Process Control

What is a process?

- A representation of a running program
- Used to monitor and control the program's access to system resources like CPU, memory, and I/O
- The kernel allocates memory for each process, typically measured in "pages". The location of this address space (memory or swap) cannot be determined as UNIX combines both in the so called "virtual memory"
- Important information about the process is recorded by the kernel, including:
  - The owner
  - The state (running, sleeping, …etc)
  - Execution priority (nice value)
  - The resources used (files, network sockets…etc.)

Identification

- The kernel assigns a unique identifier to each process call PID
- When the process demands a new program (process) to be started, it clones itself first, then the clone replaces the initial program with the desired one. The original process here is called *parent* and it's id is referred to as *parent ID* or PPID.
- The PPID can be used to trace the source of a running process. It can also be used to track the behavior of a given process (how many and what processes it is spawning).

Ownership

- The real UID (RUID) is the user id of the process owner.
- The effective UID (EUID) is the id of the user that the process is using *temporarily* to do some tasks. This is often the root id (0). This behavior can be found in programs that have setuid like passwd
- The group id GID is treated the same as the UID, with GID, and EGID.
- The GID is not used most of the time, instead, the EGID and the supplementary group list determine the process access permissions, because a give process may need to use one of the supplementary groups to be granted the required access level.
- The GID may be used, however, when the process accesses the filesystem, to create new files for example.

CPU priority

- The kernel internally determines how much CPU time a process is allowed to consume
- The administrator can set this value manfully. It is called the nice value. It ranges from -20 (highest priority) to 19 (lowest priority)
- By default, the child process inherits its parent's nice value
- A normal user can use the `renice` command to lower the priority of his own process (positive value). But he cannot use it to increase it's priority even to it's original value. The superuser can place any nice value on any process on the system.

  ```
  renice value pid
  ```
- The nice value only controls the CPU time assigned to the process and not the utilization of memory or I/O
- The nice command is used to start a process in the specified nice value.

  ```
  nice –n value /path/to/command
  ```
- The most common use of the nice command is when the system is under high load and you want to start a shell to investigate the issue. Starting a normal shell will cause a lot of lagging. Using, for example, nice –n -10 bash will start a bash shell in a higher priority than other processes

Process spawning and termination

- A running process can launch a new *child* process by using the `fork` system call
- This creates an identical clone of the running process. It has, however, some differences like:
  - A different, unique PID
  - Zero CPU, memory consumption (initially)
  - Empty list of pending signals
- The child process uses the exec command (system call) to execute the new program
- When the child process dies first, the parent process issue the `wait` system call to notify the kernel that the child process is ready to vanish. It also receives the exit code that indicates why and how the termination occurred (0 means normal)
- If the parent process dies first, the child process becomes *orphaned*. In this case, the `init` process becomes the parent.

Communicating with a process

- A running process can receive signals from the kernel, the user, or other processes as a means of communication
- The process may respond to the signal (if programmed to), or the kernel will act on behalf of the process and respond to the signal
- The following are some of the most familiar signals:

Number	Name	Short for	Sent when
1	SIGHUP	Hang-up	The controlling terminal has been closed.
2	SIGINT	Interrupt	Break key is pressed (like CTRL-C)
3	SIGQUIT	Quit	The user wants the process to terminate and provide a core dump file. Sent by CTRL-Y or CTRL-\
9	SIGKILL	Kill	Terminate immediately. This signal cannot be ignored and the process does not perform any clean up before exiting

Process signals (cont.)

Number	Name	Short for	Sent when
11	SEGV	Segmentation fault	The process tries to access virtual memory that does not belong it to. The process is terminated and core dump is provided
15	SIGTERM	Termination	The process is asked to terminate. This signal can be caught and ignored, allowing the process to perform a clean shutdown.

- In shells like csh, the HUP signals are ignored automatically. That's is, you can run a program in the background and it will continue to run after you log out of the shell. In shells like BASH, this has to be done by placing `nohup` before the command.

Terminating a process

- Use the kill command to terminate a process
- The kill command sends a TERM signal by default, but it can be used to send any other signal as an argument to the command
- The TERM signal does not necessarily *kill* a process because it can be intercepted by the program and ignored
- The SIGKILL signal is often enough to terminate a process. However, in some cases, a reboot is required to kill a process which is waiting for a response that will never come from a disk or network device
- You can use `killall` *`process_name`* to kill all the processes associated with a program. Be ware that in UNIX, the same command will kill all the processes initialized by the current user. if it’s root, the system will halt immediately.
- The pkill command can be used to kill a process by it’s name

Process modes

- If you monitor the processes on a Linux system, you will find that they have one of the following letters beside them indicating their current state:
  - S: The process is *sleeping*. This means it is waiting for something or a resource
  - R: The process is *running* or *runnable*. This means it is being executed or can be executed whenever the CPU time is avaiable
  - Z: The process is supposed to be dead but it's not (Zombie). Normally it will die shortly. If not, a reboot may be required
  - T: The process is suspended (Traced). This can be obvious if you pressed CTRL-Z to pause a running command

The ps command

- It is the *de facto* command of process monitoring in Linux
- The most common argument is aux:
  - a: show all processes not just the current user's
  - u: user oriented output
  - x: show also processes that were not started from the terminal
- You can also use the l option instead of u to print the long version of the output. This prints the UID as a number rather than the username
- A similar output can be obtained by using the `-ef` command line switch

Real time process monitoring

- The top command offers a real time view of the processes running on the system with many useful information like the amount of CPU and memory used. Output is sorted by the CPU, but you can choose another sort field.
- You can use the top command to dynamically renice processes.
- Another command that extends top (third party) is htop. It can be downloaded from this URL: http://hisham.hm/htop/index.php?page=downloads
- Top is not a privileged command. that is, it can be run by other users than root.
- The default refresh interval is 3 seconds. This can be modified by using the –d option

Using the `/proc` filesystem

- It is a virtual filesystem that is created on the fly when the system boots
- Contains useful information about the processes currently running
- You can use ls to list the files and directories, cat to show file contents just as you deal with any other filesystem
- Given a process PID, you can access its properties in /proc like this:
  `ls -l /proc/PID/`
- Some useful directories include:
  - fd : for the file descriptors used by the process (STDIN, STDOUT, STDERR, and any files currently being written too, like log files for example)
  - cmdline: the command line that the process was started with
  - cwd: a symbolic link to the process working directory

Tracing a process

- You can use the `strace` command to monitor the system calls that a process makes on a very low level
- Strace is useful when you want to troubleshoot a hanging or a frequently-crashing process, because you will know exactly what was the last thing the process was trying to do before it crashed, or kept on waiting for.
- You can use `–f` to trace child processes (forked) as well
- The `–e` file can be used to direct the output to a file. This is useful when you want to monitor a process that will take a very long time, and you want to analyze it's behavior over that long period.