

# Processes & User Accounts

*"Nice processes finish last"*

CIS 68C1

UNIX System Administration

# Process Components

- What is a Process?
  - ✗ A process is an **instantiation** of a program
  - ✗ Kernel data structures used to manage RAM, CPU usage, I/O, process status, etc.
  - ✗ Processes have the following attributes:
    - ✗ **PID** – the process ID
      - ✗ Unique, recyclable integer process identifier
    - ✗ **UID** – User ID
      - ✗ Determines the owner of the process
    - ✗ **GID** – Group ID
      - ✗ Determines the group in which the process belongs

Updated Oct 7, 2002

CIS 68C1 UNIX System Administration  
Copyright 2002 - Mike Cappella

2

# Process Components

- Processes attributes (*continued*)
  - ✗ **EUID** – Effective User ID
    - ✗ Determines user access permissions (for setuid programs)
  - ✗ **EGUID** – Effective Group ID
    - ✗ Determines group access permissions (for setgid programs)
  - ✗ **Nice Value**
    - ✗ Influences how often kernel gives process a chance to run
    - ✗ Nice processes run less often
  - ✗ **Controlling Terminal**
    - ✗ Defines device for STDIN, STDOUT, STDERR

Updated Oct 7, 2002

CIS 68C1 UNIX System Administration  
Copyright 2002 - Mike Cappella

3

# Process Life Cycle

- Process Birth
  - ✗ First, a process clones itself using the **fork()** system call
  - ✗ Next the cloned process overlays itself with a new program using the **exec()** system call
  - ✗ All processes are created using **fork / exec**
    - ✗ Except init, and some kernel pseudo-processes, which are created by the kernel itself during bootstrap

Updated Oct 7, 2002

CIS 68C1 UNIX System Administration  
Copyright 2002 - Mike Cappella

4

## Process Life Cycle

- Process Life
  - ✗ The cloning process is the **parent process**
  - ✗ The cloned process is the **child process**
  - ✗ Parent waits for child process to die
    - ✗ Using **wait()** system call
  - ✗ Child process runs independently until completion

## Process Life Cycle

- Process Death
  - ✗ Completed process
    - ✗ Exits with an **exit code** indicating reason for exiting
    - ✗ Is becomes a **zombie**
  - ✗ Parent process receives child's exit code
    - ✗ Its earlier call to **wait()** returns
    - ✗ Kernel cleans up completed process
      - ✗ No longer a zombie
  - ✗ If parent dies before child, or does not call **wait()**
    - ✗ **init** waits for any child process whose parent did not wait

## Signals

- Signals are process-level interrupt requests
  - ✗ Small messages delivered to a process by the kernel
  - ✗ There are more than 30 different signals
  - ✗ Cause processes to take some action
  - ✗ Processes start with default **signal handlers**
  - ✗ A process may override a handler, installing its own
  - ✗ The signal-specific handler is run when the kernel delivers the signal to the process

## Signals

- A process can request that a signal be...
  - ✗ **Caught**
    - ✗ Delivered to the process' handler for the particular signal
  - ✗ **Blocked**
    - ✗ Not delivered until the process unblocks the signal
  - ✗ **Ignored**
    - ✗ Not delivered to process at all – the signal is discarded
- Each signal has its own default action
  - ✗ Terminates the process
  - ✗ Stops the process
  - ✗ Is ignored (discarded)

## Signals

- Two signals, KILL and STOP, can never be caught, blocked, or ignored
  - ✗ KILL (signal 9) causes kernel to terminate process

## Signals

- Some common signals
  - ✗ STOP
    - ✗ Kernel stops the process from running
    - ✗ Cannot be caught, blocked, or ignored – never delivered
  - ✗ KILL
    - ✗ Kernel terminates the process
    - ✗ Cannot be caught, blocked, or ignored – never delivered
  - ✗ INT
    - ✗ Interrupt – sent by terminal driver in response to ^C
    - ✗ Programs catch this signal if they need to clean up before quit, or may just ignore it

## Signals

- Some common signals (*continued*)
  - ✗ TERM
    - ✗ Request to terminate – like INT, process should clean up and quit
  - ✗ QUIT
    - ✗ Similar to TERM, but also causes core (memory image) dump
    - ✗ For debugging a program
  - ✗ HUP
    - ✗ Hang-up – sent by terminal driver (e.g. when connection is lost)
    - ✗ Also caught by some daemons, used as a request to restart
- Learn differences between the similar sounding KILL, INT, TERM, QUIT, HUP

## Signals

- Signals are sent using the **kill** command
- Synopsis
  - ✗ `kill [ -signal ] pid [...]`
  - ✗ *signal* will default to TERM if not specified
- Examples:
  - ✗ `kill -9 1045`
    - ✗ Forces process with PID 1045 to be killed
  - ✗ `kill -USR1 957 902`
    - ✗ Sends signal USR1 to process 957 and process 902
  - ✗ `kill 568`
    - ✗ Sends TERM signal to process with PID 568

## Process States

- A process is always in one of four states
  - ✗ **Runnable**
    - ✗ The process is ready to run
  - ✗ **Sleeping**
    - ✗ The process is waiting for an event to occur
  - ✗ **Stopped**
    - ✗ Process cannot run, until receipt of CONT signal
  - ✗ **Zombie**
    - ✗ Process is dead, awaiting clean up

## Process Niceness

- Process Nice Value (or niceness)
  - ✗ Influences time process receives on CPU
  - ✗ High nice value = less CPU time
  - ✗ Low nice value = more CPU time
  - ✗ The actual nice values vary across UNIX systems
    - ✗ RedHat: -20 to 20
    - ✗ Some others: 0 to 39
  - ✗ Processes inherit their nice values from their parents
  - ✗ Nice value set with **nice** / **renice** commands
    - ✗ The **nice** command is built into some shells

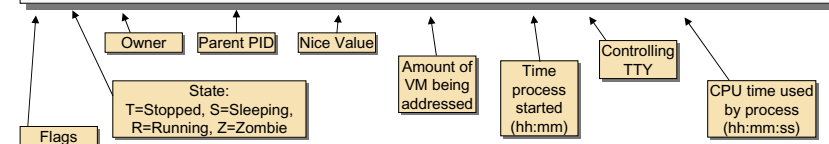
## Process Examination

- The ps Command
  - ✗ Utility to get information about processes on the system
  - ✗ Two forms of **ps** use different arguments
    - ✗ BSD-based
      - ✗ **ps -aux** lists all processes
    - ✗ System V
      - ✗ **ps -elf** lists all processes
- The top Command
  - ✗ Shows real time, periodically updated process information

## Process Examination

- Example ps Output

```
% ps -elf
 F S UID          PID  PPID  C PRI  NI ADDR      SZ  WCHAN  STIME TTY          TIME CMD
100 S root           1    0  0  68  0   -    331 13d2a6 09:32 ?          00:00:03 init
040 S root           2    1  0  69  0   -     0 11f7fd 09:32 ?          00:00:00 [keventd]
040 S root           3    1  0  79 19   -     0 11821e 09:32 ?          00:00:00 [ksoftirqd_CPU0]
040 S root           4    1  0  69  0   -     0 129da8 09:32 ?          00:00:00 [kswapd]
040 S root           5    1  0  69  0   -     0 133f5a 09:32 ?          00:00:00 [bdflush]
040 S root           6    1  0  69  0   -     0 133ff5 09:32 ?          00:00:00 [kupdated]
040 S root           7    1  0  69  0   -     0 1c6a20 09:32 ?          00:00:00 [khubd]
040 S root          124   1  0  69  0   -     0 8a8c7a 09:32 ?          00:00:00 [kjournald]
040 S root          125   1  0  69  0   -     0 8a8c7a 09:32 ?          00:00:00 [kjournald]
040 S root          126   1  0  69  0   -     0 8a8c7a 09:32 ?          00:00:00 [kjournald]
040 S root          127   1  0  69  0   -     0 8a8c7a 09:32 ?          00:00:00 [kjournald]
040 S root          537   1  0  69  0   -    341 13d2a6 09:34 ?          00:00:00 syslogd -m 0
140 S root          542   1  0  69  0   -     0 325 113e11 09:34 ?          00:00:00 klogd -x
000 T cappella    1583 1150  0  69  0   -    1268 10865b 10:01 tty1    00:00:00 vim -X rpmupdate
000 R cappella    1921 1890  0  78  0   -     636   - 12:48 pts/0  00:00:00 ps -elf
```



## Process Examination

### □ Example top Output

```
1:25pm up 3:52, 2 users, load average: 0.00, 0.04, 0.01
88 processes: 85 sleeping, 2 running, 0 zombie, 1 stopped
CPU states: 21.7% user, 2.9% system, 0.0% nice, 75.2% idle
Mem: 516348K av, 428668K used, 87680K free, OK shrd, 108988K buff
Swap: 1044184K av, 2268K used, 1041916K free, 65256K cached

PID USER PRI NI SIZE RSS SHARE STAT %CPU %MEM TIME COMMAND
2020 cappella 11 0 1028 1028 792 R 2.9 0.1 0:00 top
1 root 8 0 488 436 416 S 0.0 0.0 0:03 init
2 root 9 0 0 0 0 SW 0.0 0.0 0:00 keventd
3 root 19 19 0 0 0 SWN 0.0 0.0 0:00 ksoftirqd_CPU0
4 root 9 0 0 0 0 SW 0.0 0.0 0:00 kswapd
5 root 9 0 0 0 0 SW 0.0 0.0 0:00 bdflush
6 root 9 0 0 0 0 SW 0.0 0.0 0:00 kupdated
7 root 9 0 0 0 0 SW 0.0 0.0 0:00 khubd
127 root 9 0 0 0 0 SW 0.0 0.0 0:00 kjournald
537 root 9 0 548 544 460 S 0.0 0.1 0:00 syslogd
542 root 9 0 448 440 388 S 0.0 0.0 0:00 klogd
630 root 9 0 0 0 0 SW 0.0 0.0 0:00 nfsd
632 root 9 0 0 0 0 SW 0.0 0.0 0:00 nfsd
631 root 9 0 0 0 0 SW 0.0 0.0 0:00 lockd
633 root 9 0 0 0 0 SW 0.0 0.0 0:00 rpciod
775 root 9 0 972 928 804 S 0.0 0.1 0:00 sshd
808 root 9 0 968 924 764 S 0.0 0.1 0:00 xinetd
841 root 9 0 1116 1116 928 S 0.0 0.2 0:00 safe_mysqld
898 root 9 0 1648 1320 1124 S 0.0 0.2 0:00 sendmail
917 root 9 0 464 456 404 S 0.0 0.0 0:00 gpm
```

## Process Runaways

- Resource hog processes can cause trouble
  - ✗ Excessive CPU time, disk usage, etc.
- Examine process information with **ps**
  - ✗ Look at CPU usage, size, nice value, etc.
- Learn more about process before you kill it
  - ✗ Talk to person who started process if possible
- Consider stopping it with STOP signal
- Kill with -9 when all else fails

## User Accounts

- `/etc/passwd`
  - ✗ Users are defined in the `/etc/passwd` file
  - ✗ It contains basic user account information
  - ✗ Each line is a single user account
  - ✗ There are 7 colon-separated fields per line (user)
    - ✗ Login name
    - ✗ Encrypted password
    - ✗ UID
    - ✗ GID
    - ✗ GECOS information
    - ✗ Home directory
    - ✗ Login shell

## User Accounts

- Login Name
  - ✗ Used to log into system
  - ✗ Should limit to 8, lower-case, alpha-numeric characters
  - ✗ Should create standardized naming scheme
  - ✗ Should be unique across all systems
  - ✗ User should have same login name on all systems
- Encrypted password
  - ✗ Set to `*` to disable account or when creating account
  - ✗ Set to `x` to use `/etc/shadow` password file

## User Accounts

- UID
  - × Numeric ID representing a User
  - × Newer systems have 32-bit UID value
    - × May need to limit values to < 32,767 or 65,535 for compatibility with older systems
  - × UID 0 is always privileged user, or root account
  - × UIDs should be:
    - × Unique per system
    - × Consistent across all systems when using NFS

## User Accounts

- GID
  - × Numeric ID, similar to UID
  - × Default group for user at login
  - × Groups defined in **/etc/group**
  - × User can be in several groups
    - × Users can switch their current group with **newgrp** command
  - × GIDs should be consistent across systems when using NFS

## User Accounts

- GECOS Field
  - × Originally designed for institutional use to contain office, phone, and department information
  - × Format not well-defined
  - × Can be used to record personal information
  - × Typically used now for user's full name
  - × The **finger** utility uses this information
  - × The **chfn** command allows user to change data
    - × Many administrators disable **chfn**

## User Accounts

- Home Directory
  - × Current working directory upon login
  - × Users should own their own home directories
    - × Set the owner ID of the directory to user's UID
    - × Likewise for the group ID and GID
  - × Home directories are often mounted over NFS
    - × Network problems may cause unavailability
- Login Shell
  - × The shell created for a user's login session
  - × The **chsh** command lets users change their login shell
    - × **/etc/shells** is the list of valid shells that **chsh** allows
    - × Many administrators disable **chsh**

## User Accounts

- /etc/shadow
  - ✗ Used to store read-protected encrypted passwords
    - ✗ Prevents users from reading encrypted passwords in /etc/passwd, and trying brute-force cracking techniques
  - ✗ For security, /etc/shadow must be readable only by root
  - ✗ Contains one line per user
  - ✗ Each line is 9 colon-separated fields
    - ✗ Field 1: username, field 2: encrypted password
  - ✗ The login name and password fields cannot be empty
    - ✗ Login name is same as that in /etc/passwd
    - ✗ Password is created by running **passwd** command

## User Accounts

- /etc/group
  - ✗ Contains names groups and their members
  - ✗ Each line is 4 colon-separated fields
    - ✗ Group name
      - ✗ The name of the group
    - ✗ Encrypted password
      - ✗ Rarely used – just set to “x”
    - ✗ GID number
      - ✗ The ID number of the group
    - ✗ List of members (comma separated)
      - ✗ The users who belong to the group

## User Accounts

- Adding New Users
  - ✗ Add line for new user into /etc/passwd and /etc/shadow, filling in appropriate fields
  - ✗ Set initial password using **passwd** command
  - ✗ Create users home directory
  - ✗ Copy default startup files from /etc/skel
  - ✗ Add user to /etc/group file
  - ✗ Change permissions and ownership of user’s files and home directory
  - ✗ Login to the account, and verify that everything is correct

## User Accounts

- Options for Disabling User Accounts
  - ✗ Remove account
  - ✗ Change shell to a custom program that outputs a message indicating that the account is disabled, then exits
    - ✗ Program should not be listed in /etc/shells
  - ✗ Place a \* in the password field of /etc/passwd
    - ✗ No password will ever encrypt to \*; thus a user cannot login