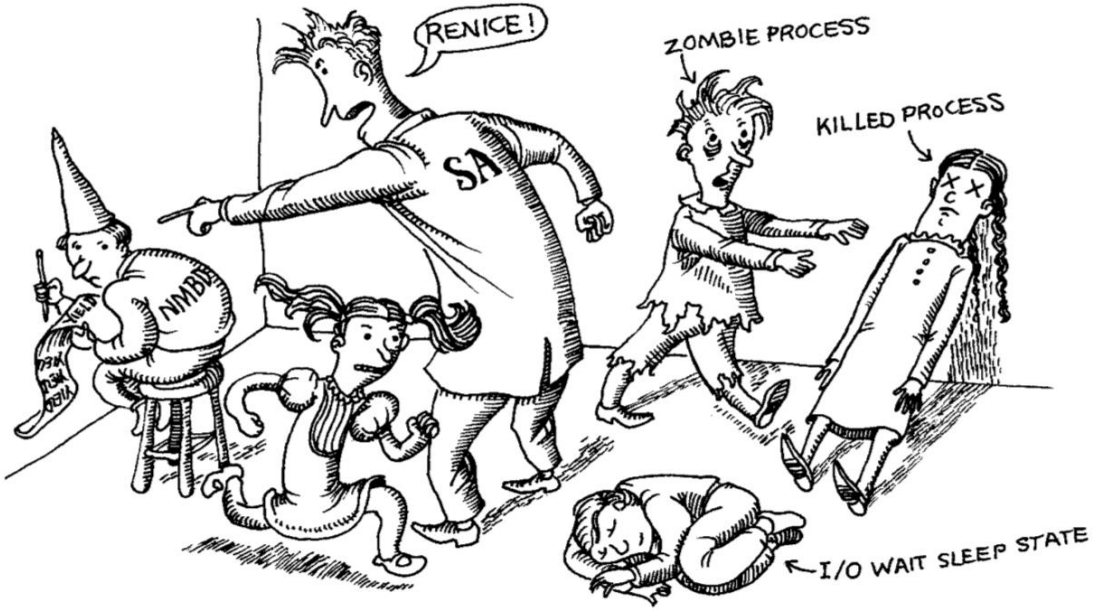
Process Control



What is a Process?

- A process represents a running program. It’s the abstraction through which memory, processor time, and I/O resources can be managed and monitored
- UNIX philosophy states that as much work as possible should be done with processes rather than being handled specially by the kernel
- System and user processes follow the same rules, so you can use a single set of tools to control them

Process Bookkeeping

- The kernel tracks several pieces of information for each process:
  - Address Space Map
  - Current Status
  - Execution Priority
  - Resource Usage
  - Files and Network Ports used
  - Signal Mask (which signals are blocked)
  - Owner
- As well as lots of other pieces of information

Processes vs. Threads

- Processes represent programs which the kernel coordinates
- Process has memory allocated to it by the kernel which contains: code, libraries, variables, process stack, and bookkeeping information

- Threads are an execution context within a process
- There is at least one thread associated with every process, but there can be more (multithreaded programs)
- Threads have their own stack and CPU context, but use the processes memory

Important Process Information

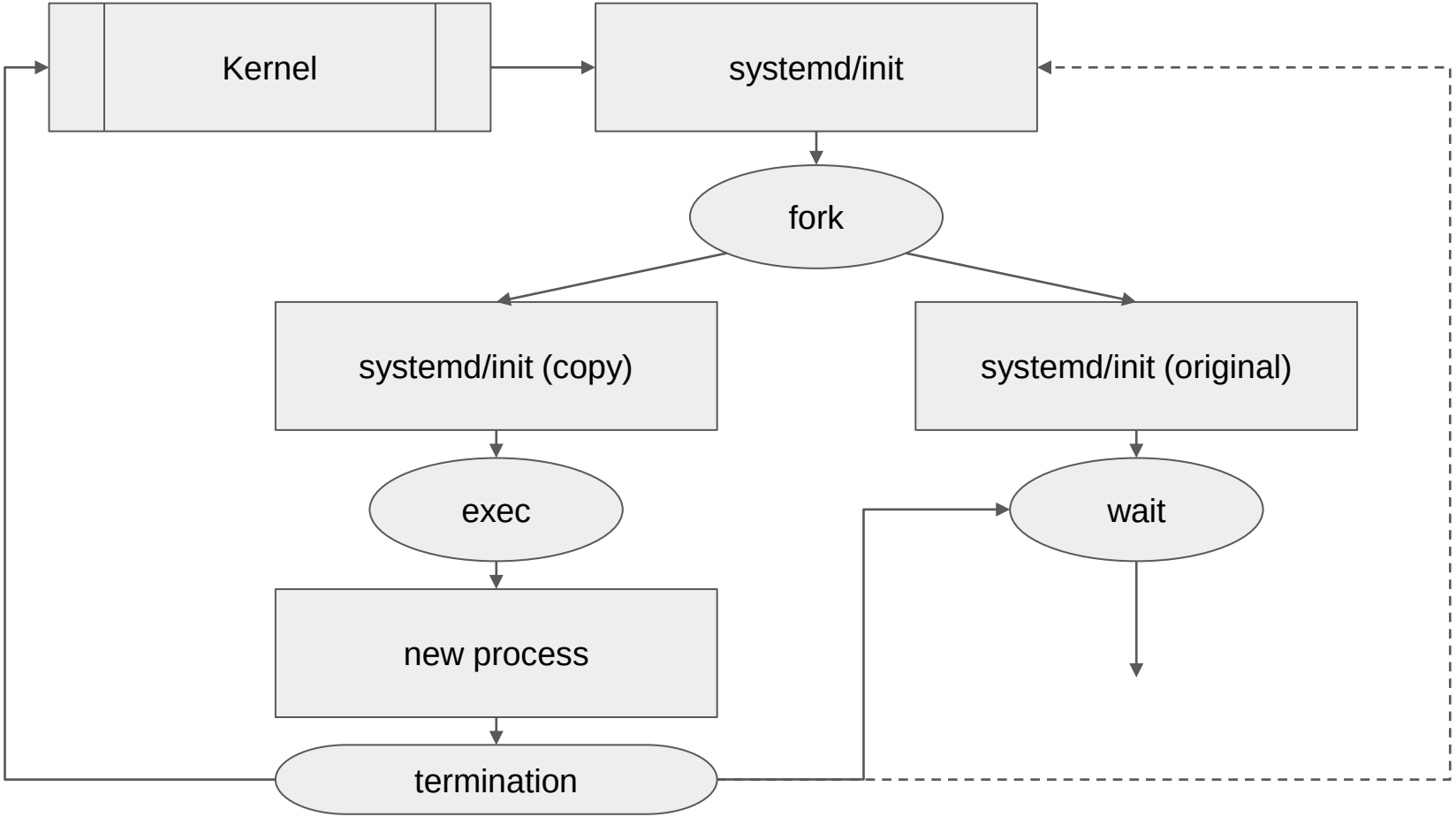
- PID: Process ID Number
  - Unique ID number assigned to the process by the kernel
  - Most commands require PID to specify which process to effect
  - Low PIDs are typically special system processes (PID1 is systemd/init)
- PPID: Parent PID
  - There is no method to create new processes directly in UNIX (systemd/init is a special case)
  - Instead, all processes are forks of another process with their execution replacing the parents
  - PPID allows you to track where a process spawned from

Important Process Information

- UID and EUID: real and effective user ID
  - Process UID is the user identification number of the user who created it
  - In reality it is a copy of the UID value of the parent process (since that is how it is created)
  - EUID is the “effective” UID and determines what resources and files a process can access
  - EUID describes permissions rather than identity, allowing processes to gain and lose privileges through the use of setuid
- Control Terminal
  - Displays which terminal the process is connected to for standard input, output, and error printing

Important Process Information

- GID and EGID: real and effective group ID
  - Functions the same as UID and EUID but for group identity and permissions
  - Since users can belong to multiple groups, access permissions are determined using the EGID and a supplemental group list
- Niceness
  - Measurement of priority for a process which is combined with how much CPU time the process has used and how long its been waiting to determine what should be executed when
  - Low niceness value means it has high priority (it doesn’t plan on being very nice to other processes)


Kernel
systemd/init
fork
systemd/init (copy)
systemd/init (original)
exec
wait
new process
termination

Process Generation

What are three ways (that you've learned so far) that can cause processes to be spawned and executed?

Periodic Processes

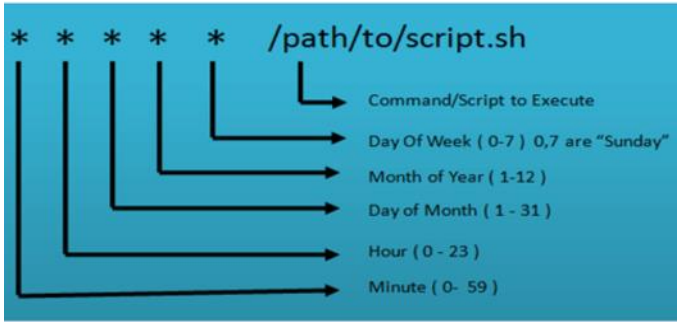
- There are 3 primary ways that we have learned processes can be executed
  - Executed by systemd/init during boot
  - Executed as a subprocess of a boot process
  - Executed explicitly by a user
- However, there are two additional ways that processes can spawn which may be less obvious
  - The cron (crond on CentOS) process scheduling system
  - Systemd timers for scheduling processes
- These systems are used to schedule tasks that happen at regular intervals such as sending mail reports, cleaning filesystems, rotating log files, processing batch jobs, or backing up the system

cron and crontab

- Traditional tool for running commands on a predetermined schedule in Linux
- Uses a configuration file called a crontab (cron table) to specify what command should be run when
- There is one crontab per user, located at `/var/spool/cron` as well as system level crontab files at `/etc/crontab` (for user administrator use) and `/etc/cron.d` (for software package use)
- Cron works silently, but most versions can keep a log file (usually located at `/var/log/cron`)
- `crontab` informs `cron` about new configurations and allows you to edit and view crontab files
- As root, you can specify a filename or username to most `crontab` commands to view and edit other users cron tasks

Crontab syntax

- Examples:
  - 30 08 10 06 * /home/maverick/full-backup
    - 10th of June 8:30 AM
  - */1 * * * * /[PATH-TO-SOMETHING]/script.sh
    - Every minute
- Use online generators if understanding syntax is hard in the beginning:
  - https://crontab.guru/every-2-minutes



Tracking, Diagnosing, and Modifying Processes

Signals

- Signals can be used in a variety of different ways
  - Communication between processes
  - Sent by the terminal to kill, interrupt, or suspend a process
  - Sent directly to a process using `kill`
  - Sent by the kernel because of an infraction (segfault)
  - Sent by the kernel for coordination (death of a child process, data on an I/O channel)

Important Signals

Signals every administrator should know[a]

#[b]	Name	Description	Default	Can catch?	Can block?	Dump core?
1	HUP	Hangup	Terminate	Yes	Yes	No
2	INT	Interrupt	Terminate	Yes	Yes	No
3	QUIT	Quit	Terminate	Yes	Yes	Yes
9	KILL	Kill	Terminate	No	No	No
10	BUS	Bus error	Terminate	Yes	Yes	Yes
11	SEGV	Segmentation fault	Terminate	Yes	Yes	Yes
15	TERM	Software termination	Terminate	Yes	Yes	No
17	STOP	Stop	Stop	No	No	No
18	TSTP	Keyboard stop	Stop	Yes	Yes	No
19	CONT	Continue after stop	Ignore	Yes	No	No
28	WINCH	Window changed	Ignore	Yes	Yes	No
30	USR1	User-defined #1	Terminate	Yes	Yes	No
31	USR2	User-defined #2	Terminate	Yes	Yes	No

a. A list of signal names and numbers is also available from the **bash** built-in command **kill -l**.
b. May vary on some systems. See **/usr/include/signal.h** or **man signal** for more information.

kill Command

- Three versions, all used to send signals (TERM by default) to processes
  - `kill [-signal] pid` sends the specified signal to the process specified by the given pid
  - `killall [-signal] pname` sends the specified signal to all processes matching the process name pname
  - `pkill [-signal] [-P ppid] [-u euid] [-U uuid]` sends the specified signal to all processes matching the criteria, and allows you to specify a large set of search criteria

ps Command

- Allows you to view process information, exact commands and how they are displayed vary by distribution
- Primary commands are `ps aux` (user friendly print) and `ps lax` (more technical)
- Pipe the output of `ps` to `grep` in order to search for particular processes
  - `ps aux | grep sshd`

Explanation of ps aux output

Field	Contents
USER	Username of the process's owner
PID	Process ID
%CPU	Percentage of the CPU this process is using
%MEM	Percentage of real memory this process is using
VSZ	Virtual size of the process
RSS	Resident set size (number of pages in memory)
TTY	Control terminal ID
STAT	Current process status: R = Runnable D = In uninterruptible sleep S = Sleeping (< 20 sec) T = Traced or stopped Z = Zombie Additional flags: W = Process is swapped out < = Process has higher than normal priority N = Process has lower than normal priority L = Some pages are locked in core s = Process is a session leader
TIME	CPU time the process has consumed
COMMAND	Command name and arguments [a]

a. Programs can modify this information, so it's not necessarily an accurate representation of the actual command line.

ps aux

```
redhat$ ps aux
  USER   PID %CPU %MEM    VSZ   RSS   TTY STAT  TIME  COMMAND
  root     1  0.1   0.2   3356   560   ?   S     0:00  init [5]
  root     2    0     0      0     0   ?   SN    0:00  [ksoftirqd/0]
  root     3    0     0      0     0   ?   S<    0:00  [events/0]
  root     4    0     0      0     0   ?   S<    0:00  [khelper]
  root     5    0     0      0     0   ?   S<    0:00  [kacpid]
  root    18    0     0      0     0   ?   S<    0:00  [kblockd/0]
  root    28    0     0      0     0   ?   S     0:00  [pdflush]
...
  root   196    0     0      0     0   ?   S     0:00  [kjournald]
  root  1050    0   0.1   2652   448   ?   S<s   0:00  udevd
  root  1472    0   0.3   3048  1008   ?   S<s   0:00  /sbin/dhclient -1
  root  1646    0   0.3   3012  1012   ?   S<s   0:00  /sbin/dhclient -1
  root  1733    0     0      0     0   ?   S     0:00  [kjournald]
  root  2124    0   0.3   3004  1008   ?   Ss    0:00  /sbin/dhclient -1
  root  2182    0   0.2   2264   596   ?   Ss    0:00  rsyslog -m 0
  root  2186    0   0.1   2952   484   ?   Ss    0:00  klogd -x
  root  2519  0.0   0.0  17036   380   ?   Ss    0:00  /usr/sbin/atd
  root  2384    0   0.6   4080  1660   ?   Ss    0:00  /usr/sbin/sshd
  root  2419    0   1.1   7776  3004   ?   Ss    0:00  sendmail: accept
...
```

ps lax

```
redhat$ ps lax
  F  UID   PID  PPID  PRI   NI    VSZ   RSS  WCHAN  STAT  TIME  COMMAND
  4    0     1     0   16    0   3356   560  select  S     0:00  init [5]
  1    0     2     1   34   19      0     0  ksofti  SN    0:00  [ksoftirqd/0
  1    0     3     1    5  -10      0     0  worker  S<    0:00  [events/0]
  1    0     4     3    5  -10      0     0  worker  S<    0:00  [khelper]
  5    0  2186     1   16    0   2952   484  syslog  Ss    0:00  klogd -x
  5   32  2207     1   15    0   2824   580  -       Ss    0:00  portmap
  5   29  2227     1   18    0   2100   760  select  Ss    0:00  rpc.statd
  1    0  2260     1   16    0   5668  1084  -       Ss    0:00  rpc.idmapd
  1    0  2336     1   21    0   3268   556  select  Ss    0:00  acpid
  5    0  2384     1   17    0   4080  1660  select  Ss    0:00  sshd
  1    0  2399     1   15    0   2780   828  select  Ss    0:00  xinetd -sta
  5    0  2419     1   16    0   7776  3004  select  Ss    0:00  sendmail: a
...
```

top Command and nice/renice

- While `ps` gives you a snapshot of the process information, `top` gives you an interactive display which updates every 1-2 seconds (configurable) and displays both per-process information and overall system usage
- This can be very useful for finding runaway processes (as can `ps` if they are non-transient) but you should be cautious as runaway processes and processes under heavy load can look similar in `ps` and `top`
- `top` also allows you to view utilization on average and for each specific core, as well as `renice` processes to change their priority and track the results
- `renice (+/-) X` allows you to increase or decrease the priority (niceness) of a process by a value X (-20 to 19 range). This is rarely done by hand, but might be useful for services which are expected to respond quickly

```
redhat$ top
top - 20:07:43 up  1:59,  3 users,  load average: 0.45, 0.16, 0.09
Tasks: 251 total,   1 running, 250 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.7 us, 1.2 sy, 0.0 ni, 98.0 id, 0.0 wa, 0.0 hi, 0.2 si, 0.0 st
KiB Mem :  1013672 total,  128304 free,  547176 used,  338192 buff/cache
KiB Swap:  2097148 total, 2089188 free,    7960 used.  242556 avail Mem

 PID  USER   PR  NI    VIRT   RES   SHR S %CPU %MEM    TIME+  COMMAND

 1647  root   20   0  177436  3656  2632 S  0.3  0.4  0:04.47  cupsd
10246  ulsah  20   0  130156  1936  1256 R  0.3  0.2  0:00.10  top
    1  root   20   0   59620  5472  3348 S  0.0  0.5  0:02.09  systemd
    2  root   20   0       0     0     0 S  0.0  0.0  0:00.02  kthreadd
    3  root   20   0       0     0     0 S  0.0  0.0  0:00.03  ksoftirqd/0
    5  root    0 -20       0     0     0 S  0.0  0.0  0:00.00  kworker/0:+
    7  root   rt   0       0     0     0 S  0.0  0.0  0:00.20  migration/0
    8  root   20   0       0     0     0 S  0.0  0.0  0:00.00  rcu_bh
    9  root   20   0       0     0     0 S  0.0  0.0  0:00.00  rcuob/0
...
```

strace

- Tool which allows you to view the execution of a process, usually used to diagnose root-causes of misbehaving processes
- Optional package for Linux, so may need to be installed before use
- Allows you to attach to a process, shows all the processes system calls, and then detach from the process “without disturbing it”
- System calls are low level, but strace shows parameters used in the system calls as well as the kernels response
- Look for error codes (which strace will often decode for you) first

Questions?

Additional Resources

TLDP Chapter 4: Processes