

# CS 332/532 Systems Programming

Lecture 21

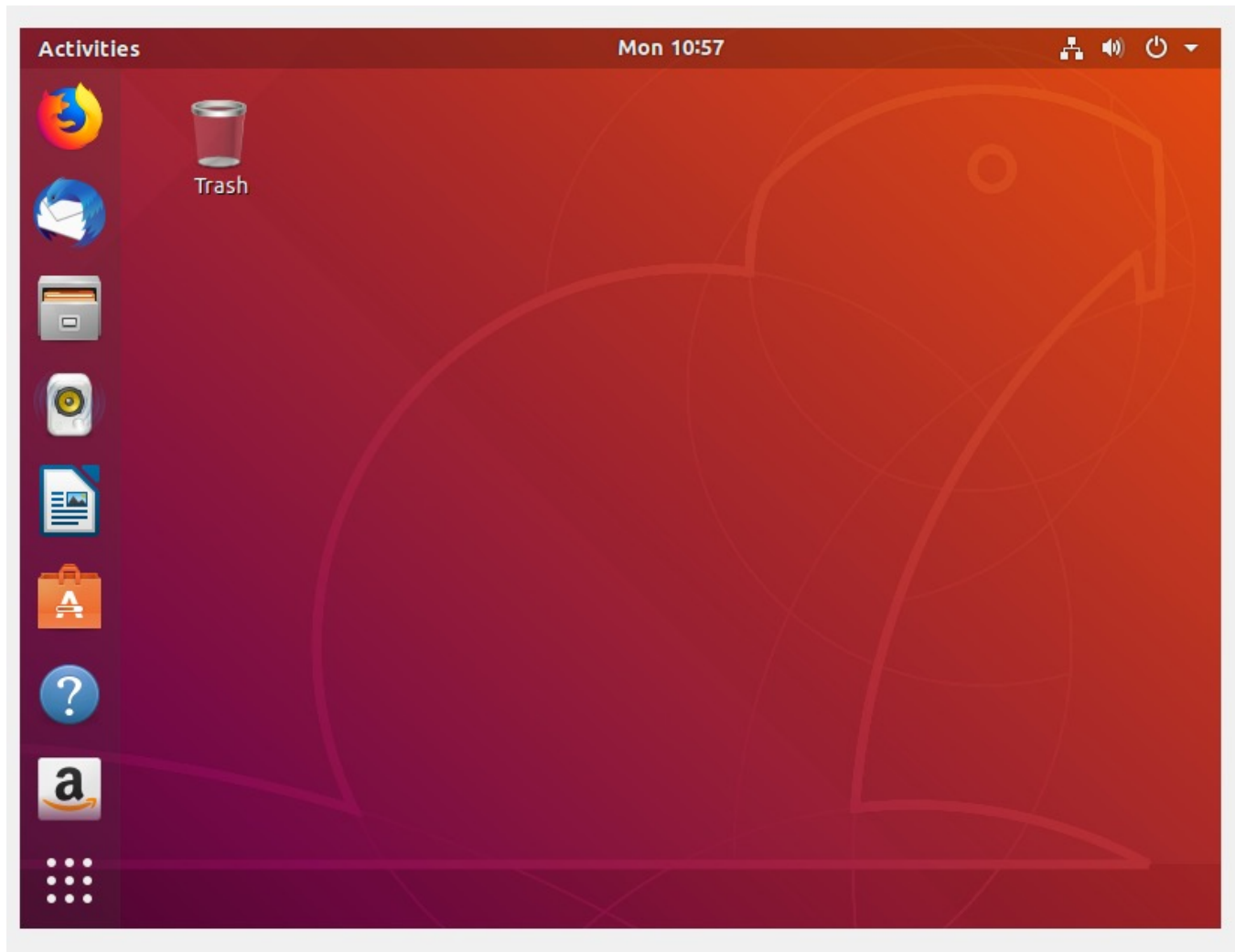
Linux Processes / 2

Professor : Mahmut Unan – UAB CS

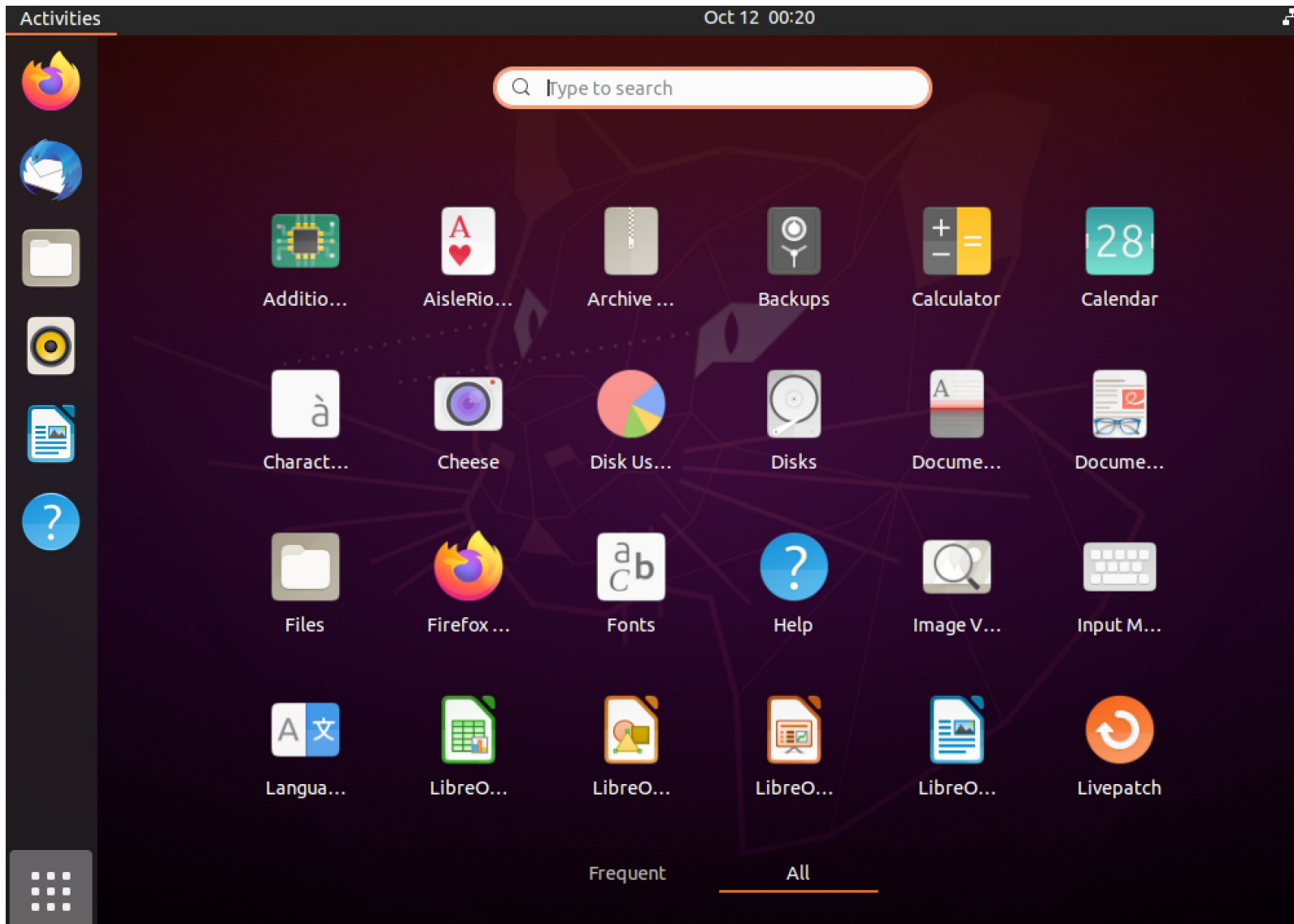
# Agenda

- Linux Processes
- Monitor processes in Linux environment
- Explore the /proc file system

# Ready To Go!!!

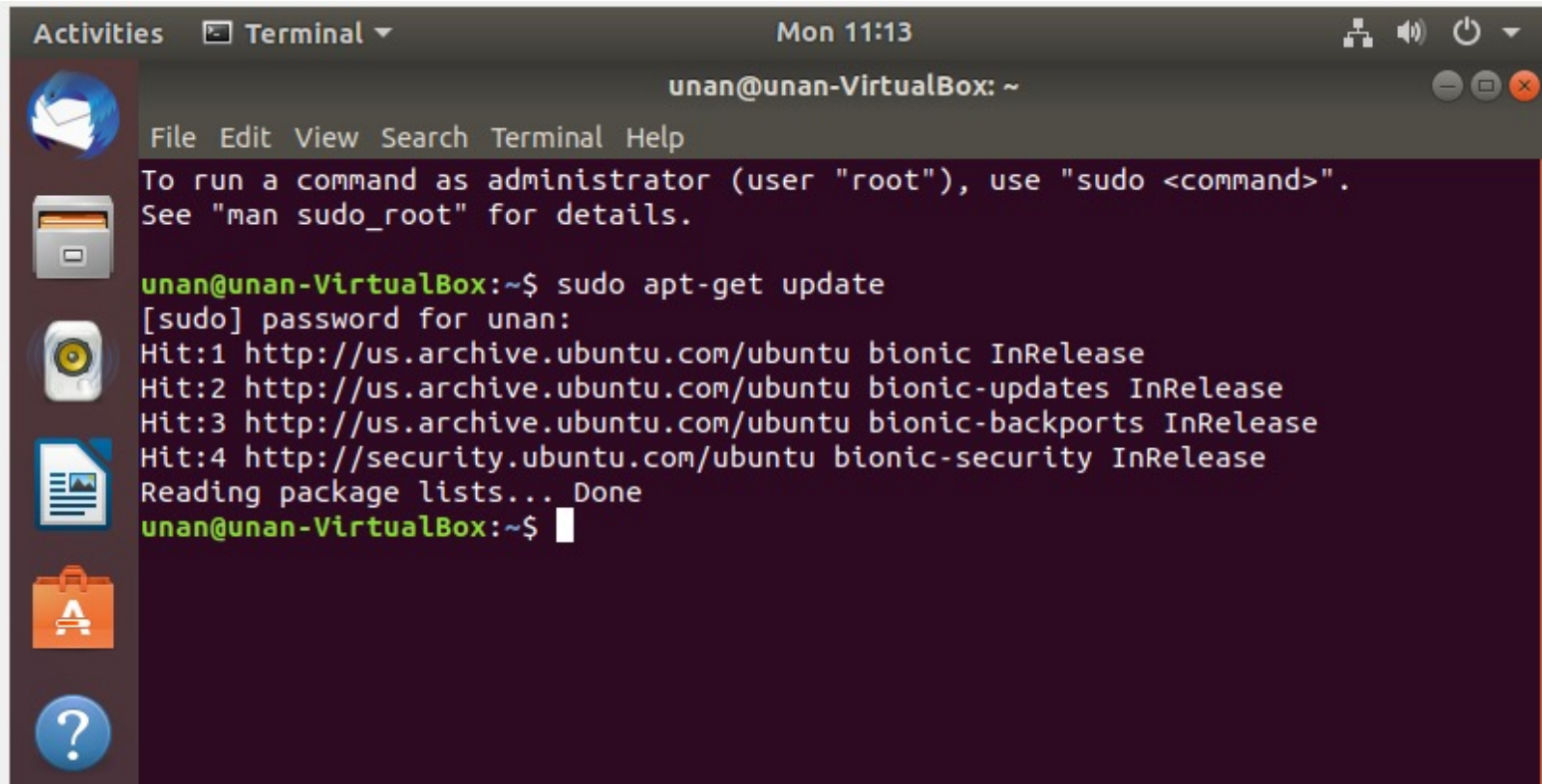


# You can start terminal



# Linux

- Start Linux Terminal and install the updates
  - `sudo apt-get update`

A screenshot of a Linux terminal window. The window title is "Terminal" and it shows the user "unan" at the prompt "unan@unan-VirtualBox: ~". The terminal displays the command "sudo apt-get update" and its output, which includes the password prompt, the list of repositories to be updated, and the status "Reading package lists... Done". The terminal window has a dark background and a sidebar with various application icons.

```
Activities  Terminal  Mon 11:13
unan@unan-VirtualBox: ~
File Edit View Search Terminal Help
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
unan@unan-VirtualBox:~$ sudo apt-get update
[sudo] password for unan:
Hit:1 http://us.archive.ubuntu.com/ubuntu bionic InRelease
Hit:2 http://us.archive.ubuntu.com/ubuntu bionic-updates InRelease
Hit:3 http://us.archive.ubuntu.com/ubuntu bionic-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu bionic-security InRelease
Reading package lists... Done
unan@unan-VirtualBox:~$
```

- `sudo apt-get upgrade`

# Processes in a Linux environment

- User processes in a Linux environment could be in one of the following three states: foreground, background, or suspended.
- Most interactive applications that take input from the keyboard or command-line argument and display output in a terminal are considered as foreground processes. Till now, we have been executed all our programs as foreground processes by typing the name of the command in the bash shell.

- Non-interactive processes that are typically not connected to a terminal and execute in the background are considered as background processes.
- You can execute a program in the background by typing the program name followed by the symbol `&` at the end.
- You will notice that the shell returns the command-prompt with the background process number and the corresponding process identifier (PID) of the process that was created.
- For example:

```
$ nano myprog.c &  
[1] 16946  
$
```

- You can display various jobs that are currently running in the background using the *jobs* command. It will show the job number, the current state of the job, and the job name. For example:

```
$ jobs
[1]+  Stopped                  nano myprog.c
$
```



- If you like to list additional information such as the PID of the job you can use the `-l` option. For example:

```
$ jobs -l
[1]+ 16946 Stopped (tty output)  nano myprog.c
$
```

In case of the above example, we have invoked an editor and it has been stopped since it requires terminal to display the output and continue. If we had created a non-interactive job in the background, then it would be in the running state. For example:

```
$ sleep 20 &
[2] 17513
$ jobs
[1]+  Stopped                nano myprog.c
[2]-  Running                sleep 20 &
$
```

- We can bring a job that is running in the background to foreground by using the *fg* command. For example, to switch the *sleep* process to foreground, we specify the job number after the % symbol:

```
$ fg %2
```

```
sleep 20
```

```
$ jobs
```

```
[1]+  Stopped
```

```
nano myprog.c
```

```
$
```

- You will notice that the sleep process will return the command prompt in the terminal when it completes execution and when you type *jobs* again, it will only show one process. You can use *fg* command to switch to the editor using: *fg %1*.
- You can continue with your edits, save the file, and exit the editor.
- After you exit, if you type *jobs* again, you will notice that it does not list any processes since the *nano* process is no longer executing.
- If you like to suspend a foreground process then type *Control-Z* when the program is executing and that process will be suspended

- For example, if we started the sleep process in foreground and would like to suspend it, then type *Control-Z* and you will see a message similar to what you saw when you started a process in background:

```
$  
$ jobs  
$ sleep 100  
^Z  
[1]+  Stopped                  sleep 100  
$ jobs  
[1]+  Stopped                  sleep 100  
$
```

- However, notice that the sleep process is stopped (it is not running) unlike the previous case when it was running in the background. If you like the sleep process to continue then you have to using the *bg* command as follows:

```
$ bg %1
[1]+  sleep 100 &
$ jobs
[1]+  Running                  sleep 100 &
$
[1]+  Done                     sleep 100
$ jobs
```

- Now you notice that the sleep process is running and when it is done you will see the message *Done* in your terminal.
- There are special background processes that are started at system startup and they continue to run till the system is shutdown.
- These special background processes are called ***daemons***.
- These processes typically end in “d” and some examples are: systemd, crond, ntpd, nfsd, sshd, httpd, named.
- If you like to terminate a process that is executing in the foreground, you use Control-C to kill it.
- If you like to terminate a process in the background, you could bring it to foreground and then use Control-C or use the *kill* command to terminate the background process directly

- For example:

- ```
$ jobs
$ sleep 100 &
[1] 1519
$ jobs
[1]+  Running                  sleep 100 &
$ kill %1
[1]+  Terminated              sleep 100
$
$ jobs
```

\$ You can also provide the PID as the argument to *kill* command to terminate a process.

## Monitor processes in Linux environment

- You can use the *ps* command to display information about various processes running on a Linux system. Login to one of CS Linux systems and enter the *ps* command, you will see the following information displayed:

```
~~{2.00}~{unan@vulcan18:~}~~  
[$ ps  
    PID TTY          TIME CMD  
  30544 pts/1        00:00:00 bash  
  30692 pts/1        00:00:00 ps
```



# ps -man page

- ps - report a snapshot of the current processes.
- **ps** [*options*]

To see every process on the system using standard syntax:

```
ps -e  
ps -ef  
ps -eF  
ps -ely
```

To see every process on the system using BSD syntax:

```
ps ax  
ps axu
```

To print a process tree:

```
ps -ejH  
ps axjf
```

To get info about threads:

```
ps -eLf  
ps axms
```

To get security info:

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

- By default, *ps* lists processes for the current user that are associated with the terminal that invoked the command and the output is unsorted.
- The following information is shown above: the process ID (PID), the terminal associated with the process (TTY), the cumulative CPU time in hh:mm:ss format (TIME), and the executable name (CMD).
- You will notice that there are two processes currently executing – bash and ps (the command you just executed) along with their corresponding process ID (in the first column under PID).

- The *ps* command has a large number of options that you can use to get more detailed information about the various processes currently running. We will look at some of these options below, you can find out more about these options using *man ps*.
- The *-u username* option lists all processes that belong to the user *username*:

```
[ $ ps -u unan
  PID TTY          TIME CMD
 30543 ?            00:00:00 sshd
 30544 pts/1        00:00:00 bash
 30716 pts/1        00:00:00 ps
```

- You can select all processes owned by you (runner of the **ps command**, root in this case), type:

```
ps -x
```

- We can also view every process running with **root** user privileges (real & effective ID) in user format.

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

- The command below allows you to view the PID, PPID, username and command of a process.

```
$ ps -eo pid,ppid,user,cmd
```

- To select a specific process by its name, use the -C flag, this will also display all its child processes.

```
$ ps -C sshd
```

- Now you notice that there is an additional process (sshd) that belongs to you and there is no terminal associated with that process (hence the ? under the TTY column). This process was started by the OS when you connected to this computer using an SSH client. You can use the -f or -F option to get a full listing:

```
~{2.00}~{unan@vulcan18:~}~  
[$ ps -fu unan  
UID          PID    PPID    C  STIME TTY          TIME CMD  
unan        30543  30532    0  13:59 ?           00:00:00 sshd: unan@pts/1  
unan        30544  30543    0  13:59 pts/1       00:00:00 -bash  
unan        30731  30544    0  14:00 pts/1       00:00:00 ps -fu unan  
~{2.00}~{unan@vulcan18:~}~
```

```

$ ps -Fu unan
UID      PID  PPID  C   SZ   RSS  PSR  STIME  TTY      TIME  CMD
unan    30543 30532  0 49950 2784   0 13:59  ?        00:00:00 sshd: unan@pts/1
unan    30544 30543  0 33493 3748   0 13:59 pts/1    00:00:00 -bash
unan    30746 30544  0 42042 1940   0 14:00 pts/1    00:00:00 ps -Fu unan

```

Now we see several additional fields displayed such the user ID, parent PI, process with command-line options, etc.

By looking at the PID and PPID we can identify that the *ps* command was created by the *bash* process and the *bash* process was in turn created by the *sshd* process. We can display the process tree with the *ps* command using the *--forest* option:

```
~~{2.00}~{unan@vulcan18:~}~~
```

```
$ ps -fu unan --forest
```

| UID  | PID   | PPID  | C | STIME | TTY   | TIME     | CMD                     |
|------|-------|-------|---|-------|-------|----------|-------------------------|
| unan | 30543 | 30532 | 0 | 13:59 | ?     | 00:00:00 | sshd: unan@pts/1        |
| unan | 30544 | 30543 | 0 | 13:59 | pts/1 | 00:00:00 | \_ -bash                |
| unan | 30789 | 30544 | 0 | 14:01 | pts/1 | 00:00:00 | \_ ps -fu unan --forest |

You can use *ps* to list every process currently running (not just processes that belong to you) using the *-e* option

```
ps -e | more
```

```
ps -ef | more
```

```
ps -eF | more
```

```
ps -ely | more
```

You have to press the **spacebar** to scroll through the list.



- You can send the output to the `wc` command to determine the total number of processes currently running on the system, for example:

```
~~{2.00}~{unan@vulcan18:~}~~  
[$ ps -e | wc -l  
158
```

At this the above command was executed on one of the CS Linux system, we had 158 processes currently running on the system, even though there are only three processes that belong to you. What are all these processes? Who is running these processes?

- You can also use the ***pstree*** command to display the process tree (you can find out more about the various options supported by *pstree* using *man pstree*).
- For example:

```
pstree -np | more
```

```
[ $ ps -e | more
  PID TTY          TIME CMD
    1 ?           00:14:34 systemd
    2 ?           00:00:03 kthreadd
    4 ?           00:00:00 kworker/0:0H
    6 ?           00:00:29 ksoftirqd/0
    7 ?           00:00:05 migration/0
    8 ?           00:00:00 rcu_bh
    9 ?           00:10:59 rcu_sched
   10 ?           00:00:00 lru-add-drain
   11 ?           00:00:34 watchdog/0
   12 ?           00:00:29 watchdog/1
   13 ?           00:00:06 migration/1
   14 ?           00:00:23 ksoftirqd/1
   16 ?           00:00:00 kworker/1:0H
```

- Similarly, you can use the `top` command (instead of `ps`) to display all processes currently running (you have to enter `q` to quit `top`).
- The `top` command provides a real-time update on the various processes running on the system.
- You can look at processes that belong to a specific user by typing `u` and the user name when `top` is running or start `top` with the `-u user` option.
- It provides a dynamic real-time view of the running system. Usually, this command shows the summary information of the system and the list of processes or threads which are currently managed by the Linux Kernel.
- <https://man7.org/linux/man-pages/man1/top.1.html>

# kill

- To terminate a process, we can use the *kill* command. The *kill* command can take the PID or the command name and terminate a specific process with the given PID or terminate all processes with the specified command name. We can also specify the type of signal, either as a signal name (*e.g.*, KILL for kill) or signal number (9 for kill), to send to a process with the *kill* command. Here is an example that shows how to use the *kill* command to terminate a process using PID.

# kill - man page

- kill - terminate a process
- The command **kill** sends the specified *signal* to the specified processes or process groups.
- If no signal is specified, the TERM signal is sent. The default action for this signal is to terminate the process. This signal should be used in preference to the KILL signal (number 9), since a process may install a handler for the TERM signal in order to perform clean-up steps before terminating in an orderly fashion
- <https://man7.org/linux/man-pages/man1/kill.1.html>

```
~~{2.00}~{unan@vulcan18:~}~~  
[$ sleep 100 &  
[1] 30898  
~~{2.00}~{unan@vulcan18:~}~~  
[$ kill -TERM 30898  
[1]+  Terminated                  sleep 100  
~~{2.00}~{unan@vulcan18:~}~~  
$
```

You can find the complete list of signal names and numbers using the `-l` option of *kill* command.

We will discuss more about signals in the later labs

```
kill SIGNAL PID
```

- Where SIGNAL is the signal to be sent and PID is the Process ID to be killed.
- Let's say we need to terminate the process 3827. We need to send the kill signal;

```
kill -9 3827
```

To display all the available signals, we should use below command option

```
kill -l
```

```
$ kill -l
```

|                 |                 |                 |                 |                 |
|-----------------|-----------------|-----------------|-----------------|-----------------|
| 1) SIGHUP       | 2) SIGINT       | 3) SIGQUIT      | 4) SIGILL       | 5) SIGTRAP      |
| 6) SIGABRT      | 7) SIGBUS       | 8) SIGFPE       | 9) SIGKILL      | 10) SIGUSR1     |
| 11) SIGSEGV     | 12) SIGUSR2     | 13) SIGPIPE     | 14) SIGALRM     | 15) SIGTERM     |
| 16) SIGSTKFLT   | 17) SIGCHLD     | 18) SIGCONT     | 19) SIGSTOP     | 20) SIGTSTP     |
| 21) SIGTTIN     | 22) SIGTTOU     | 23) SIGURG      | 24) SIGXCPU     | 25) SIGXFSZ     |
| 26) SIGVTALRM   | 27) SIGPROF     | 28) SIGWINCH    | 29) SIGIO       | 30) SIGPWR      |
| 31) SIGSYS      | 34) SIGRTMIN    | 35) SIGRTMIN+1  | 36) SIGRTMIN+2  | 37) SIGRTMIN+3  |
| 38) SIGRTMIN+4  | 39) SIGRTMIN+5  | 40) SIGRTMIN+6  | 41) SIGRTMIN+7  | 42) SIGRTMIN+8  |
| 43) SIGRTMIN+9  | 44) SIGRTMIN+10 | 45) SIGRTMIN+11 | 46) SIGRTMIN+12 | 47) SIGRTMIN+13 |
| 48) SIGRTMIN+14 | 49) SIGRTMIN+15 | 50) SIGRTMAX-14 | 51) SIGRTMAX-13 | 52) SIGRTMAX-12 |
| 53) SIGRTMAX-11 | 54) SIGRTMAX-10 | 55) SIGRTMAX-9  | 56) SIGRTMAX-8  | 57) SIGRTMAX-7  |
| 58) SIGRTMAX-6  | 59) SIGRTMAX-5  | 60) SIGRTMAX-4  | 61) SIGRTMAX-3  | 62) SIGRTMAX-2  |
| 63) SIGRTMAX-1  | 64) SIGRTMAX    |                 |                 |                 |



# Explore the /proc file system

- The proc file system (procfs) is a virtual file system that is created by the OS at system boot time to provide an interface between the kernel space and user space.
- It is commonly mounted at /proc.
- It provides information of processes currently running on the system and tools such as *ps* use this to display information about these processes

# ls -l /proc

```
~~{1.06}~{unan@vulcan16:~}~~
```

```
$ ls -l /proc
```

```
total 0
```

|            |   |         |         |   |     |    |       |       |
|------------|---|---------|---------|---|-----|----|-------|-------|
| dr-xr-xr-x | 9 | root    | root    | 0 | Apr | 28 | 14:54 | 1     |
| dr-xr-xr-x | 9 | root    | root    | 0 | Sep | 23 | 22:27 | 10    |
| dr-xr-xr-x | 9 | root    | root    | 0 | Oct | 1  | 21:17 | 10596 |
| dr-xr-xr-x | 9 | root    | root    | 0 | Oct | 1  | 21:17 | 10597 |
| dr-xr-xr-x | 9 | root    | root    | 0 | Sep | 23 | 22:27 | 11    |
| dr-xr-xr-x | 9 | root    | root    | 0 | Sep | 23 | 22:27 | 111   |
| dr-xr-xr-x | 9 | root    | root    | 0 | Oct | 1  | 21:16 | 1175  |
| dr-xr-xr-x | 9 | root    | root    | 0 | Oct | 1  | 21:16 | 1176  |
| dr-xr-xr-x | 9 | root    | root    | 0 | Oct | 1  | 21:16 | 1177  |
| dr-xr-xr-x | 9 | root    | root    | 0 | Oct | 1  | 21:16 | 1179  |
| dr-xr-xr-x | 9 | root    | root    | 0 | Sep | 23 | 22:27 | 12    |
| dr-xr-xr-x | 9 | root    | root    | 0 | Oct | 1  | 21:16 | 1218  |
| dr-xr-xr-x | 9 | root    | root    | 0 | Oct | 1  | 21:16 | 1296  |
| dr-xr-xr-x | 9 | postfix | postfix | 0 | Oct | 1  | 21:16 | 1298  |
| dr-xr-xr-x | 9 | root    | root    | 0 | Oct | 1  | 21:16 | 1299  |
| dr-xr-xr-x | 9 | root    | root    | 0 | Sep | 23 | 22:27 | 13    |
| dr-xr-xr-x | 9 | root    | root    | 0 | Oct | 1  | 21:16 | 1313  |

| File                | Description                                                                                                                                                              |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| /proc/cpuinfo       | information about the CPU architecture, used by <i>lscpu</i> command                                                                                                     |
| /proc/loadavg       | system load averages data, used by <i>uptime</i> command                                                                                                                 |
| /proc/meminfo       | memory usage statistics on the system, used by <i>free</i> command                                                                                                       |
| /proc/stat          | kernel/system statistics                                                                                                                                                 |
| /proc/version       | version of the kernel currently running on the system, used by <i>uname</i> command                                                                                      |
| /proc/[pid]         | a subdirectory for each running process                                                                                                                                  |
| /proc/[pid]/cmdline | command string of the process along with arguments separated by null ('\0') character                                                                                    |
| /proc/[pid]/cmd     | a symbolic link to the current working directory of the process                                                                                                          |
| /proc/[pid]/environ | environment variables and values used by the process separated by null ('\0') character (use strings /proc/[pid]/environ to display the environment variable and values) |
| /proc/[pid]/maps    | information on memory mapped regions of the process                                                                                                                      |
| /proc/[pid]/mem     | information to access pages of the process through I/O calls                                                                                                             |
| /proc/[pid]/stat    | status information about the process, used by <i>ps</i> command                                                                                                          |
| /proc/[pid]/statm   | status of memory used by the process, measured in pages                                                                                                                  |

- Let's take a look at one of the files :

```
# cat /proc/meminfo
```

```
~{ }~{unan@vulcan0:~}~  
[$ cat /proc/meminfo  
MemTotal:          3880088 kB  
MemFree:           2366368 kB  
MemAvailable:      3141080 kB  
Buffers:            0 kB  
Cached:            841836 kB  
SwapCached:         7264 kB  
Active:            687940 kB  
Inactive:          181940 kB  
Active(anon):       93296 kB  
Inactive(anon):     113528 kB  
Active(file):       594644 kB  
Inactive(file):     68412 kB  
Unevictable:        120 kB  
Mlocked:            120 kB  
SwapTotal:          6160380 kB  
SwapFree:           6083580 kB  
Dirty:              404 kB  
Writeback:          0 kB  
AnonPages:          25520 kB
```

- You can list the contents of `/proc` using the `ls` command and view files using the `cat` command.
- For files that contain strings separated with null characters you have use the `strings` command to display the contents of such files correctly.
- Here is an example to look at the `environ` file for the `bash` process:  

```
$ strings /proc/$$/environ
```
- Note: `$$` in `bash` refers to the process ID of the current process, you can replace it with the actual PID of `bash` and test the above command.

# Linux Shell Tutorial