
Relative	now	Specifies that the command be run immediately.
	now + value	Specifies that the command be run at a certain time in the future. For example, you could enter any of the following: now + 5 minutes now + 2 hours now + 3 days
	today	Specifies that the command be run today. You can mix this value with a fixed value from one of the preceding types, such as 2 pm today.
	tomorrow	Specifies that the command be run tomorrow. You can also mix this value with a fixed value, such as 2 pm tomorrow.

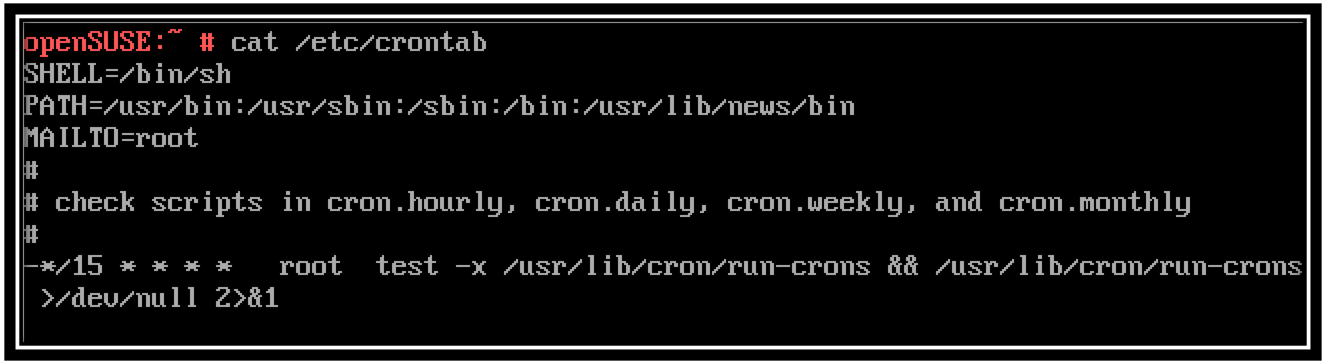
Using the cron Daemon

- ❑ The at daemon can only schedule a job to run once in the future
- ❑ To run jobs in the future on a regular schedule, use cron
- ❑ The cron daemon is a service that runs continuously in the background
- ❑ The crontab file is checked once every minute for any scheduled jobs
- ❑ The cron daemon is configured to run automatically every time the system boots
- ❑ Cron can run system jobs or user-specific jobs

Using cron to Manage Scheduled System Jobs

- ❑ To run system jobs, the cron service uses the /etc/crontab file

- ❑ Different cron scenarios
  - /etc/cron.hourly Contains cron scripts that are run every hour
  - /etc/cron.daily Contains cron scripts that are run every day
  - /etc/cron.weekly Contains cron scripts that are run once a week
  - /etc/cron.monthly Contains cron scripts that are run once a month



```
openSUSE:~ # cat /etc/crontab
SHELL=/bin/sh
PATH=/usr/bin:/usr/sbin:/sbin:/bin:/usr/lib/news/bin
MAILTO=root
#
# check scripts in cron.hourly, cron.daily, cron.weekly, and cron.monthly
#
-*/15 * * * *   root  test -x /usr/lib/cron/run-crons && /usr/lib/cron/run-crons
 >/dev/null 2>&1
```

crontab File

- **Suppose you wanted to run the tar command to back up the /home directory using the tar –cvf /media/usb/backup.tar /home command every day of every month, except Sundays, the following line can be added a crontab file:**

```
5 23 * * 1-6 /bin/tar -cvf /media/usb/backup.tar /home
```

The above example specifies that the command be run at 5 minutes after 11:00 p.m. (23) every day (*) of every month (*) on Monday (1) through Saturday (6).

Field	Description
1	Minutes. This field specifies the minutes past the hour that the command should be run.
2	Hour. This field specifies the hour of the day when the command should be run. The cron daemon prefers military time, so you should use a value of 0 to 23 in this field.
3	Day. This field specifies the day of the month that the command should be run.
4	Month. This field specifies the month of the year when the command should be run.
5	Day of the week. Sunday is 0 and Saturday is 6.
6	The name of the command, including the full path, to be run.

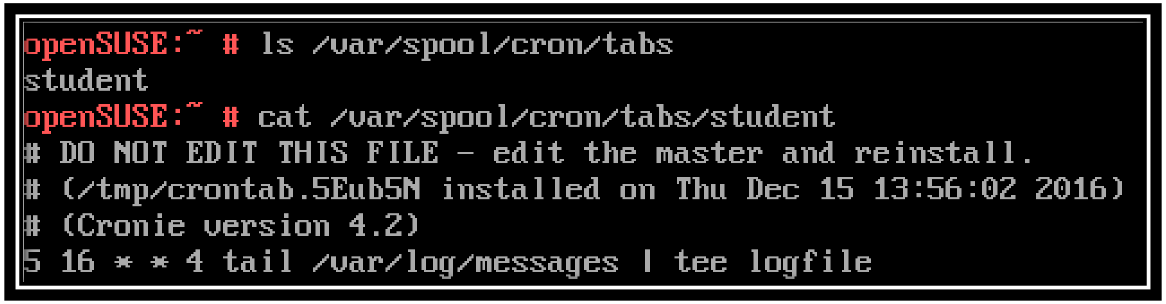
```
* Is a wildcard that means match everything
```

Using cron to Manage Scheduled User Jobs

- ❑ Users can create their own schedules using a crontab file associated with their user account
- ❑ User crontab files are stored in /var/spool/cron/tabs in a file under the specified username
- ❑ Users can be restricted from creating their own cron jobs:
  - – /etc/cron.allow
  - – /etc/cron.deny
- ❑ Use the crontab –e command to create a user crontab file
- ❑ Use the same syntax from the previous example
- ❑ After the file is saved, it is automatically stored in the /var/spool/cron/tabs folder in a file for the specified username

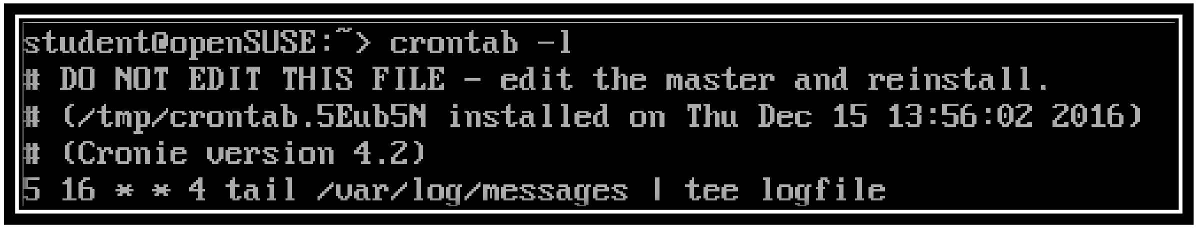
Using cron to Manage Scheduled User Jobs

- ❑ **Example:**



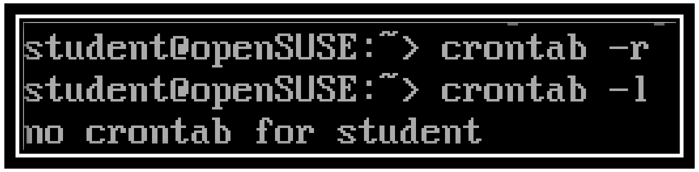
```
openSUSE:~ # ls /var/spool/cron/tabs
student
openSUSE:~ # cat /var/spool/cron/tabs/student
# DO NOT EDIT THIS FILE - edit the master and reinstall.
# (/tmp/crontab.5Eub5N installed on Thu Dec 15 13:56:02 2016)
# (Cronie version 4.2)
5 16 * * 4 tail /var/log/messages | tee logfile
```

- ❑ **To view a user's crontab, enter crontab –l**



```
student@openSUSE:~> crontab -l
# DO NOT EDIT THIS FILE - edit the master and reinstall.
# (/tmp/crontab.5Eub5N installed on Thu Dec 15 13:56:02 2016)
# (Cronie version 4.2)
5 16 * * 4 tail /var/log/messages | tee logfile
```

- ❑ **To remove a user's crontab, enter crontab -r**



```
student@openSUSE:~> crontab -r
student@openSUSE:~> crontab -l
no crontab for student
```

Exercise 8-2: Scheduling Linux Processes

Please open your Practical Exercise book to Exercise 8-2.

Time to Complete: 5 Minutes

Using anacron

- ❑ anacron and cron work in similar fashions
- ❑ Key difference is that cron assumes that the system will remain up and running 24/7
- ❑ anacron doesn’t make such assumptions:
  - – if the system is off when anacron was scheduled to run, the missed job will automatically run upon the system coming back up
- ❑ anacron use the /etc/anacrontab file
- ❑ anacron is not installed in openSUSE 13.1 but you may see it in other distributions

Using anacron

Field	Description	Value
period	The first field specifies the recurrence period (in days)	1 The task recurs daily 7 The task recurs weekly 30 The task recurs monthly
delay	The second field specifies the delay (in minutes) anacron should wait before executing a skipped job after the system starts up.	minutes
job-identifier	The third field contains the job identifier. This is the name that will be used for the job's timestamp file and must be unique for each anacron job. This file is created in the /var/spool/anacron directory and contains a single line with a timestamp that indicates the last time the particular job was run.	
command	The fourth field specifies the command or script that should be run.	command

Summary

❑ Understanding Linux processes

❑ Managing processes

❑ Scheduling processes

Questions

Questions?

Check on Learning

Question 1

You just entered vi at the shell prompt. What type of process was created on your Linux system?

A. User

B. System

C. Daemon

D. System V

Check on Learning

Question 2

Your current shell session has a PID of 3456. You run the su command to change to the root user account. The su process has a PID of 3457. You then run vi from the shell prompt as root. The vi process has a PID of 3458. What is the PPID of the vi process?

A. 3456

B. 3457

C. 3458

D. 3459

Check on Learning

Question 3

You want to use ps to display extended information about only the processes associated with your current terminal session. Which command will do this?

A. ps

B. ps –e

C. ps –f

D. ps –ef

Check on Learning

Question 4

The myapp process has a nice value of 1. Which of the following nice values would increase the priority of the myapp process? (Choose two.)

A. –15

B. 5

C. 19

D. 0

E. 2

Check on Learning

Question 5

Which of the following shell commands will load the myapp program with a nice value of –5?

A. myapp –n –5

B. nice –5 myapp

C. renice –5 myapp

D. nice –n –5 myapp

Check on Learning

Question 6

The myapp process (PID 2345) is currently running on your system. Which of the following commands will reset its nice value to –5 without unloading the process?

A. **myapp –n –5 –p 2345**

B. **renice –n –5 2345**

C. **renice –5 2345**

D. **nice –n –5 2345**

Check on Learning

Question 7

You want to load the myapp program from the shell prompt and run it in the background. Which command will do this?

A. **myapp –b**

B. **myapp &**

C. **myapp –bg**

D. **load myapp into background**

Check on Learning

Question 8

You want to run the rsync command to synchronize your home directory with another server on the network. You know this command will take several hours to complete and you don't want to leave your system logged in during this time. Which commands could you use to leave rsync running after your logout? (Choose two.)

A. SIGHUP

B. nohup

C. stayalive

D. kill –NOHUP

E. screen

Check on Learning

Question 9

It's currently 1:00 in the afternoon. You want to schedule the myapp program to run automatically tomorrow at noon (12:00). Which of the following at commands could you use? (Choose two.)

A. at 12 pm tomorrow

B. at tomorrow –1 hour

C. at now +1 day

D. at today +23 hours

E. at now +23 hours

Check on Learning

Question 10

Which of the following crontab lines will cause the /usr/bin/myappcleanup process to run at 4:15 a.m. on the first of every month?

A. 15 4 1 * * /usr/bin/myappcleanup

B. 15 4 * 1 * /usr/bin/myappcleanup

C. 1 4 15 * * /usr/bin/myappcleanup

D. 4 1 * * 15 /usr/bin/myappcleanup