INSIDE THE LINUX SYSTEM AND THE BASH SHELL

Professor Ken Birman
CS4414 Lecture 4

IDEA MAP FOR TODAY

If our program will run on Linux, we should learn about Linux

Process abstraction. Daemons

How programs learn what to do: rc files, environment variables, arguments

Along the way… many useful Linux commands and bash features

RECAP

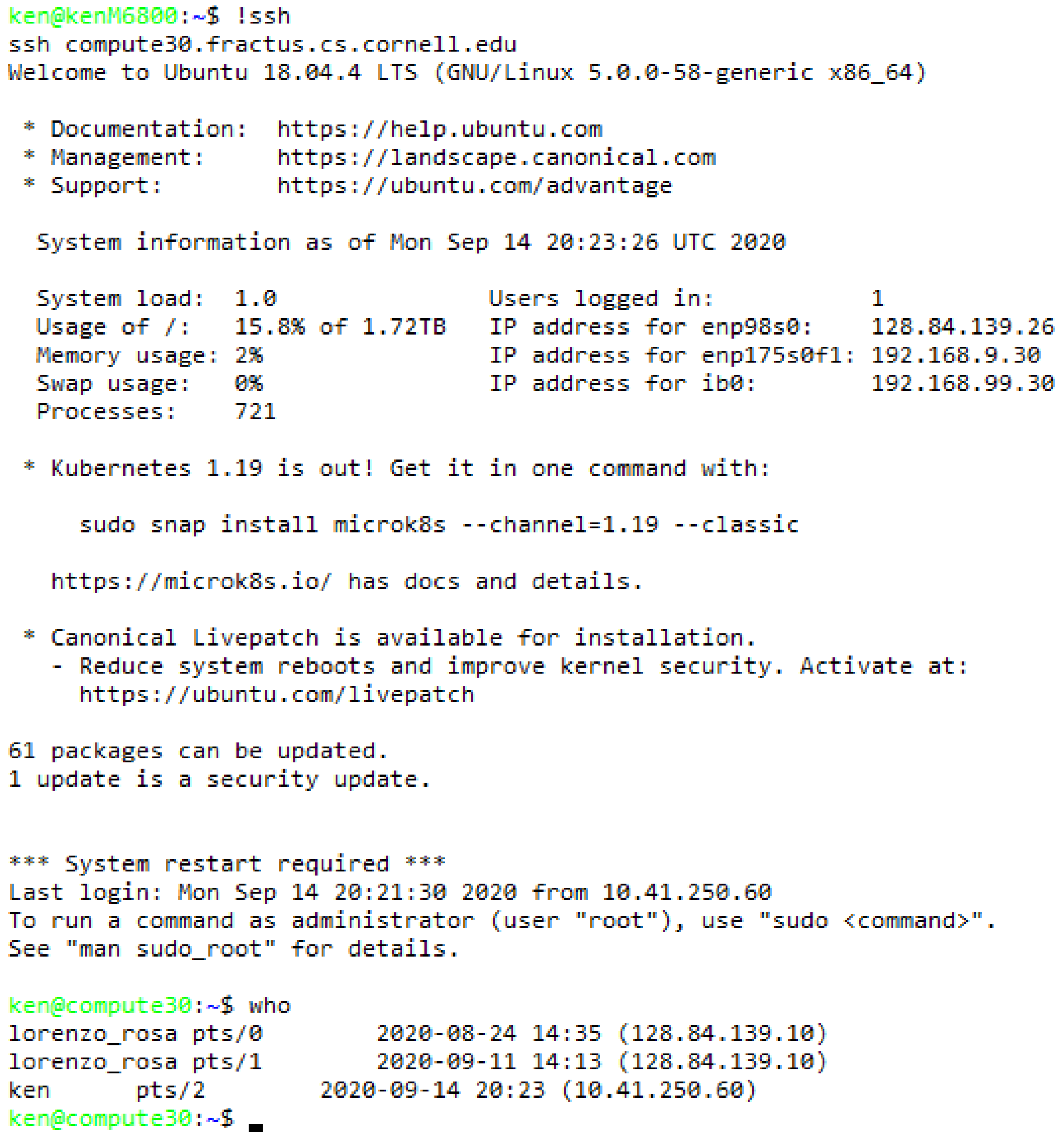
We saw that when our word-count program was running, parallelism offered a way to get much better performance from the machine, as much as a 30x speedup for this task.

In fact, Linux systems often have a lot of things running on them, in the background (meaning, “not talking to the person typing commands on the console.”)

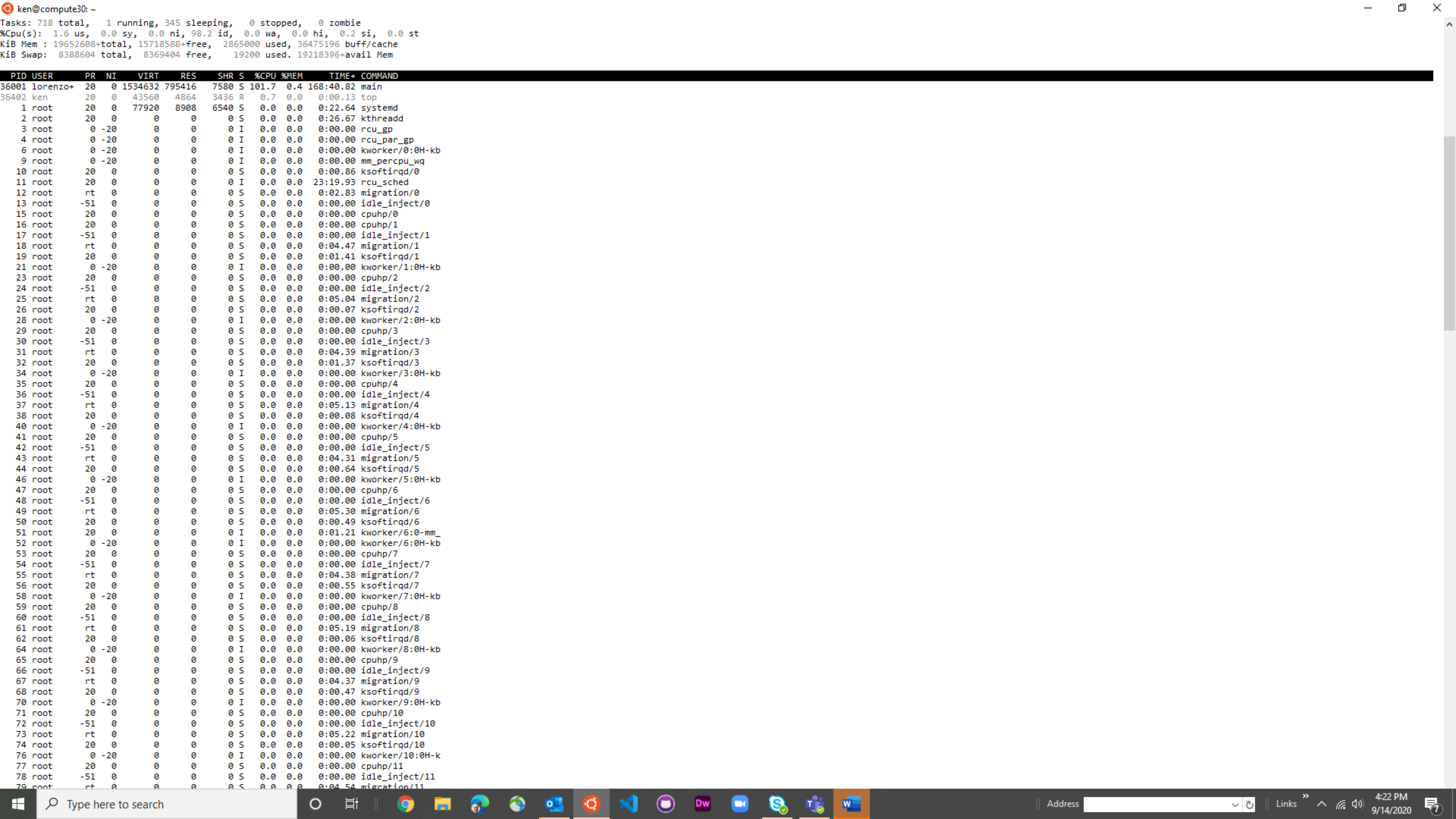
STUFF THAT WAS RUNNING ON A SERVER

At a random moment on a server similar to the setup in the CS engineering server pool, what happens to be running?

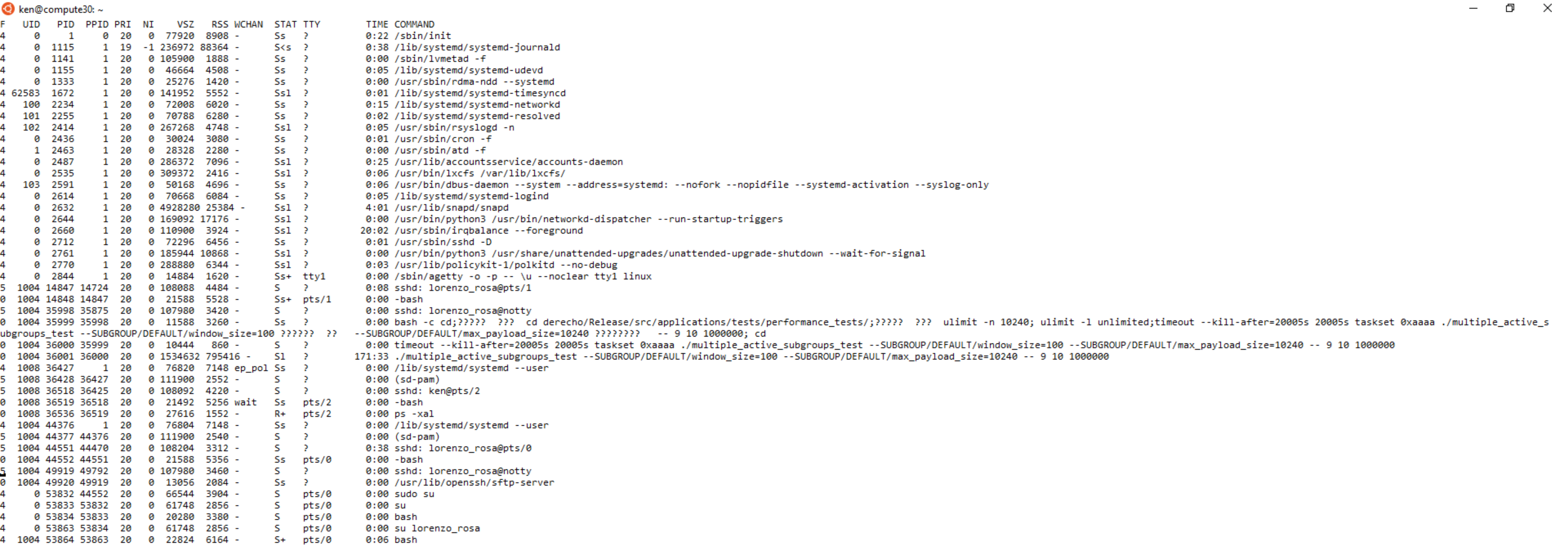
Ken made these next two slides to show you



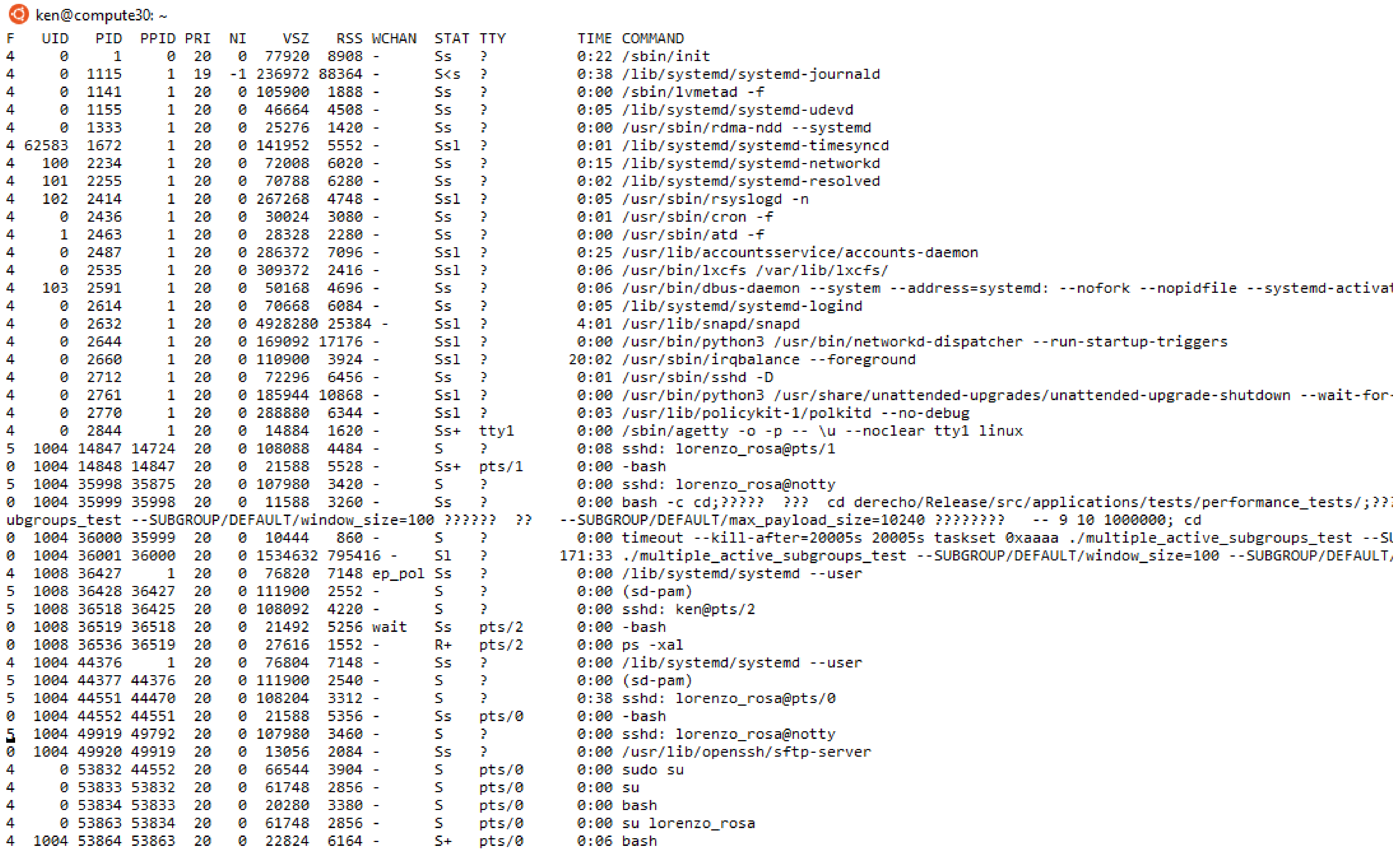
```
ken@kenM6800:~$ !ssh
ssh compute30.fractus.cs.cornell.edu
Welcome to Ubuntu 18.04.4 LTS (GNU/Linux 5.0.0-58-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Mon Sep 14 20:23:26 UTC 2020

  System load:  1.0                Users logged in:           1
  Usage of /:   15.8% of 1.72TB    IP address for enp98s0:    128.84.139.26
  Memory usage: 2%                 IP address for enp175s0f1: 192.168.9.30
  Swap usage:   0%                 IP address for ib0:        192.168.99.30
  Processes:    721

 * Kubernetes 1.19 is out! Get it in one command with:

     sudo snap install microk8s --channel=1.19 --classic

   https://microk8s.io/ has docs and details.

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

61 packages can be updated.
1 update is a security update.


*** System restart required ***
Last login: Mon Sep 14 20:21:30 2020 from 10.41.250.60
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ken@compute30:~$ who
lorenzo_rosa pts/0        2020-08-24 14:35 (128.84.139.10)
lorenzo_rosa pts/1        2020-09-11 14:13 (128.84.139.10)
ken      pts/2        2020-09-14 20:23 (10.41.250.60)
ken@compute30:~$ 
```



```
Tasks: 718 total,   1 running, 345 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.6 us,  0.0 sy,  0.0 ni, 98.2 id,  0.0 wa,  0.0 hi,  0.2 si,  0.0 st
KiB Mem : 19652608+total, 15718588+free,  2865000 used, 36475196 buff/cache
KiB Swap:  8388604 total,  8369404 free,    19200 used. 19218396+avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
36001 lorenzo+  20   0 1534632 795416   7580 S 101.7  0.4 168:40.82 main
36402 ken       20   0   43560   4864   3436 R   0.7  0.0   0:00.13 top
    1 root      20   0   77920   8908   6540 S   0.0  0.0   0:22.64 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:26.67 kthreadd
    3 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 rcu_gp
    4 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 rcu_par_gp
    6 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/0:0H-kb
    9 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 mm_percpu_wq
   10 root      20   0       0      0      0 S   0.0  0.0   0:00.86 ksoftirqd/0
   11 root      20   0       0      0      0 I   0.0  0.0  23:19.93 rcu_sched
   12 root      rt   0       0      0      0 S   0.0  0.0   0:02.83 migration/0
   13 root     -51   0       0      0      0 S   0.0  0.0   0:00.00 idle_inject/0
   15 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/0
   16 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/1
   17 root     -51   0       0      0      0 S   0.0  0.0   0:00.00 idle_inject/1
   18 root      rt   0       0      0      0 S   0.0  0.0   0:04.47 migration/1
   19 root      20   0       0      0      0 S   0.0  0.0   0:01.41 ksoftirqd/1
   21 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/1:0H-kb
   23 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/2
   24 root     -51   0       0      0      0 S   0.0  0.0   0:00.00 idle_inject/2
   25 root      rt   0       0      0      0 S   0.0  0.0   0:05.04 migration/2
   26 root      20   0       0      0      0 S   0.0  0.0   0:00.07 ksoftirqd/2
   28 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/2:0H-kb
   29 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/3
   30 root     -51   0       0      0      0 S   0.0  0.0   0:00.00 idle_inject/3
   31 root      rt   0       0      0      0 S   0.0  0.0   0:04.39 migration/3
   32 root      20   0       0      0      0 S   0.0  0.0   0:01.37 ksoftirqd/3
   34 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/3:0H-kb
   35 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/4
   36 root     -51   0       0      0      0 S   0.0  0.0   0:00.00 idle_inject/4
   37 root      rt   0       0      0      0 S   0.0  0.0   0:05.13 migration/4
   38 root      20   0       0      0      0 S   0.0  0.0   0:00.08 ksoftirqd/4
   40 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/4:0H-kb
   41 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/5
   42 root     -51   0       0      0      0 S   0.0  0.0   0:00.00 idle_inject/5
   43 root      rt   0       0      0      0 S   0.0  0.0   0:04.31 migration/5
   44 root      20   0       0      0      0 S   0.0  0.0   0:00.64 ksoftirqd/5
   46 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/5:0H-kb
   47 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/6
   48 root     -51   0       0      0      0 S   0.0  0.0   0:00.00 idle_inject/6
   49 root      rt   0       0      0      0 S   0.0  0.0   0:05.30 migration/6
   50 root      20   0       0      0      0 S   0.0  0.0   0:00.49 ksoftirqd/6
   51 root      20   0       0      0      0 I   0.0  0.0   0:01.21 kworker/6:0-mm_
   52 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/6:0H-kb
   53 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/7
   54 root     -51   0       0      0      0 S   0.0  0.0   0:00.00 idle_inject/7
   55 root      rt   0       0      0      0 S   0.0  0.0   0:04.38 migration/7
   56 root      20   0       0      0      0 S   0.0  0.0   0:00.55 ksoftirqd/7
   58 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/7:0H-kb
   59 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/8
   60 root     -51   0       0      0      0 S   0.0  0.0   0:00.00 idle_inject/8
   61 root      rt   0       0      0      0 S   0.0  0.0   0:05.19 migration/8
   62 root      20   0       0      0      0 S   0.0  0.0   0:00.06 ksoftirqd/8
   64 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/8:0H-kb
   65 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/9
   66 root     -51   0       0      0      0 S   0.0  0.0   0:00.00 idle_inject/9
   67 root      rt   0       0      0      0 S   0.0  0.0   0:04.37 migration/9
   68 root      20   0       0      0      0 S   0.0  0.0   0:00.47 ksoftirqd/9
   70 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/9:0H-kb
   71 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/10
   72 root     -51   0       0      0      0 S   0.0  0.0   0:00.00 idle_inject/10
   73 root      rt   0       0      0      0 S   0.0  0.0   0:05.22 migration/10
   74 root      20   0       0      0      0 S   0.0  0.0   0:00.05 ksoftirqd/10
   76 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/10:0H-k
   77 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/11
   78 root     -51   0       0      0      0 S   0.0  0.0   0:00.00 idle_inject/11
   79 root      rt   0       0      0      0 S   0.0  0.0   0:04.54 migration/11
```

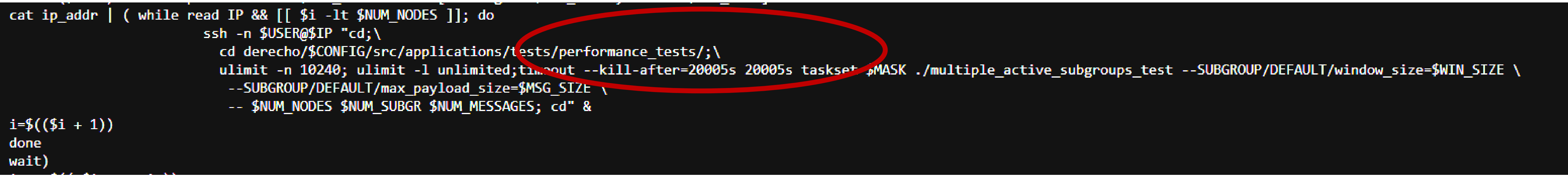

ken@compute30: ~

```
F   UID   PID  PPID PRI  NI    VSZ   RSS WCHAN  STAT TTY        TIME COMMAND
4     0     1     0  20   0  77920  8908 -      Ss   ?          0:22 /sbin/init
4     0  1115     1  19  -1 236972 88364 -      S<s  ?          0:38 /lib/systemd/systemd-journald
4     0  1141     1  20   0 105900  1888 -      Ss   ?          0:00 /sbin/lvmetad -f
4     0  1155     1  20   0  46664  4508 -      Ss   ?          0:05 /lib/systemd/systemd-udevd
4     0  1333     1  20   0  25276  1420 -      Ss   ?          0:00 /usr/sbin/rdma-ndd --systemd
4 62583  1672     1  20   0 141952  5552 -      Ssl  ?          0:01 /lib/systemd/systemd-timesyncd
4   100  2234     1  20   0  72008  6020 -      Ss   ?          0:15 /lib/systemd/systemd-networkd
4   101  2255     1  20   0  70788  6280 -      Ss   ?          0:02 /lib/systemd/systemd-resolved
4   102  2414     1  20   0 267268  4748 -      Ssl  ?          0:05 /usr/sbin/rsyslogd -n
4     0  2436     1  20   0  30024  3080 -      Ss   ?          0:01 /usr/sbin/cron -f
4     1  2463     1  20   0  28328  2280 -      Ss   ?          0:00 /usr/sbin/atd -f
4     0  2487     1  20   0 286372  7096 -      Ssl  ?          0:25 /usr/lib/accountsservice/accounts-daemon
4     0  2535     1  20   0 309372  2416 -      Ssl  ?          0:06 /usr/bin/lxcfs /var/lib/lxcfs/
4   103  2591     1  20   0  50168  4696 -      Ss   ?          0:06 /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation --syslog-only
4     0  2614     1  20   0  70668  6084 -      Ss   ?          0:05 /lib/systemd/systemd-logind
4     0  2632     1  20   0 4928280 25384 -     Ssl  ?          4:01 /usr/lib/snapd/snapd
4     0  2644     1  20   0 169092 17176 -      Ssl  ?          0:00 /usr/bin/python3 /usr/bin/networkd-dispatcher --run-startup-triggers
4     0  2660     1  20   0 110900  3924 -      Ssl  ?         20:02 /usr/sbin/irqbalance --foreground
4     0  2712     1  20   0  72296  6456 -      Ss   ?          0:01 /usr/sbin/sshd -D
4     0  2761     1  20   0 185944 10868 -      Ssl  ?          0:00 /usr/bin/python3 /usr/share/unattended-upgrades/unattended-upgrade-shutdown --wait-for-signal
4     0  2770     1  20   0 288880  6344 -      Ssl  ?          0:03 /usr/lib/policykit-1/polkitd --no-debug
4     0  2844     1  20   0  14884  1620 -      Ss+  tty1       0:00 /sbin/agetty -o -p -- \u --noclear tty1 linux
5  1004 14847 14724  20   0 108088  4484 -      S    ?          0:08 sshd: lorenzo_rosa@pts/1
0  1004 14848 14847  20   0  21588  5528 -      Ss+  pts/1      0:00 -bash
5  1004 35998 35875  20   0 107980  3420 -      S    ?          0:00 sshd: lorenzo_rosa@notty
0  1004 35999 35998  20   0  11588  3260 -      Ss   ?          0:00 bash -c cd;?????  ???  cd derecho/Release/src/applications/tests/performance_tests/;?????  ???  ulimit -n 10240; ulimit -l unlimited;timeout --kill-after=20005s 20005s taskset 0xaaaa ./multiple_active_s
ubgroups_test --SUBGROUP/DEFAULT/window_size=100 ??????  ??   --SUBGROUP/DEFAULT/max_payload_size=10240 ????????   -- 9 10 1000000; cd
0  1004 36000 35999  20   0  10444   860 -      S    ?          0:00 timeout --kill-after=20005s 20005s taskset 0xaaaa ./multiple_active_subgroups_test --SUBGROUP/DEFAULT/window_size=100 --SUBGROUP/DEFAULT/max_payload_size=10240 -- 9 10 1000000
0  1004 36001 36000  20   0 1534632 795416 -    Sl   ?        171:33 ./multiple_active_subgroups_test --SUBGROUP/DEFAULT/window_size=100 --SUBGROUP/DEFAULT/max_payload_size=10240 -- 9 10 1000000
4  1008 36427     1  20   0  76820  7148 ep_pol Ss   ?          0:00 /lib/systemd/systemd --user
5  1008 36428 36427  20   0 111900  2552 -      S    ?          0:00 (sd-pam)
5  1008 36518 36425  20   0 108092  4220 -      S    ?          0:00 sshd: ken@pts/2
0  1008 36519 36518  20   0  21492  5256 wait   Ss   pts/2      0:00 -bash
0  1008 36536 36519  20   0  27616  1552 -      R+   pts/2      0:00 ps -xal
4  1004 44376     1  20   0  76804  7148 -      Ss   ?          0:00 /lib/systemd/systemd --user
5  1004 44377 44376  20   0 111900  2540 -      S    ?          0:00 (sd-pam)
5  1004 44551 44470  20   0 108204  3312 -      S    ?          0:38 sshd: lorenzo_rosa@pts/0
0  1004 44552 44551  20   0  21588  5356 -      Ss   pts/0      0:00 -bash
5  1004 49919 49792  20   0 107980  3460 -      S    ?          0:00 sshd: lorenzo_rosa@notty
0  1004 49920 49919  20   0  13056  2084 -      Ss   ?          0:00 /usr/lib/openssh/sftp-server
4     0 53832 44552  20   0  66544  3904 -      S    pts/0      0:00 sudo su
4     0 53833 53832  20   0  61748  2856 -      S    pts/0      0:00 su
4     0 53834 53833  20   0  20280  3380 -      S    pts/0      0:00 bash
4     0 53863 53834  20   0  61748  2856 -      S    pts/0      0:00 su lorenzo_rosa
4  1004 53864 53863  20   0  22824  6164 -      S+   pts/0      0:06 bash
```



```
ken@compute30: ~
F   UID   PID  PPID PRI  NI    VSZ   RSS WCHAN  STAT TTY        TIME COMMAND
4     0     1     0  20   0  77920  8908 -      Ss   ?          0:22 /sbin/init
4     0  1115     1  19  -1 236972 88364 -      S<s  ?          0:38 /lib/systemd/systemd-journald
4     0  1141     1  20   0 105900  1888 -      Ss   ?          0:00 /sbin/lvmetad -f
4     0  1155     1  20   0  46664  4508 -      Ss   ?          0:05 /lib/systemd/systemd-udevd
4     0  1333     1  20   0  25276  1420 -      Ss   ?          0:00 /usr/sbin/rdma-ndd --systemd
4 62583  1672     1  20   0 141952  5552 -      Ssl  ?          0:01 /lib/systemd/systemd-timesyncd
4   100  2234     1  20   0  72008  6020 -      Ss   ?          0:15 /lib/systemd/systemd-networkd
4   101  2255     1  20   0  70788  6280 -      Ss   ?          0:02 /lib/systemd/systemd-resolved
4   102  2414     1  20   0 267268  4748 -      Ssl  ?          0:05 /usr/sbin/rsyslogd -n
4     0  2436     1  20   0  30024  3080 -      Ss   ?          0:01 /usr/sbin/cron -f
4     1  2463     1  20   0  28328  2280 -      Ss   ?          0:00 /usr/sbin/atd -f
4     0  2487     1  20   0 286372  7096 -      Ssl  ?          0:25 /usr/lib/accountsservice/accounts-daemon
4     0  2535     1  20   0 309372  2416 -      Ssl  ?          0:06 /usr/bin/lxcfs /var/lib/lxcfs/
4   103  2591     1  20   0  50168  4696 -      Ss   ?          0:06 /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activat
4     0  2614     1  20   0  70668  6084 -      Ss   ?          0:05 /lib/systemd/systemd-logind
4     0  2632     1  20   0 4928280 25384 -     Ssl  ?          4:01 /usr/lib/snapd/snapd
4     0  2644     1  20   0 169092 17176 -      Ssl  ?          0:00 /usr/bin/python3 /usr/bin/networkd-dispatcher --run-startup-triggers
4     0  2660     1  20   0 110900  3924 -      Ssl  ?         20:02 /usr/sbin/irqbalance --foreground
4     0  2712     1  20   0  72296  6456 -      Ss   ?          0:01 /usr/sbin/sshd -D
4     0  2761     1  20   0 185944 10868 -      Ssl  ?          0:00 /usr/bin/python3 /usr/share/unattended-upgrades/unattended-upgrade-shutdown --wait-for-
4     0  2770     1  20   0 288880  6344 -      Ssl  ?          0:03 /usr/lib/policykit-1/polkitd --no-debug
4     0  2844     1  20   0  14884  1620 -      Ss+  tty1       0:00 /sbin/agetty -o -p -- \u --noclear tty1 linux
5  1004 14847 14724  20   0 108088  4484 -      S    ?          0:08 sshd: lorenzo_rosa@pts/1
0  1004 14848 14847  20   0  21588  5528 -      Ss+  pts/1      0:00 -bash
5  1004 35998 35875  20   0 107980  3420 -      S    ?          0:00 sshd: lorenzo_rosa@notty
0  1004 35999 35998  20   0  11588  3260 -      Ss   ?          0:00 bash -c cd;?????  ???  cd derecho/Release/src/applications/tests/performance_tests/;??
ubgroups_test --SUBGROUP/DEFAULT/window_size=100 ??????  ??   --SUBGROUP/DEFAULT/max_payload_size=10240 ????????    -- 9 10 1000000; cd
0  1004 36000 35999  20   0  10444   860 -      S    ?          0:00 timeout --kill-after=20005s 20005s taskset 0xaaaa ./multiple_active_subgroups_test --Sl
0  1004 36001 36000  20   0 1534632 795416 -    Sl   ?        171:33 ./multiple_active_subgroups_test --SUBGROUP/DEFAULT/window_size=100 --SUBGROUP/DEFAULT/
4  1008 36427     1  20   0  76820  7148 ep_pol Ss   ?          0:00 /lib/systemd/systemd --user
5  1008 36428 36427  20   0 111900  2552 -      S    ?          0:00 (sd-pam)
5  1008 36518 36425  20   0 108092  4220 -      S    ?          0:00 sshd: ken@pts/2
0  1008 36519 36518  20   0  21492  5256 wait   Ss   pts/2      0:00 -bash
0  1008 36536 36519  20   0  27616  1552 -      R+   pts/2      0:00 ps -xal
4  1004 44376     1  20   0  76804  7148 -      Ss   ?          0:00 /lib/systemd/systemd --user
5  1004 44377 44376  20   0 111900  2540 -      S    ?          0:00 (sd-pam)
5  1004 44551 44470  20   0 108204  3312 -      S    ?          0:38 sshd: lorenzo_rosa@pts/0
0  1004 44552 44551  20   0  21588  5356 -      Ss   pts/0      0:00 -bash
5  1004 49919 49792  20   0 107980  3460 -      S    ?          0:00 sshd: lorenzo_rosa@notty
0  1004 49920 49919  20   0  13056  2084 -      Ss   ?          0:00 /usr/lib/openssh/sftp-server
4     0 53832 44552  20   0  66544  3904 -      S    pts/0      0:00 sudo su
4     0 53833 53832  20   0  61748  2856 -      S    pts/0      0:00 su
4     0 53834 53833  20   0  20280  3380 -      S    pts/0      0:00 bash
4     0 53863 53834  20   0  61748  2856 -      S    pts/0      0:00 su lorenzo_rosa
4  1004 53864 53863  20   0  22824  6164 -      S+   pts/0      0:06 bash
```

WHAT'S WITH THE ????? STUFF?



```
cat ip_addr | ( while read IP && [[ $i -lt $NUM_NODES ]]; do
                        ssh -n $USER@$IP "cd;\
                          cd derecho/$CONFIG/src/applications/tests/performance_tests/;\
                          ulimit -n 10240; ulimit -l unlimited;timeout --kill-after=20005s 20005s taskset $MASK ./multiple_active_subgroups_test --SUBGROUP/DEFAULT/window_size=$WIN_SIZE \
                           --SUBGROUP/DEFAULT/max_payload_size=$MSG_SIZE \
                           -- $NUM_NODES $NUM_SUBGR $NUM_MESSAGES; cd" &
i=$(($i + 1))
done
wait)
```

… apparently those are escaped newline characters!

In fact any "non-printing" characters are shown as ?

LET'S SUMMARIZE SOME OF WHAT WE SAW

In addition to the Linux operating system "kernel", Linux had many helper programs running in the background.

We used the term *daemon* programs for these. The term is a reference to physics, but a bit obscure.

A daemon program is launched during startup (or periodically) and doesn't connect to a console. It lives in the background.

YOU CAN ALSO CREATE BACKGROUND TASKS OF YOUR OWN

One way to do this is with a command called "nohup", which means "when I log out ("hang up"), leave this running."

A second is with a command named "disown".

- When you log out, bash kills any background jobs that you still own.
- If you "disown" a job, it leaves it running

ONE REASON FOR DAEMONS: PERIODIC TASKS

In production systems, many things need to happen periodically

Linux and C++ have all sorts of features to help

- Within Linux, a tool called "cron" (for "chronological") runs jobs on a schedule that you can modify or extend
- Example: *Once every hour, check for new photos on the camera and download them.*

HOW CRON WORKS

There is a file in a standard location called the "crontab", meaning "table of jobs that run chronologically"

Each line in the file uses a special notation to designate when the job should run and what program to launch

The program itself could be in any language and can even be a Linux "bash script" (also called a "shell script").

HOW AT WORKS

Very similar to cron, but for a one-time command

The “atd” waits until the specified time, then runs it

Whereas **cron** is controlled from the crontab file, **at** is used at the command-line.

HOW DO THESE PROGRAMS KNOW WHAT WE WANT THEM TO DO?

On Linux, programs have three ways to discover runtime parameters that tell them what to do.

- Arguments provided when you run the program, on the command line
- Configuration files, specific to the program, that it can read to learn parameter settings, files to scan, etc.
- Linux environment variables. These are managed by bash and can be read by the program using "getenv" system calls.

PROGRAMS CONTROLLED BY CONFIGURATION FILES

In Linux, *many* programs use some sort of configuration file, just like cron is doing. Some of those files are hidden but you can see them if you know to ask.

- In any directory, hidden files will simply be files that start with a name like ".bashrc". The dot at the start says "invisible"
- If you use "ls –a" to list a directory, it will show these files. You can also use "echo .*" to do this, or find, or ….

A FEW COMMON HIDDEN FILES

Bash replaces "~" with the pathname to your home directory

~/.bashrc – The Bourne shell (bash) initialization script

~/.vimrc – A file used to initialize the vim visual editor

~/.emacs – A file used to initialize the emacs visual editor

/etc/init.d – When Linux starts up, the files here tell it how to configure the entire computer

/etc/init.d/cron – Used by cron to track periodic jobs

VISUAL STUDIO CODE USES THEM TOO

When you create or open a project, it makes a folder called .vscode.   You can see it if you look for it

Settings are in files with a “.json” extension

JSON is the Javascript Object Notation, and is a way to write down (in a file) information about an object or data structure

VISUAL STUDIO USE

The items are intended to h

Sometimes we need to edit
pulldown menus on "configu
load the JSON file, format

When working remotely the
remote machine, perhaps dif

```
"task

  }
```

launch.json

```
"version": "0.2.0",
  "configurations": [

      {
          "name": "(gdb) Launch",
          "type": "cppdbg",
          "request": "launch",
          "program": "${workspaceFolder}/scheduler.c",
          "args": [],
          "stopAtEntry": false,
          "cwd": "${workspaceRoot}",
          "environment": [],
          "externalConsole": false,
          "MIMode": "gdb",
          "miDebuggerPath": "/usr/bin/gdb",
          "setupCommands": [
              {
                  "description": "Enable pretty-printing for gdb",
                  "text": "-enable-pretty-printing",
                  "ignoreFailures": true
              },
              {
                  "description":  "Set Disassembly Flavor to Intel",
                  "text": "-gdb-set disassembly-flavor intel",
                  "ignoreFailures": true
              }
          ]
      }
  ]
}
```

ENVIRONMENT VARIABLES

The bash configuration file is used to set the environment variables.

Examples of environment variables on Ubuntu include

- HOME: my “home directory”
- USER: my login user-name
- PATH: A list of places Ubuntu searches for programs when I run a command
- PYTHONPATH: Where my version of Python was built

ENVIRONMENT VARIABLES

The bash configuration file is used t bles.

Other versions of Linux, like CentOS, RTOS, etc might have different environment variables, or additional ones. And different shells could use different variables too!

Examples of environment variables

- HOME: my "home directory"
- USER: my login user-name
- PATH: A list of places Ubuntu searches for programs when I run a command
- PYTHONPATH: Where my version of Python was built

EXAMPLE, FROM KEN’S LOGIN

```
HOSTTYPE=x86_64
USER=ken
HOME=/home/ken
SHELL=/bin/bash
PYTHONPATH=/home/ken/z3/build/python/
PATH=/home/ken/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
```

SO… LET’S WALK THROUGH THE SEQUENCE THAT CAUSES THESE TO BE “USED”

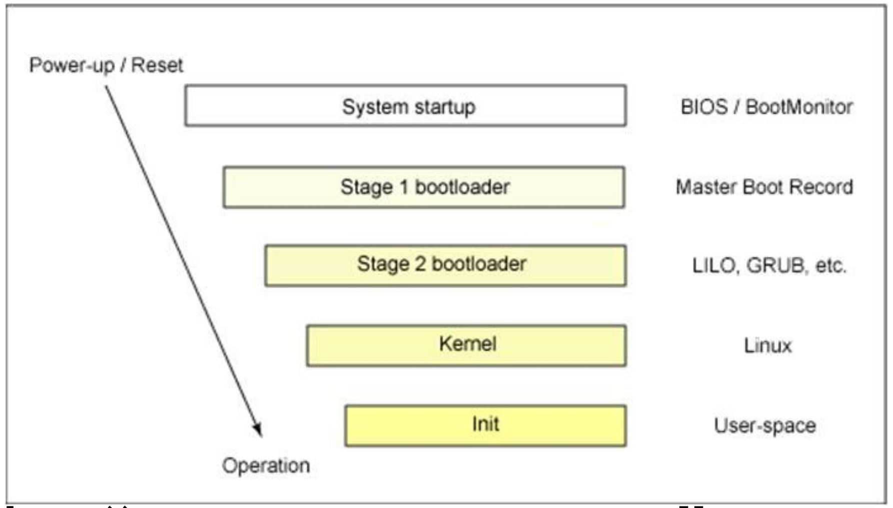
We will review

1) How Linux boots when you restart the computer

2) How bash got launched (this is when it read .bashrc)

3) How a command like “c++” gets launched

WHEN UBUNTU BOOTS



Ubuntu is a version of Linux. It runs as the "operating system" or "kernel". But when you start the computer, it isn't yet running.

Every computer has a special firmware program to launch a special stand-alone program call the "bootstrap" program. In fact this is a 2-stage process (hence "stage ½ bootloader")

This stand-alone program than reads the operating system binary from a file on disk into memory and launches it.

WHAT ABOUT UBUNTU ON WINDOWS?

Microsoft Windows has a “microkernel” on which they can host Ubuntu as a kind of application. (Same with MacOS, via bootcamp)

In effect, both Windows and Ubuntu are “apps” that in turn can host and run additional programs, all on your machine.

Early virtualization approaches of this kind were slow, but over time they have become highly performant. Now we do this all the time.

UBUNTU LINUX STARTS BY SCANNING THE HARDWARE

Linux figures out how much memory the machine has, what kind of CPU it has, what devices are attached, etc.

It accesses the same disk it booted on to learn configuration parameters and also which devices to activate. For these activated devices, it loads a "device driver".

Then it starts the "init" daemon.

THE INIT AND RLOGIN DAEMONS

The init daemon is the "parent" of all other processes that run on an Ubuntu Linux system. /etc/init.d told it what to initially do at boot time.

It launched **cron** and the **at** daemon, and it also launches the application that allows you to log in and have a bash shell connected to your console.

The rlogin daemon allows remote logins, if you configured Ubuntu to permit them. If firewalls and IP addresses allow, you can then use rlogin to remotely connect to a machine, like I did to access compute30 on Fractus.

WHEN YOU LOG IN

The login process sees that “ken” is logging in.

It checks the secure table of permitted users and makes sure I am a user listed for this machine – if not, “goodbye”!

In fact I am, and I prefer the bash shell. So it launches the bash shell, and configures it to take command-line input from my console. Now when I type commands, bash sees the string as input.

BASH INITIALIZES ITSELF

The .bashrc file is “executed” by bash to configure itself for me

I can customize this (and many people do!), to set environment variables, run programs, etc – it is actually a script of bash commands, just like the ones I can type on the command line.

By the time my command prompt appears, bash is configured.

WHEN WE LAUNCH PROGRAMS…

Bash (or cron, or whatever) looks for the program to launch using the PATH variable as guidance on where to look. A special Linux operation called "fork" followed by "exec" runs it.

The program is now active and will read the environment plus any arguments you provided to know what to do. Some programs fail at this stage because they can't find a needed file in the places listed in the relevant path, or an argument is wrong.

EXAMPLE

"It's a UNIX System! I know this."

I log in, and then edit a file using vim (Sagar prefers emacs). So:

1. init ran a login daemon.
2. That daemon launched bash.
3. Bash initialized using .bashrc, then gave a command-line prompt
4. When I ran "vim", bash found the program and ran it, using PATH to know where to look. "which vim" would tell me which it found.
5. Vim initialized itself, and created a visual editing window for me.

BASH NOTATION

First, just to explain about “prompts”, bash has a command prompt that it shows when it is waiting for a command:

```
ken@compute30: echo “Hello world”
```

Even if my slide doesn’t show a prompt, it is really there. You can customize it to show anything you like (your computer name, the folder you are in, etc). On old Linux systems, it was “% ”

BASH NOTATION

First, just to explain about "prompts", bash has a command prompt that it shows when it is waiting for a command:

```
ken@compute30: echo "Hello world"
```

Even if my slide doesn't show a prompt, it is really there. You can customize it to show anything you like (your computer name, the folder you are in, etc). On old Linux systems, it was "% "

BASH NOTATION

In a bash script, you can always set environment variables using the special bash command "export" (or the older "setenv"):

```
      export PATH=/bin
```

Normally you want to "add" a directory to path. To do this you expand the old value:

```
      export PATH=$PATH:$HOME/myapp/bin
```

This says that in my home directory is a directory myapp/bin with programs I might want to run. Bash will now look there, too.

BASH NOTATION

In fact bash allows a shorthand version too

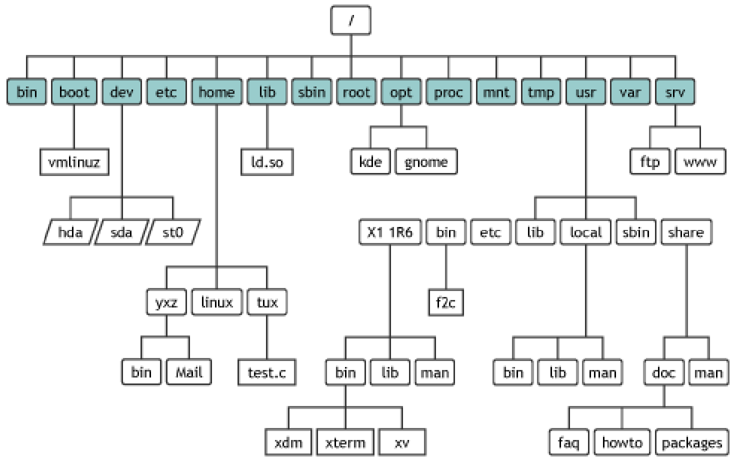
```
        % PATH=$PATH:$HOME/myapp/bin
```

or even

```
        % PATH=$PATH:~/myapp/bin            # ~ is short for $HOME
```

Why so many notations? Linux evolved over 40 years… people got tired of typing “export” or “setenv” or $HOME

DIRECTORIES, FILES



Linux organizes files into a tree. Even a directory is actually a special kind of file. Use "ls –l" to see details about a file.

Chdir ("cd") to enter a directory.

- "/" is the root of the file system tree.
- "." refers to the current directory.
- ".." is a way to access the parent directory.
- In the bash shell, "~" refers to your home directory.

http://researchhubs.com/post/computing/linux-cmd/linux-directory.html

RULES ABOUT FILE NAMES

Linux directories limit the length of a file name to 255 chars.

The maximum length of a pathname, from the root, is 4096

Alphanumeric and a few characters like . _ -

Unlike Windows and Mac, don’t use spaces in file names.

PROCESSES

When you launch a process (lke from bash), it gets executed and has a process id.

The “ps” and “top” commands let you see what you have running

You can kill a process in various ways: ^C, killpid, logging out (there is also a way to prevent this, called “nohup”)

LINUX COMMANDS

There are *hundreds* of them!

In fact you have to install them, in batches, because they use so much space if you install everything.

Learn about each command using its “manual” page. Just google it, like “Linux find command” (or “man 1 find”)

ALIASES: A COMMON CAUSE FOR CONFUSION

In Linux, one "file" or program can have multiple names that refer to the identical thing!

… and some programs even check to see which name you typed when launching them, and customize their behavior accordingly

For example, c++ is really an alias for gcc or clang…

COMMANDS ARE REALLY EXECUTABLE FILES: READ/WRITE/EXECUTE FILE “PERMISSIONS”

Each file in Linux has permissions, visible via “ls –l”. Permissions are shown as [dlcb]rwxrwxrwx. The d, if present, means that this file is a directory. The other letters are for special types of files

The next three are permissions for the user who created the file

The next three are for other users in the owner’s “group”

The last three are for users outside these two categories

INODE NUMBERS

We will look more closely at this Linux concept in a different lecture…

An entry in a directory is actually a tuple:

The file name	**init.d**
Its inode number	**34**

Each hard drive has a table of inodes. Each is a data structure holding all the information about the file with the corresponding inode number.

SPECIAL FILES (S/D/C/B/R…)

Linux uses file names to refer to devices like the disk, or your camera (if you attach it) or your computer display and keyboard.

There are also files types with other special meanings:

- Links: a way to give a file a second name (an "alias")
- c or b: character (keyboard) or block (disk) devices
- r: "raw". A way to access a device "directly".

THE PERMISSIONS THEMSELVES

Read means “allowed to see the contents”. For a file, this means the bytes. For a directory, this means you can list the files in the directory.

Write means “allowed to make changes”. For a directory this means creating or deleting files.

Execute is very complicated…

EXECUTE: THEY RAN OUT OF BITS SO THEY GAVE IT MULTIPLE MEANINGS

If the file is a program, execute means "(attempt to) run the program". This applies even if the filename doesn't end with **.exe**

If the file is a "shell file", execute means "launch the bash program (or it could be some other shell), and tell it to run it.

If the file is a directory, "execute" means "can access files in it". Note: this means you can sometimes read or run a file that you wouldn't be able to "see" by listing the directory it is in!

SUDO

Linux has the concept of a “superuser”. Used when installing programs

Running a command using “sudo” can “override” the normal restrictions.   You’ll need this to install extra commands.

Be aware that you can also break Linux easily by changing settings or modifying/removing a file that matters.

REMEMBER THE DAEMONS? KILLING THEM IS RISKY!

Sometimes a computer seems very busy, or even stuck, and novice users will check for what is running and kill it.

With "sudo" you can kill anything! Like a daemon-killing sword…

… but you need to know what you are killing. Linux depends on many of the background daemons!

SOME DIRECTORIES TO KNOW ABOUT

The current working directory: this is where you are right now, and where files created by commands or programs will be put by default.

For example, if you compile fast-wc.cpp and name the executable fwc, you could run it by typing ./fwc

If "." is in PATH, then you can just type fwc

SOME DIRECTORIES TO KNOW ABOUT

/tmp is a place for programs to put temporary files needed while executing. These are automatically deleted if you forget to do so (on reboot).

/dev/null: a black hole. We'll see a use for it soon!

A fun one: You can configure Linux to have a temporary file system entirely in memory ("RAM"). Called /ramfs

MOUNT COMMAND

Linux treats each storage device (including “ramdisk”) as a separate entity.

A storage device can be “raw” meaning “blocks of bytes” or it can have a file system on it (a tree data structure). At boot time there is just one storage device with an active file system.

The “mount” command attaches a storage device with a file system on it to your directory structure, so that you can access the files in it.

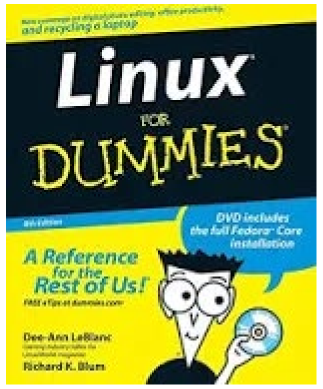
MORE DIRECTORIES TO KNOW ABOUT

/bin and /usr/bin: Standard places where programs are put. Of course you can add more places by installing programs or building your own, and modifying the search PATH variable

/include: The header files for system calls and standard libraries

/etc, /init.d: Configuration files used by Linux itself, and the ones used by daemons like cron

HOW DO PEOPLE LEARN THIS STUFF?


Linux
FOR
DUMMIES
A Reference for the Rest of Us!
DVD includes the full Fedora Core installation
Dee-Ann LeBlanc
Richard K. Blum

Linux is “self documented”! You can buy a book… but no need!

The Linux “man” program is a user manual for Linux, and has sections covering commands (**man 1 find**, for example), system calls (**man 2 open**), libraries (man 3) …

Bash has a “help” command that will print these same pages. **help**, by itself, lists all available commands. **help find** would print the man page for the find command.

SOME REALLY USEFUL COMMANDS TO LEARN

You've seen: **bash, vim/emacs** [pick one], **cat, ls, chdir, mkdir, rm, rmdir, more, find, tr, sort, uniq, cron, rlogin, c++, which, sudo**

apt and apt-get are used to install packages. Many "missing" things just need to be installed. For example this sequence:

```
ken@compute30% sudo apt-get update          // updates everything
ken@compute30% sudo apt-get upgrade         // adds optional features
ken@compute30% sudo apt-get install g++  // installs GNU C++ compiler
```

Ken's .bashrc file set the prompt to the machine he is on, followed by "% "

SOME REALLY USEFUL COMMANDS TO LEARN

ps: Used to see what processes are running

who: Used to see if other people are on this same machine

top: Used to see the "heavy hitters" among active processes

apt/apt-get: Used to install packages like the GNU C++ compiler, Python, Java, Eclipse

tr and sed: two "editors" controlled by command-line options

tar: Makes a single big file from a list of files or a directory

gzip: Compresses a big file

SOME REALLY USEFUL COMMANDS TO LEARN

C++: Alias for the compiler you wish to use for C++ programs

gdb: Debugger used with C++ programs. Requires c++ -g

time: Measures how long something takes to run.

- It breaks it down: wall-clock time ("real"), time spent running processes ("user") and time spent in the Linux kernel ("sys")

gprof: Fancy tool to understand where your code was spending time. Requires a special c++ command-line argument.

```
g++ -std=c++20  -O3 -Wall -Wpedantic -pthread  -o fast-wc fast-wc.cpp
```

ALL OF THESE TAKE ARGUMENTS

- -std=c++20 means “enable C++ 20 features”
- -g means “I’m still debugging”.
- -O3 means “apply heavy optimizations”
- -pg means “I plan to run the gprof profiler”
- -Wxxx means “warn about xxx…” (many options)
- -pthreads means “I’m using C++ threads”
- -o xxx means “name the compiled program xxx”

FOREGROUND/BACKGROUND

In Linux, each command you execute runs as a “process”. All the commands I showed you run in the “foreground”.

A process will have some source of input (stdin), output (stdout) and some place for error messages (stderr).

We say that a process is in the foreground if console input is currently controlled by that process. A background process can run, but will pause if it reads console input.

HOW TO RUN A BACKGROUND PROCESS

In bash, just give the command line but put a single & at the end.

(Note: double &, as in &&, means something else).

Another option: run a command, then use “^Z” and say “bg”. Bash will freeze the command (^Z), then restart it in the background.

^C VERSUS ^Z

^C kills the foreground process. There is also a command, “kill” to terminate a background process, e.g.: kill %1”

^Z “freezes” a process. It halts but is restartable. To restart it, type the process “number” (%1) or “bg” or “fg”.

- bg puts it in the background. You can run other commands.
- fg puts in the foreground. It is connected to the console.
- “jobs” command lists things you’ve put in the background

^S, ^Q, ^O

^S pauses the screen display of output, but not the process.

^Q resumes the screen output.

^O redirects console output to a black hole (/dev/null). ^O is a toggle: typing ^O again restores console output.

ESC, ^D

Many editors use the “ESC” character to mean “drop out of visual editing mode into command mode”

In vim, “:” lets you do this for a single command.

^D is used to say “no more input”.   Applications that read console input will see an “end of file”

FILE NAME EXTENSIONS

In Windows and Mac systems, we get used to the idea that files have types like “powerpoint” (name.pptx), PDF (name.pdf), image (name.jpg or name.jpeg).

In Linux, file name extensions are optional, but some are common, like name.cpp, name.h or name.hpp, etc.

You can rename a file: Many people rename a.out (default executable name) with something sensible like “myWordCount”

PIPES AND REDIRECTION

If we write

- find . | wc

This means “find all files in this directory and its children and list file names. Here we piped the output into the “wc” command, which will counts lines (the number of files!) and characters.

- find . > file_list

means “create a file called file_list containing the output”. If you use >> it means “append the output to the end of the file”.

HEAD, MORE, TAIL

The “head” command shows just the first lines of a file, or of the input received via a pipe.

More shows one page of its input at a time. Type “q” to quit.

Tail is like head, but shows the end of the file.

EXAMPLE

Ken often compiles programs this way:

```
c++ -std=c++20 myprogram.cpp |& more
```

|& means "pipe output, <u>including any error reports</u>". With |, error messages go to the console (not to the target of the pipe)

"more" "pauses" after each full page of error message. **Ken usually fixes the very first error before looking at the others.**

PIPES AND REDIRECTION

You can also send the contents of a file into a program:

- more < file_list

Shows the data in file_list one page at a time

- find . > file_list &

Runs that same find command "in the background"

- fg

Pulls it back into the "foreground" (and waits for it)

DO YOU REMEMBER THE TIMED WORD-COUNT RACE FROM LECTURE 1?

Bash has a built-in timing capability:

```
time command
```

But of course printing our sorted list of counts would be the main time spent. So I used

```
time command > /dev/null
```

This timed the command but "threw away" the actual output!

SHELL SCRIPTS

You can take it to the next level by creating a file with bash commands and then setting the execute permission bit for it.

Now if you “run” that file name, it runs the script of commands!

Bash supports variables, loops, conditional tests, simple math, string manipulations. You can even pipe program output into a bash variable. Very flexible and useful!

CMAKE SCRIPTS

Similar to bash scripts, but controlled by a “makefile” (and you have to actually run cmake as a command). Again, many fancy options

A basic makefile has the form

```
something:    files it depe
              command(s) to “
```

Cmake will rebuild “iconwriter” if the .cpp or .hpp file has changed

Example

```
iconwriter:   iconwriter.cpp iconwriter.hpp
              g++ -O3 iconwriter.cpp –o iconwriter
```

SUMMARY

Our class is working with C++ on Linux, so we need to become familiar with Linux. Linux ≅ kernel + device drivers + daemons + standard programs like initd and bash

Today we reviewed some Linux concepts and tools as seen by the bash user who might be creating a C++ application.

In future lectures we will see some of the Linux system calls, that a program (in any language) can use to talk directly to the kernel.