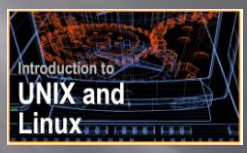


CST 126 – LESSON 11

Controlling User Processes

Chapter 10

Objectives

- To describe the concept of a process, and execution of multiple processes on a computer system with a single CPU
- To explain how a shell executes commands
- To discuss process attributes
- To explain the concept of foreground and background processes, including a description of a daemon
- To describe sequential and parallel execution of commands
- To discuss process and job control in UNIX; foreground and background processes, suspending processes, moving foreground processes into background and vice versa, and terminating processes
- To describe the UNIX process hierarchy
- To cover commands and primitives

Managing and Processing Processes

- Every running program is a separate entity, called a process.
- A process consists of several components working together, including the code, data, CPU activity, memory, input, output, and error handling.
- Each process involves reading instructions, accessing computer memory, reading from input, evaluating arguments, performing calculations, and writing to output.
- Every process on the system has its own unique process ID number.
- A process is a program in execution.

Running Multiple Processes Simultaneously

- The time a process is 'in' the CPU burst before it is switched 'out' of the CPU is called the ***quantum*** or ***time slice***
- The technique used to choose the process that gets to use the CPU is called ***CPU scheduling***

Running Multiple Processes Simultaneously (contd.)

- Firstcome,First-serve(FCFS)
  - The process that enters the system first is assigned the highest priority
- Assign priority value based on the amount of time a processor has used the CPU; a newly arriving process or a process that spends most of its time doing I/O operations(I/O bound processes)
- Round Robin (RR)
  - A process gets to use the CPU for one quantum and then the CPU is given to another process, the next process in the queue of processes waiting to use the CPU

Running Multiple Processes Simultaneously (contd.)

- Processor Scheduler
  - The operating system code that implements the CPU scheduling algorithm
- Dispatcher
  - The OS code that takes the CPU away from the current process and hands it over to the newly scheduled process
- Priority value=Threshold priority + Nice value +(Recent CPU usage/2)
- Threshold priority is an integer having a value of 40 or 60
- CPU usage is the number of clock ticks for which the process has used the CPU
- Nice value is a positive integer with a default value of 20

Unix Process States

- A UNIX process can be in one of many states , as it moves from one state to another , eventually finishing its execution(normally or abnormally) and getting out of the system

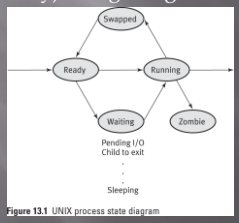


Figure 13.1 UNIX process state diagram

Unix Process States

TABLE 13.1 A Brief Description of the UNIX Process States

State	Description
Ready	The process is ready to run but does not have the CPU. Based on the scheduling algorithm, the scheduler decided to give the CPU to another process. Several processes can be in this state, but on a machine with a single CPU, only one can be executing (using the CPU).
Running	The process is actually running (using the CPU).
Waiting	The process is waiting for an event. Possible events are an I/O (e.g., disk/terminal read or write) is completed, a child process exits (parent waits for one or more of its children to exit), or the sleep period expires for the process.
Swapped	The process is ready to run, but it has been temporarily put on the disk (on the swap space); perhaps it needs more memory and there is not enough available at this time.
Zombie	A dying process is said to be in a zombie state. Usually, when the parent of a process terminates before it executes the exit call, it becomes a zombie process. The process finishes and finds that the parent is not waiting. The zombie processes are finished for all practical purposes and do not reside in the memory, but they still have some kernel resources allocated to them and cannot be taken out of the system. All zombies (and their live children) are eventually adopted by the granddaddy, the init process, which removes them from the system.

Execution of shell Commands

- A shell command can be external or internal
- An ***internal (built-in) command*** is one whose code is part of the shell process
  - bg, cd,continue, echo, exec
- An ***external command*** is one whose code is in a file; contents of the file can be binary code or shell script
  - grep,more ,cat, mkdir, rmdir, ls
- A UNIX process can create another process by using the fork system call, which creates an exact main memory map of the original process
- The forking process is known as the ***parent process***
- The created (forked) process is called the ***child process***

Execution of shell Commands (contd.)

- Shell script: a series of shell commands in a file
- Execution of a shell script is different from execution of an external binary command
- Execution of a shell script:
  - The current shell creates a child shell and lets the child shell execute commands in the shell script, one by one.
  - The child shell creates a child for every command it executes
  - While the child shell is executing commands in the script file, the parent shell waits for the child to terminate, after which it comes out of waiting state and resumes execution
  - Only purpose of child shell is to execute commands.

Process Attributes

- Owner's ID, process name, process ID(PID), process state, PID of parent process, length of time process has been running
- The **ps** command can be used to view the attributes of of processes running on the system
- **ps [options]** (System V version)

Purpose Report process status

Output Attributes of process running on the system

Commonly used options/features:

-a Display Information about the processes executing on your terminal except the session header (your login shell)

-e Display information about all the processes running on the system

-l Display long list(14 cols) of status report

-u uidlist Display information about processes belonging to the users with UIDs in the 'uidlist' (UIDs separated by commas)

```
 F S   UID   PID  PPID  C PRI  NI ADDR    SZ WCHAN  TTY          TIME CMD
100 S  1004  7719  1320  0  69   0    -   629 wait4  tty3     00:00:00 bash
000 S  1004  7746  7719  0  72   0    -   639 wait4  tty3     00:00:00 sh
000 R  1004  7758  7746  0  76   0    -   816 -      tty3     00:00:00 ps
```

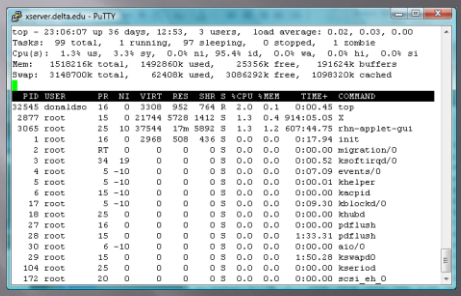
TABLE 13.2 Brief Description of Various Fields of the Output of the ps –l Command

Field	Meaning
F	Flags: Flags associated with the process indicate things like whether the process is a user or kernel process and why the process stopped or went to sleep
S	State: State of the process O Currently running (using the CPU) R Ready to run but not presently running S Sleeping for an event T Stopped (background process, suspended, or being traced) Z Zombie process (finished but is still using some kernel resources; created, for example, when parent dies before the process finishes)
UID	User ID: Process owner's user ID
PID	Process ID: ID of the process
PPID	Parent PID: PID of the parent process
C	CPU Usage: Recent CPU utilization, a parameter used in computing a process's priority for scheduling purposes
PRI	Priority: Priority value of a process that dictates when the process is scheduled; the smaller the priority value of a process, the higher its priority
NI	Nice value: The nice value of a process; another parameter used in the computation of a process's priority value
ADDR	Address: The memory or disk address of a process, that is, its location in the main memory or disk (for a swapped-out process)
SZ	Size: The size of the memory image of a process in blocks
WCHAN	Wait channel: Null for running processes; processes that are ready to run and are waiting for the CPU to be given to them; for a waiting or sleeping process, shows the event the process is waiting for
TTY	Terminal: Shows the terminal name a process is attached to
TIME	Time: The length of time (in minutes and seconds) a process has been running, or ran for before sleeping or stopping
CMD	Command: Lists the command used to start this process; the –f option is needed to see the full command in System V UNIX; otherwise, only the last component of the pathname is displayed

Process Attributes

ps -l

The Top Command



Process and Job control

- Unix is responsible for several process related activities including process creation, process termination, running processes in the foreground and background, suspending processes and switching processes from foreground to background and vice versa

Foreground and Background Process and Related Commands

- command (for foreground execution)
- command &(for background execution)
- **fg[%jobid]**
  - **Purpose** Resume execution of the process with job number 'jobid' in the foreground or move background processes to the foreground
  - **Commonly used values for '%jobid'**
    - **% or %+** Current job
    - **%-** Previous job
    - **%N** Job Number N
    - **%Name** Job beginning with 'Name'
    - **%?Name** Command containing 'Name'

Foreground and Background Process and Related Commands (contd.)

- **bg[%jobid-list]**

  Purpose: Resume execution of suspended processes/jobs with job numbers in 'jobid-list' in the background

  Commonly used values for '%jobid':
  - **% or %+** Curent job
  - **%-** Previous job
  - **%N** Job Number N
  - **%Name** Job beginning with 'Name'
  - **%?Name** Command containing 'Name'

Foreground and Background Process and Related Commands (contd.)

- **jobs [option] [%jobid-list]**

  Purpose: Display the status of the suspended and background processes specified in 'jobid-list'; with no list, display the status of the current job

  Commonly used options/features:
  - **-l** Also display PID of jobs

UNIX Daemons

- A **daemon** is a system process running in the background
- Used to offer various types of services to users and handle system administration tasks
  - Print, e-mail, finger

Sequential and Parallel Execution of Commands

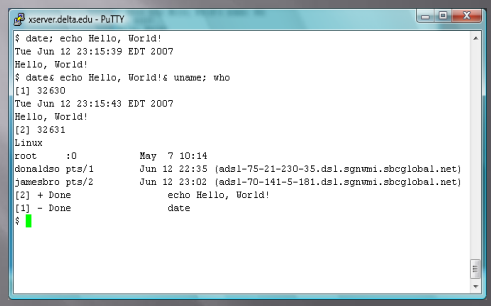
- **cmd1,;cmd2;…;cmdN**

  Purpose: Execute the 'cmd1', 'cmd2', 'cmd3',…,'cmdN' commands sequentially

- **cmd1& cmd2&…cmdN&**

  Purpose: Execute commands 'cmd1','cmd2',…'cmdN' in parallel as separate processes

Sequential and Parallel Execution of Commands (contd.)

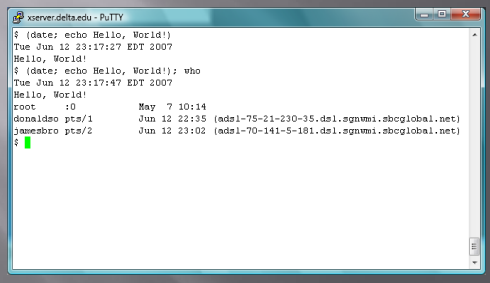


Sequential and Parallel Execution of Commands (contd.)

- UNIX allows you to group commands and execute them as one process by separating commands using semicolons and enclosing them in parenthesis. This is called **command grouping**.
- **(cmd1;cmd2;…cmdN)**

  Purpose Execute commands 'cmd1','cmd2'…,'cmdN' sequentially but as one process

Sequential and Parallel Execution of Commands (contd.)



Abnormal Termination of Commands and Processes

- Can terminate a foreground process by `<Ctrl-C>`
- Can terminate a background process:
  - Use the `kill` command
  - By first bringing the process into foreground by using the fg command and then pressing `<Ctrl-C>`

Abnormal Termination of Commands and Processes (contd.)

- The primary purpose of a `kill` command is to is to send a signal (**software interrupt**) to a process
- A process can take one of three actions upon receiving a signal
  - Accept the default action as determined by the UNIX kernel
  - Ignore the signal
  - Intercept the signal and take a user-defined action
- A signal caused by an event internal to a process is known as an **internal signal** or **trap**.
- A signal caused by an event external to a process is called an **external signal**

Abnormal Termination of Commands and Processes (contd.)

- kill [-signal_number] proc-list
- Kill –l

Purpose Send the signal for 'signal_number' to processes whose PIDs or jobIDs are specified in the 'poc-list'jobIDs must start with %.The command kill –l return a list of all signals and their names (on some systems, numbers are not displayed)

Commonly used signal_numbers:

1	Hangup
2	Interrupt(<Ctrl-C>)
3	Quit(<Ctrl-\>)
9	Sure kill
15	Software signal (default signal number)

Abnormal Termination of Commands and Processes (contd.)

`nohup commands [args]`

Purpose: Run command and make it immune to the hangup signal

```
$ nohup find / -name foo -print 1> foo.paths 2> /dev/null &
[1] 15928
$
```

Process Hierarchy in UNIX

- The process which has no parent is called the **init process** and is the granddady of all the processes that are created so long as the system is up and running
- The init process has a PID of 1
- The **login process** prompts you for your password and checks the validity of your login name and password
- The **swapper** and **init** processes exist throughout the life time of a system
- The **getty process**, which monitors a terminal line, lives for as long as the terminal is attached to the system
- Use the ps –efH command to display the process tree of the currently running processes on the system , showing the parent-child relationship

Summary

- Programs running in a system are executed by a process that reads the appropriate code and accomplishes the tasks.
- All the processes have their own unique PID, the PID of their parent, an owner, group, memory, code, input, output, error, and their tty port.
- A command-line can consists of one process or a series of processes connected by pipes or semicolons, and is often called a job.
- The kill command terminates processes identified by either their process ID or job number/job name.