Southern Connecticut State University

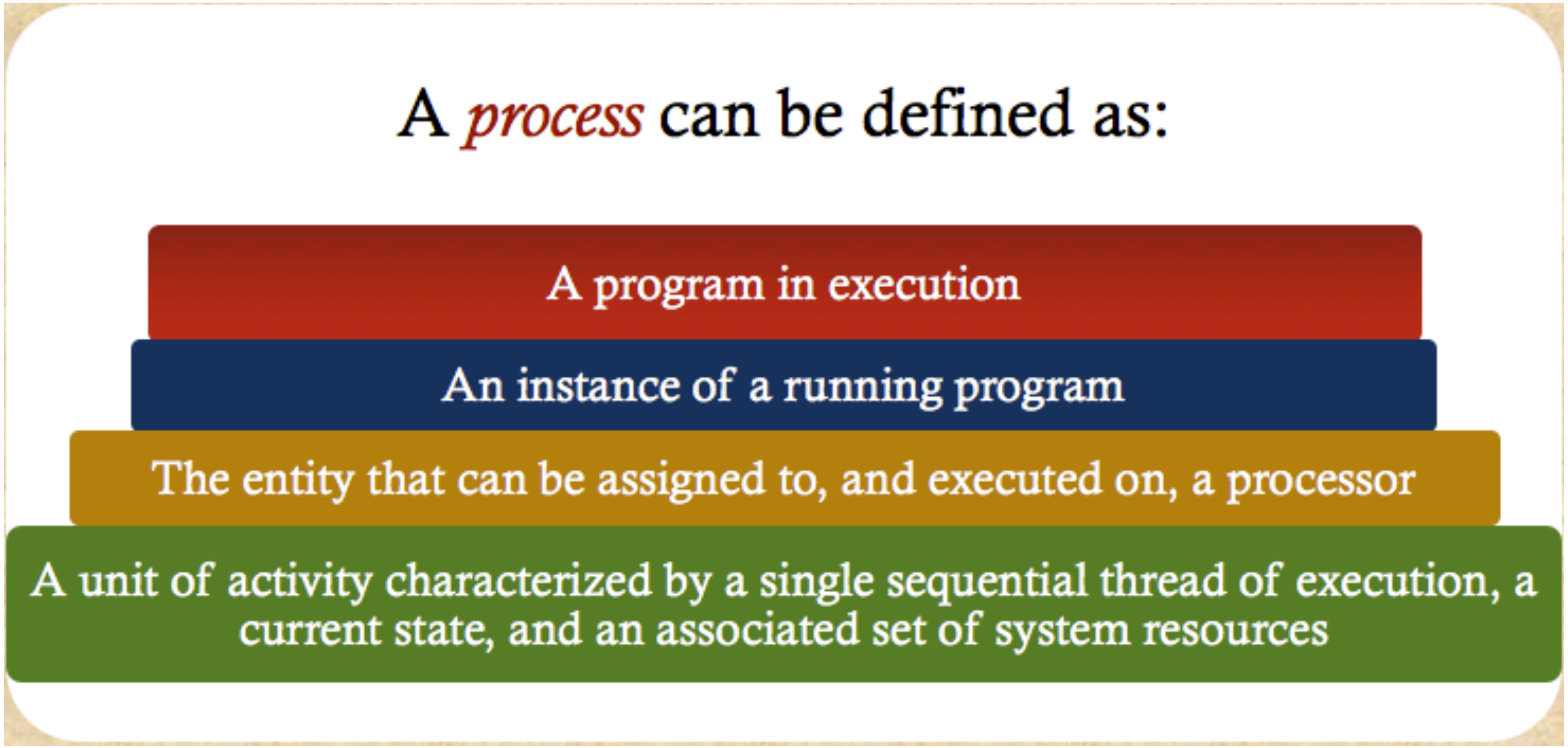
CSC424 System Administration

Instructor: Dr. Hao Wu

Week 5 User Management, Access Control and Rootly Powers

Process

- Fundamental to the structure of operating system



A *process* can be defined as:

A program in execution

An instance of a running program

The entity that can be assigned to, and executed on, a processor

A unit of activity characterized by a single sequential thread of execution, a current state, and an associated set of system resources

Components of a process

- An address space
  - a set of memory pages that the kernel has marked for the process's use
  - these pages contain the code and libraries, process's variable, its stacks, and various extra information needed by the kernel while the process is running
- A set of data structures within the kernel
  - The process's address space map
  - The current status of the process
  - The execution priority of the process
  - Information about the resources the process has used
  - Information about the files and network ports the process has opened
  - The process's signal mask
  - The owner of the process

Thread

- A thread is an execution context within a process
  - Every process has at least one thread
  - Some processes have many thread
  - Each thread has its own stack and CPU context but operates within the address space of its enclosing process
  - A process's threads can run simultaneously on different cores

Process information

- PID: process ID number
  - The kernel assigns a unique ID number to every process
  - Most commands that manipulate processes require you to specify a PID to identify the target of the operation.
  - PIDs are assigned in order as processes are created
- PPID: parent PID
  - Linux create a new process in two separate steps
  - First, an existing process must clone itself to create a new process
  - The clone then exchange the program it's running for a different one
  - When a process is cloned, the original process is referred to as the parent

Process information

- UID: real user ID
  - The user identification number of the person who created it
  - Usually, only the creator and the superuser can manipulate a process
- EUID: effective user ID
  - an extra UID that determines what resources and files a process has permission to access at any given moment
  - For most processes, the UID and EUID are the same
- UID v.s. EUID:
  - it's useful to maintain a distinction between identity and permissions
- GID and EGID: real and effective group ID

Process information

- Niceness
  - A process's scheduling priority determines how much CPU time it receives
  - The kernel also pays attention to an administratively set value that's usually called the "nice value" or "niceness". It specifies how nice you are planning to be to other users of the system.
- Control terminal
  - Most nondaemon processes have an associated control terminal
  - The control terminal determines the default linkages for the standard input, standard output, and standard error channels
  - It also distributes signals to processes in response to keyboard events such as <Control-C>

The life cycle of a process

- **Fork**: to create a new process, a process copies itself with the fork system call.
- After a fork, the child process often uses one of the **exec** family of routines to begin the execution of a new program.
- These calls change the program that the process is executing and reset the memory segments to a predefined initial state
- When the system boots, the kernel autonomously creates and installs several processes. The most notable of these is **init** or **systemd,** which is always process number 1.
- **init**(or **systemd**) also plays another important role in process management. When a process completes, it calls a routine named _**exit** to notify the kernel that it is ready to die.

Signals

- Signals are process-level interrupt requests. About thirty different kinds are defined, and they're used in a variety of ways:
  - They can be sent among processes as a means of communication
  - They can be sent by the terminal driver to kill, interpt, or suspend processes when keys such as <Control-C> and <Control-Z> are pressed
  - They can be sent by an administrator (with kill) to achieve various ends
  - They can be sent by the kernel when a process commits an infraction such as division by zero
  - They can be sent by the kernel to notify a process of an "interesting" condition such as the heath of a child process or the availability of data on an I/O channel.

Signals every administrator should know

#	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
10	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

Southern Connecticut State University
SC SU

Signals

- The signals KILL, INT, TERM, HUP, and QUIT all sound as if they mean approximately the same thing, but their uses are actually quite different:
  - KILL: is unblock able and terminates a process at the kernel level. A process can never actually receive or handle this signal
  - INIT: is sent by the terminal driver when the user press <Control-C>. It's a request to terminate the current operation. Simple programs should quit or simply allow themselves to be killed, which is the default if the signal is not caught.
  - TERM: is a request to terminate execution completely. It's expected that the receiving process will clean up its state and exit.
  - QUIT: is similar to TERM, except that it defaults to producing a core dump if not caught.

Signals

- HUP: has two common interpretations.
  - First, it's understood as a reset request by many daemons. If a daemon is capable of rereadding its configuration file and adjusting to changes without restarting, a HUP can generally trigger this behavior
  - Second, HUP signals are sometimes generated by the terminal driver in an attempt to "clean up" the processes attached to a particular terminal.
  - **nohup:** makes a process running in background
  - What happened when you execute the following command:

```
nohup yes sir&
```

kill: send signals

- kill: a command most often been used to terminate a process. It can send any signal, but by default it sends a TERM.
- Syntax:
- kill [-signal] pid
- A kill without a signal number does not guarantee that the process will die, because the TERM signal can be caught, blocked, or ignored.
- Command:
- kill -9 pid
- guarantees that the process will die

Send signals

- Command:
- killall pname
- kills processes by name.

- Command:
- pkill -u pname
- searches for processes by name (or other attributes, such as EUID) and sends the specified signal.

ps: monitor processes

- The ps command is the system administrator's main tool for monitoring processes.
- Most commonly used options:
  - a : show all processes
  - x : show even processes that don't have a control terminal
  - u : selects the "user oriented" output format

Example of "ps aux"

```
[root@node1 ~]# ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.6 128164  6824 ?        Ss   19:52   0:01 /usr/lib/systemd/
systemd --s
root           2  0.0  0.0      0     0 ?        S    19:52   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        S    19:52   0:00 [ksoftirqd/0]
root           4  0.4  0.0      0     0 ?        R    19:52   0:20 [kworker/0:0]
root           5  0.0  0.0      0     0 ?        S<   19:52   0:00 [kworker/0:0H]
root           6  0.0  0.0      0     0 ?        S    19:52   0:00 [kworker/u2:0]
root           7  0.0  0.0      0     0 ?        S    19:52   0:00 [migration/0]
root           8  0.0  0.0      0     0 ?        S    19:52   0:00 [rcu_bh]
root           9  0.0  0.0      0     0 ?        R    19:52   0:00 [rcu_sched]
root          10  0.0  0.0      0     0 ?        S    19:52   0:00 [watchdog/0]
root          12  0.0  0.0      0     0 ?        S    19:52   0:00 [kdevtmpfs]
root          13  0.0  0.0      0     0 ?        S<   19:52   0:00 [netns]
root          14  0.0  0.0      0     0 ?        S    19:52   0:00 [khungtaskd]
root          15  0.0  0.0      0     0 ?        S<   19:52   0:00 [writeback]
```

Understand 'ps aux'

- 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
- TIME: CPU time the process has consumed
- COMMAND: Command name and arguments

Understand 'ps aux'

- Process status 'STAT':
  - R = Runnable
  - S = Sleeping(<20 sec)
  - Z = Zombie
  - D = In uninterruptible sleep
  - T = Traced or stopped
  - 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

top: Interactive monitoring

- Command ps show you a snapshot of the system as it was at the time. Often, that limited sample is insufficient to convey the big picture of what's really going on.
- top is a sort of real-time version of ps that gives a regularly updated, interactive summary of process and their resource usage.

```
top - 21:19:12 up  1:26,  2 users,  load average: 0.00, 0.01, 0.05
Tasks: 100 total,   3 running,  97 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.1 us,  0.2 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.1 si,  0.0 st
KiB Mem :  1016232 total,    66320 free,   191644 used,   758268 buff/cache
KiB Swap:   839676 total,   839676 free,        0 used.   586052 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
    1 root      20   0  128164   6824   4060 S  0.0  0.7   0:01.39 systemd
    2 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kthreadd
    3 root      20   0       0      0      0 S  0.0  0.0   0:00.24 ksoftirqd/0
    5 root       0 -20       0      0      0 S  0.0  0.0   0:00.00 kworker/0:0H
    6 root      20   0       0      0      0 S  0.0  0.0   0:00.28 kworker/u2:0
    7 root      rt   0       0      0      0 S  0.0  0.0   0:00.00 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 R  0.0  0.0   0:00.65 rcu_sched
```

cron: schedule commands

- The cron daemon is the traditional tool for running commands on a predetermined schedule.
- It starts when the system boots and runs as long as the system is up.
- cron reads configuration files containing lists of command lines and times at which they are to be invoked.
- A cron configuration file is called a 'crontab', short for 'cron table'.
- Crontabs for individual users are stored under /var/spool/cron

The format of crontab files

- All the crontab files on a system share a similar format. Comments are introduced with a pound sign (#) in the first column of a line. Each non-comment line contains six fields and represents one command:
- *minute hour dom month weekday command*

Field	Description	Range
minute	Minute of the hour	0 to 59
hour	Hour of the day	0 to 23
dom	Day of the month	1 to 31
month	Month of the year	1 to 12
weekday	Day of the week	0 to 6 (0=Sunday)

The format of crontab files

- Each of the time-related fields can contain:
  - A star, which matches everything
  - A single integer, which matches exactly
  - Two integers separated by a dash, matching a range of values
  - A range followed by a slaph and a step value, e.g. 1-10/2
  - A comma-separated list of interiors or ranges, matching any value