UNIX

Processes and Related
Commands

Lesson Objectives

- UNIX processes:
  - Parent and Child processes
  - Process Status Command – ps
  - Running processes in background mode
  - Terminate process
  - Process scheduling

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

6.1: UNIX Processes
What is a Process?

- Characteristics of processes:
  - Process is an instance of program in execution.
  - Many processes can run at the same time.
  - Processes are identified by the Process Identifier.
  - PID is allocated by kernel.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

UNIX Processes:
When you execute a program on your UNIX system, the system creates a special environment for that program. This environment contains everything needed for the system to run the program as if no other program were running on the system.

What is a Process?
A process is an instance of a program in execution. When any executable file is executed, a process starts. It remains active while the program is executing. When the program terminates, the process dies. Generally the name of the process is the name of the executable.

Since Unix is a multi-tasking system, there can be many processes that run at the same time. A unique number called as the Process Identifier, PID, identifies each of these processes. The kernel allocates this PID, which is a number from 0 to 32767. It is the responsibility of the kernel to manage the processes – in terms of time allocated to process, its associated priorities and swapping etc.

6.1: UNIX Processes > 6.1.1: Parent and Child Processes

Concepts

- On logging to a system, a process is set up due to execution of shell.
- Shell is the parent process for every other process setup due to the execution of commands.
- Every process, with the exception of PID 0 processes, has a parent process.
- Parent process waits for death of child process before resuming execution.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Parent and Child Processes:

When a user logs on to the system, the kernel sets up a process – this is actually the process set up due to execution of sh command. This process remains active till the user logs off. The sh process is identified by the special variable $$. Any logging off and logging on again will result in an assignment of a different PID. The knowledge of this PID is required for controlling the activities of the terminal.

Any command written at the prompt is actually the standard input to the sh program. When a external command is run from the command line, the shell process spawns a new process for the command, which remains active till the command is active. The shell here is the parent process while the new process is the child process. The child process inherits the environment of the parent process. The child process can alter the operating environment that it has inherited, but the modified environment will not be available to the parent after the death of the child process.

Every process has a parent. The exception is the first process, with PID 0, which does not have a parent. This is set up when the system boots. It can be treated as analogous to the "root" in the file system.
Every process can have only one parent process. However, a process can spawn multiple processes. For example, when a pipeline is set up between 2 commands, the sh process would set up two processes for each of the commands.

On the completion of the child process, a signal is sent to the parent, and the control is reverted back to the parent process. However, if the parent process dies, the child processes automatically die.

6.1: UNIX Processes > 6.1.1: Parent and Child Processes
Running a Command

- ls command: Steps for running a Unix command
  - The shell performs a fork. This creates a new process that the shell uses to run the ls program.
  - The shell performs an exec of the ls program. This replaces the shell program and data with the program and data for ls and then starts running that new program.
  - The ls program is loaded into the new process context, replacing the text and data of the shell.
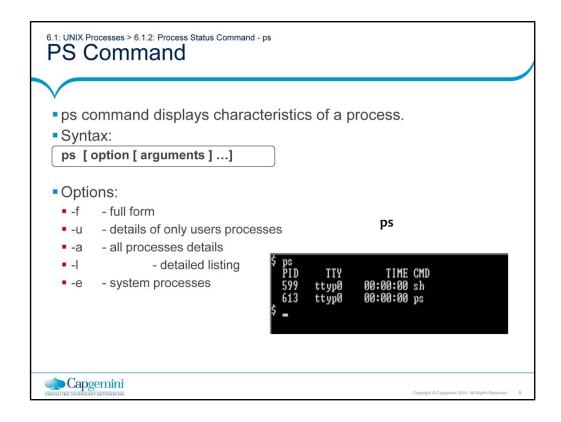 - The ls program performs its task, listing the contents of the current directory .

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Running a Command:

When you enter ls to look at the contents of your current working directory, UNIX does a series of things to create an environment for ls and the run it.
Internally, Unix creates a process with the fork system call. This system call creates a copy of the process that invokes it. This child process is an image of the parent process, but with a new PID. The exec system call is then used – the parent process will overwrite the image of child with the copy of the program that is to be executed. After this, the parent executes the wait system call – parent will continue to wait till the death of the child. After this, parent can continue with its other activities.

Process is NOT set for all commands. The shell recognizes 2 types of commands – external and internal. External commands are commands like cat, ls etc or utilities. Shell scripts also come under the category of external commands. A process will be set up for the external commands. The internal commands are the shell's own built in statements, and commands like cd and echo etc. No process is set up for such commands.

It is necessary that some commands are built into the shell itself and no process is set up. That is because it is very difficult or sometimes impossible to implement some commands as external commands.

6.1: UNIX Processes > 6.1.2: Process Status Command - ps

PS Command

- ps command displays characteristics of a process.
- Syntax:

  ps [option [arguments] …]

- Options:
  - -f   - full form
  - -u   - details of only users processes
  - -a   - all processes details
  - -l   - detailed listing
  - -e   - system processes

ps

```
$ ps
PID     TTY        TIME CMD
599    ttyp0    00:00:00 sh
613    ttyp0    00:00:00 ps
$ _
```

PS Command:

            The ps command can be used to display the characteristics of the
processes. It has the knowledge of kernel built into it, using which it can read the
process tables to get necessary information.

            This command is one of the few Unix commands, which generates
header information. It is also a highly variant command – the exact output obtained
depends on the version of Unix as well as the hardware used.

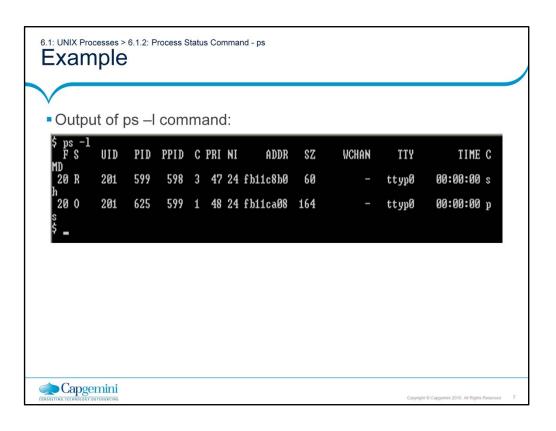   When used without any options, ps command displays the
following:
 Process Id
 The terminal with which the process is associated
 Cumulative processor time of the process
 The process name

6.1: UNIX Processes > 6.1.2: Process Status Command - ps

Example

- Output of ps –l command:

```
$ ps -l
 F S    UID  PID  PPID  C PRI NI     ADDR   SZ    WCHAN    TTY      TIME C
MD
 20 R   201  599  598   3  47 24 fb11c8b0  60       -    ttyp0  00:00:00 s
h
 20 O   201  625  599   1  48 24 fb11ca08  164      -    ttyp0  00:00:00 p
s
$ _
```

Process Status Command: Example:
l – It displays Long format
F - Octal flags which are added together to give more information about the current status of a process(20 – process is loaded in primary memory: it has not been swapped out to disk)
S - State of the process (O – Process is running on a processor, R – Process is on run queue)
UID - The Userid of the process owner (login name is printed using –f option)
PID - The process ID of the process (this number is needed to kill a process)
PPID - The process ID of the parent process
C - CPU usage by the process; combination of this value with nice value is used to calculate the priority
PRI - The priority of the process (lower number mean lower priority)
NI - The nice value of the process
ADDR - The virtual address of the process entry in the process table

6.1: UNIX Processes > 6.1.3: Running Process in Background Mode

Process in Background Mode

- Processes can run in foreground or background mode.
  - Only one process can run in foreground mode but multiple processes can run in background mode.
  - The processes, which do not require user intervention can run in background mode, e.g. sort, find.
  - To run a process in background, use & operator
    - $sort -0 emp.lst emp.lst &
- nohup (no hangup) - permits execution of process even if user has logged off.
  - $nohup sort emp.lst &   (sends output to nohup.out)

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Process in Background Mode:

It is possible to have only one job working in the foreground. But the other jobs can be made to run at the background. Background execution is a useful feature if more important jobs are to be run in the foreground and less important ones relegated to the background.

The & operator is provided by the shell to run a process in the background. On invoking a command terminated with &, shell immediately returns a PID for this number; and the shell is ready to accept another command even though the previous command is not terminated yet. It will be used as follows:

$ sort –o emp.lst emp.lst &

Even if a command is run in the background, it is possible that the output as well as error messages are still coming to the standard output. Hence, care should be taken to suitably redirect this output.

However, too many jobs should not be run in the background as significant deterioration of CPU performance can occur.

6.1: UNIX Processes > 6.1.4: Terminate Processes
Kill Command

- Kill Command- Used to terminate a process
- Syntax :
  - kill [-signumber] pid ...
- Example:
  - $kill 1005   (default signal 15) - kills job with pid 1005
  - $kill -9 1005 - sure killing of job
  - $kill 0 - kills all background process

The Kill Command:
It is possible to send signals to processes. When a process receives a signal, it can ignore it, terminate or do something else. Signals in Unix are identified by a number – each signal notifying that an event has occurred.
Other syntax

```
kill -s signame pid ...
kill -l [ exit_status ]
kill [ -signame ] pid ...
```

A signal number 15 is used by default by the kill command to terminate a process. The kill command takes one or more PID numbers as its arguments and facilities premature termination of these processes. It is used as follows (assumed that process numbered 117 is being killed):

> $ kill 117

It is possible that some programs simply ignore this command and continue normal execution. In that case, a "sure kill", with signal number 9, has to be employed.

> $ kill –9 117

In order to kill all processes except the login shell, an argument of 0 is passed to the kill command.

Of course, one can kill only one's own processes. Besides, some system processes cannot be killed at all.

6.2: Process Scheduling > 6.2.1: Overview of Process Scheduling

Details

- Scheduling Policy:
  - *time-sharing* technique
  - *Several* processes are allowed to run "concurrently," which means that the CPU time is roughly divided into "slices," one for each runnable process.
  - The scheduling policy is also based on process priority
  - In UNIX, process priority is dynamic.

6.2: Process Scheduling > 6.2.1: Overview of Process Scheduling

Continued…

- Processes are traditionally classified as "I/O-bound" or "CPU-bound."
  - **I/O-bound Processes:**
    Make heavy use of I/O devices and spend much time waiting for I/O operations to complete.
  - **CPU-bound Processes:**
    Are number-crunching applications that require a lot of CPU time.

Overview of Process Scheduling:

When speaking about scheduling, processes are traditionally classified as "I/O-bound" or "CPU-bound."

I/O-bound Processes: They make heavy use of I/O devices and spend much time waiting for I/O operations to complete.

CPU-bound Processes: They are number-crunching applications that require a lot of CPU time.

6.2: Process Scheduling > 6.2.1: Overview of Process Scheduling

Continued…

- Processes can also be classified as:
  - **Interactive processes**:
    These interact constantly with their users, and therefore spend a lot of time waiting for key presses and mouse operations.
  - **Batch processes**:
    These do not need user interaction, and hence they often run in the background.
  - **Real-time processes:**
    - Should never be blocked by lower-priority processes.
    - Should have a short response time.

Overview of Process Scheduling (contd..):
An alternative classification distinguishes three classes of processes:

Interactive processes
These interact constantly with their users, and therefore spend a lot of time waiting for key presses and mouse operations. When input is received, the process must be woken up quickly, or the user will find the system to be unresponsive. Typically, the average delay must fall between 50 and 150 ms. The variance of such delay must also be bounded, or the user will find the system to be erratic. Typical interactive programs are command shells, text editors, and graphical applications.
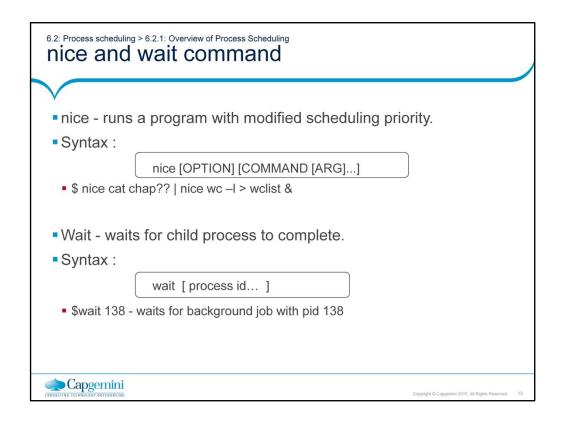
Batch processes
These do not need user interaction, and hence they often run in the background. Since such processes do not need to be very responsive, they are often penalized by the scheduler. Typical batch programs are programming language compilers, database search engines, and scientific computations.

Real-time processes:
These have very strong scheduling requirements. Such processes should never be blocked by lower-priority processes, they should have a short response time and, most important, such response time should have a minimum variance. Typical real-time programs are video and sound applications, robot controllers, and programs that collect data from physical sensors.
The two classifications we just offered are somewhat independent. For instance, a batch process can be either I/O-bound (e.g., a database server) or CPU-bound (e.g., an image-rendering program). While in UNIX real-time programs are explicitly recognized as such by the scheduling algorithm, there is no way to distinguish between interactive and batch programs. In order to offer a good response time to interactive applications, UNIX (like all Unix kernels) implicitly favors I/O-bound processes over CPU-bound ones.

6.2: Process scheduling > 6.2.1: Overview of Process Scheduling
nice and wait command

- nice - runs a program with modified scheduling priority.
- Syntax :

  nice [OPTION] [COMMAND [ARG]...]

  - $ nice cat chap?? | nice wc –l > wclist &

- Wait - waits for child process to complete.
- Syntax :

  wait [process id…]

  - $wait 138 - waits for background job with pid 138

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

nice and wait Command:

All processes on Unix are usually executed with equal priority. In order to reduce the priority of a job, the command needs to be prefixed with nice. By default, nice reduces the priority of any process by 10 units. The amount of reduction can also be specified as an argument (value from 0 to 19) to the nice command. Incase of commands in a pipeline, to reduce the priority of all commands in the pipeline, nice needs to be used in each element in the pipeline.

$ nice cat chap?? | nice wc –l > wclist &

The wait command can be used to wait for the completion of a background process. This is a built-in shell command – no process is spawned for this command. It sends the shell into a wait state so that it can acknowledge the death of child processes.

cron

- A system daemon which performs a specific task at regular intervals

- The command and schedule information is kept in the directory /var/spool/cron/crontabs or in /usr/spool/cron/crontabs.

- Each user has a crontab file. cron wakes up periodically and executes any job that are scheduled for that minute.

- Only users who are listed in /etc/cron.allow or not listed in cron.deny can make an entry in the crontab.

- Crontab <filename> -used to make an entry in the crontab file.

  - where the file contains the commands to execute

MIN	HOUR	DOM	MOY	DOW	COMMAND
(0-50)	(0-23)	(1-31)	(1-12)	(0-6)	---
$ 0	18	*	*	*	/home/gather

You can view the contents of the global cron file by logging in as root and typing:
crontab -l
The cron process executes the jobs it contains at the times that are
specified. Each "cron job" is composed of six fields. Fields one to five contain
clock information which specify when the command (given in the sixth field) should
be executed.
Minute Hour Day Month Day Task
Minute = Minute of the hour, 00 to 59. * Will indicate every minute (details later)
Hour = Hour of the day in 24-hour format, 00 to 23. * Will indicate every hour
(details later)
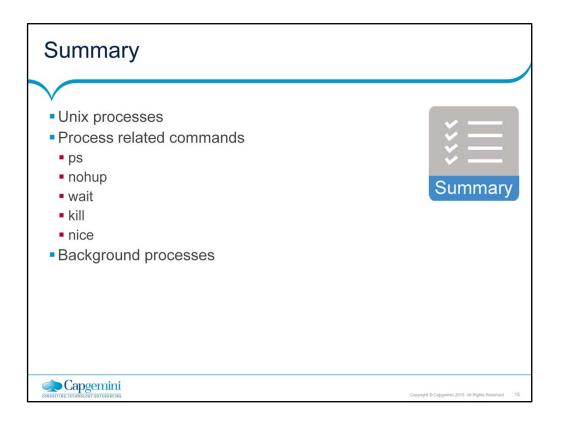Day = Day of the month, 1 to 31. * Will indicate every day (details later)
Month = Month of the year, 1 to 12. * Will indicate every month (details later)
Day = Day of the week, 3 chars - sun, mon, tue, or numeric (0=sun, 1=mon etc)....
* Will indicate every day (details later)
Task = The command you want to execute

Note: each of the above must be separated by at least 1 space.

Summary

- Unix processes
- Process related commands
  - ps
  - nohup
  - wait
  - kill
  - nice
- Background processes

Review Questions

- Complete The Following :
  - A unique number called the _____ identifies each process.
  - Processes using heavy i/o are called as _____

- True / False
  - A signal number of 9 is used, by default, by the kill command to terminate a process.
  - You can kill any process, including the system process, using the kill command.