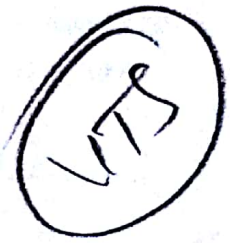
UNIX PROGRAMMING (JNTUK-R16)

UNIT-6: THE PROCESS



SYLLABUS:

- The Meaning
- Parent and Child Processes
- Types of Processes
- More about Foreground and Background processes
- Internal and External Commands
- Process Creation
- The Trap Command
- The Stty Command
- The Kill Command
- Job Control

Text Books:

The Unix programming Environment by Brain W. Kernighan & Rob Pike, Pearson.
Introduction to Unix Shell Programming by M.G.Venkateshmurthy, Pearson.

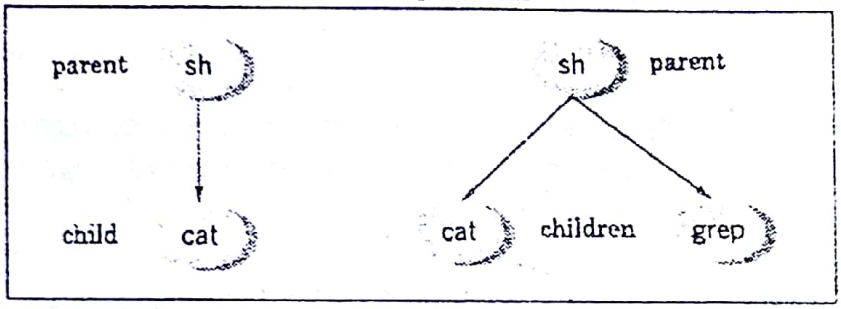
1. The Meaning of a Process:

- All the programs that are loaded into the memory for execution are called Processes.
- A program in execution is called process
- All programs share the same CPU, because Unix is Multi-user, Multi-tasking Operating System
- The Kernel generates or spawns processes for every program under execution and allocates definite and equal CPU time slots to these various programs.
- Each of these processes have a unique identification number allocated to it by the kernel, called PIDs
- PID means Process Identification Number
- Mathematically, a process is represented by the tuple
  (PID, Code, Data, Register Values, PC Value)
- Process ID (PID) is the unique identification number that is used to identify the process uniquely from other processes.
- At present the maximum value of PID is 32767
- Code is the program code under execution
- Data is the data used during execution
- Register values are the values in CPU registers
- PC value is the address in the program counter from where the execution of the program starts or continues
- As soon as the system is booted, the kernel gets loaded into the memory and then gets executed
- Immediately, a system process called the swapper is created.
- The PID of this process will be 0 (zero)
- The process 0 creates another process called init, meaning initialiser
- The PID of init process is 1
- The init sets the user mode in either the single user or the multi-user mode

2. Parent and Child Processes:

- In Unix, a process is responsible for generating another process
- A process that generates another process is a parent
- The newly generated process is a child
- For example, the shell sh being process, generates another process cat or processes like cat and grep
- Here, sh is the parent process, cat and grep are child processes

Example for parent and child process:



- When a parent creates or generates a child process, a process is said to have born.
- As long as a process is active, it is said to be alive
- Once the job of the process is over it becomes inactive and is said to be dead
- The cat and grep processes are the children of the same parent will have different PIDs.
- In general, a parent process waits for the complete execution of its child process
- A parent waits for its child to die.
- Sometimes a parent may die before its child
- In such cases, the child is said to be orphan
- Generally, these orphan processes are attached to the init process - the process with PID1

Program and a Process:

- All processes get themselves arranged in the form of a hierarchical, inverted tree like structure
- This is similar to file organization
- The only difference between file organization and process organization is
- File organization is locational whereas
- Process organization is temporal
- A program exists in a single location in space and exists for any length of time
- Thus a program is a static object that exists in a file
- It contains just the entire set of instructions
- But a process is a program in execution
- Thus it is dynamic object and can never be in a file
- It is a sequence of instructions under execution
- Thus a process has a definite life cycle

3. Types of processes:

- Processes in Unix are classified into 3 types
  - i) Interactive Process
  - ii) Non-interactive Process
  - iii) Daemons

i) Interactive Processes:

- These are also called as Foreground Processes.
- All the user processes are created by users with the shell, act up on the directions of the users and are normally attached to the terminal are called interactive processes

ii) Non-interactive processes:

- These are also called as Background Processes
- Certain processes can be made to run independent of terminals
- Such processes that run without any attachment to a terminal are called non-interactive processes.

iii) Daemons:

- All processes that keep running always without holding up any terminals and keep waiting for certain instructions either from the system or the user and then immediately get into action are called daemons.
- swapper, init, cron, bdflush, vhandle are some examples of daemons.
- These daemons come into existence as soon as the system is booted and will be alive till the system is shut down.
- One cannot kill these processes prematurely.

4. More about Foreground and Background Processes:

- When a command is given, the shell parses, rebuilds and then hands it over to the kernel for execution.
- The shell then keeps on waiting for the kernel to complete the execution.
- During the shell-waiting period the user cannot issue any other command because the terminal is held up with the command under execution.

Foreground Processes:

- Commands that hold up the terminal during their execution are called Foreground Processes

Advantage:

- No further commands can be given from the terminal as long as the older one is running

Disadvantage:

- When a currently running process is big then it takes a lot of time for processing.

Background Processes:

- It is possible to make processes to run without using the terminal.
- Such processes take their input from some file and process it without holding up the terminal (non-interactively) and write their output on to another file are called Background Processes
- Typical jobs that could be run in background are sorting of a large database file or locating a file in a big file system by using the find command and so on.

Running a command in the Background:

- A command is made to run in the background by terminating the command line with an ampersand (&) character

Example:

```
$sort -o student.lst student.lst &
567
$
```

- The shell immediately returns the PID as well as the shell prompt $
- Here, 567 is the PID of the just submitted background job

Problems using background processes:

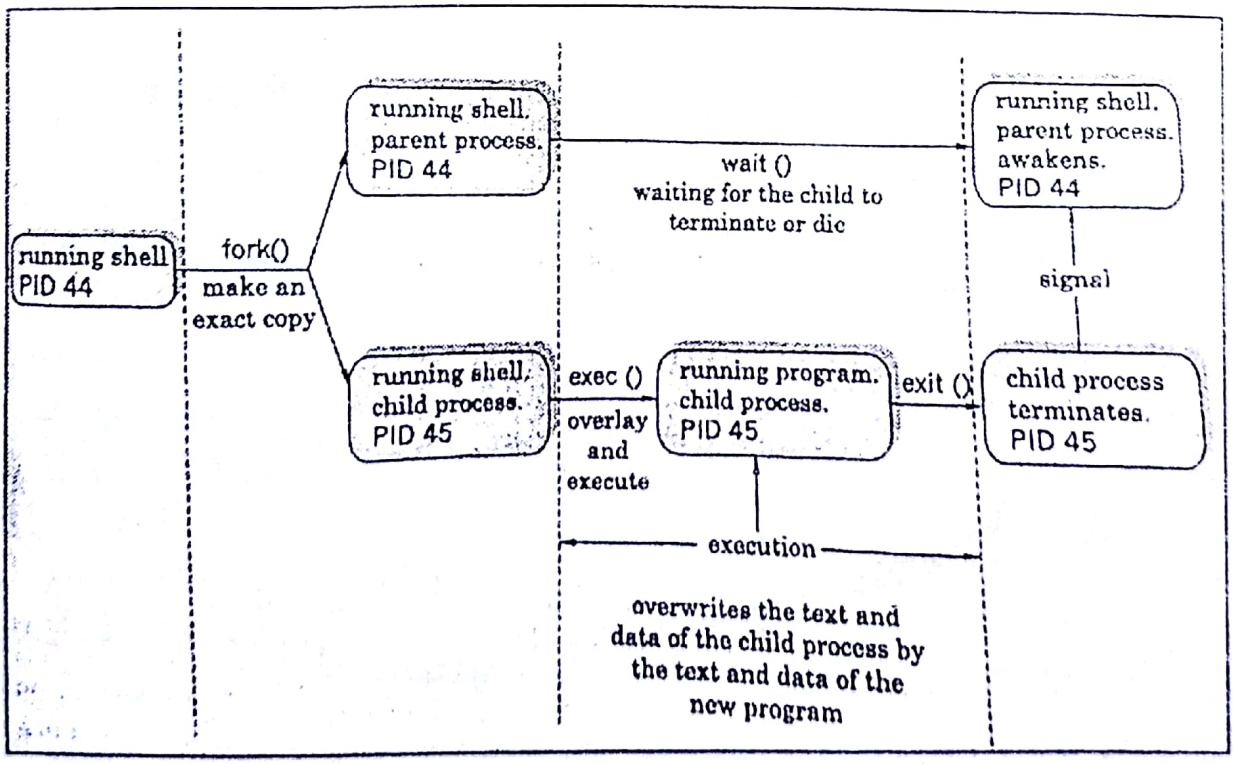
- The success or failure of the background processes is not reported. The user has to find it out using PID.
- The output has to be redirected to a file as otherwise the display on the monitor gets mixed up
- Too many processes running in the background degrades the overall efficiency of the system
- There is a danger of the user logging out when some processes are still running in the background.

5. Internal and External Commands:

- The classification of commands depending on whether they generate separate processes or not upon their running
- Most of the commands such as cat, who and others generate separate processes as soon as they are used.
- Commands that generate separate processes upon their running are called external commands.
- Some commands such as mkdir, rm, cd and others do not generate new processes when they are used.
- Such commands are called internal commands.

6. Process Creation:

- There are three distinct phases in the creation of a process
    - Forking
    - Overlaying and Execution
    - Waiting
- Mechanism of Process Creation is depicted as,



- Forking is the first phase in the creation of a process by a process.
- The calling process (parent) makes a call to the system routine fork( ) which then makes an exact copy of itself.
- After the fork( ) there will be two processes.
- The fork of the parent process returns the PID of the new process, that is the child process just created
- The fork of the child returns a 0 (zero)
- Immediately after forking, the parent makes a system call to one of the wait( ) functions
- By doing so, the parent keeps waiting for the child process to complete its task.
- In the second phase, the exec( ) function overwrites the text and data area of the child process.
- The exit( ) function terminates the child process.
- The parent awakens only when it receives a complete signal from the child, after which it will be free to continue with its other functions

7. The ps command:(knowing process attributes)

The ps command is used to display the attributes of processes that are running currently
When used with no options, the ps command lists out certain attributes associated with the terminal

Example:

```
$ps
  PID      TTY      TIME      CMD
  476      tty03    00:00:01  login
  659      tty03    00:00:01  sh
  684      tty03    00:00:00  ps
$
```

- The options used in ps command are

Option	Meaning
-a	all users
-f	full list
-u	user
-t	terminal
-e	every

8. Signals:

- A signal is a message sent to a program under execution, that is a process.
- It occurs in two occasions
  - Under some error conditions or the user interruption, the kernel generates signals
  - During inter-process communication between two or more processes. The participating process generates these signals. For example, a child process sends a signal to its parent process upon its termination.
- In Unix, Signals are identified by integers.
- They have names too.
- These names are in Uppercase and starts with SIG.

Example Signals:

Signal Number	Name	Function
1	SIGHUP	Hangup; closes process communication links
2	SIGINT	Interrupt; tells process to exit (ctrl-c)
3	SIGQUIT	Quit; forces the process to quit (ctrl-\)
9	SIGKILL	Sure kill; cannot be trapped or ignored
15	SIGTERM	Software termination; default signal for the kill command
24	SIGSTOP	Stop; (ctrl-z)

9. The trap command:

- Normally signals are used to terminate the execution of a process either intentionally or unintentionally.
- The trap command is used to trap one or more signals and then decide about the further course of action
- If no action is mentioned, then the signal or signals are just trapped and the execution of the program resumes from the point from where it had been left off

Syntax:

$trap [commands] signal_numbers

- The command part is optional
- The commands used must be enclosed using either single or double quotation marks

- Multiple commands in the commands part are separated by the ; (semicolon) character

Example:

$trap "echo killed by signal 15; exit" 15

- When a process receives a kill command, causing signal 15, it gives the message killed by signal 15 and then terminates the current process because of the execution of the exit command

Example:

$trap "ls -l" 1 2 3

- When a process generates any one of the signals 1,2 or 3, a long list of the current working directory is generated and then execution of the process resumes from the point where it had been left off

Example:

$trap "" 1 2 3 15

- This command just traps the signal numbers 1, 2, 3 and 15
- Certain signals like signal number 9 (the sure kill) cannot be trapped.

Resetting Traps:

- Normally a trap command changes the default actions of the signals.

- There are some situations like one might need to trap a certain signal in one part of a script and need the same signals not to be trapped in some other part
- The command to trap the signal is given as,

Example:

```
$trap "exit" 2 3 15
```

- The effect of the signals 2, 3 and 15 are restored by using the trap command without the command part in it.

Example:

```
$trap 2 3 15
```

10. The stty command:

- One of the most widely used methods to communicate with a system is to use terminals, that is via keyboards.
- There are certain combinations of keys, on these terminals, which control the behavior of any program in execution.

Keys	Function
<ctrl-m> (^m)	It is the <RETURN> key to end a command line and execute the command
<ctrl-c> (^c)	To interrupt a current process and to come back to the shell
<ctrl-s> (^s)	To pause display on the monitor
<ctrl-d> (^d)	To indicate end of file and so on

- The stty command is used to see or verify the settings of different keys on the keyboard
- The user can have a short listing of the settings by using this command without any arguments.
- In order to see all the settings, it has to be used with the -a (all) option

Example:

```
$stty -a
  speed 9600 baud; ispeed 9600 baud; line = O(tty);
  erase = ^?; kill = ^U; eof = ^D; intr = ^C; stop = ^S;
  echo echoe ----------------------------------------
$
```

- From the output one can see that
  - the terminal speed is 9600 bauds
  - ^U is used for killing a line
  - ^D is used to indicate end of file
  - echo typed text is echoed on the display terminal

11. The kill command:

- There are certain situations when one likes to terminate a process prematurely
- Some of these situations are,
  - When the machine has hung.
  - When a running program has gone into an endless loop.
  - When a program is doing unintended things.
  - When the system performance goes below acceptable levels because of too many background processes.

- Terminating a process prematurely is called killing.
- Killing foreground process is straight forward
- This is done by using the DEL key or the BREAK key.
- To kill a background process a kill command is used.
- This command is given with the PID of the process to be killed as its argument.
- If the PID is not known the ps command is used to know the same.

Example:

```
$kill 555
```

- Here, 555 is the PID of the process

- More than one process can be terminated using a single kill command

Example:

```
$kill 330  333  375   # Here, 330, 333, 375 are PID's
```

- A kill command, when invoked, sends a termination signal to the process being killed
- When used without any option, it sends 15 as its default signal number.
- This signal number 15 is known as the software termination signal and is ignored by many processes
- For example, the shell process sh, ignores signal 15

- One can use signal 9, the sure kill signal, to terminate a process forcibly

Example:

```
$kill -9 666    # 666 id the PID
```

- All the processes of a user (except his/her login shell) can be terminated by using a 0 as the argument of the kill command

Example:

```
$kill 0    # kills all the processes except the login sheli
```

- Using 9 as option and 0 as the argument, all processes including the shell can be killed

Example

```
$kill -9 0  # kills all the processes including the login shell
```

$! And $$ System variables:

- The special system variable $! holds the PID value of the last background job.
- The last background job can be killed by using the command

```
$kill $!
```

- The special system variable $$ holds the PID value of the current shell.
- The current shell can be killed using the sure kill command

```
$kill -9 $$
```

12. The wait command:

- Sometimes it is necessary to wait for either all the background jobs or a specific job to be executed completely before any further action is initiated.
- These situations are handled by the wait command

Example:

```
$wait       # waits till all the background processes are completely executed
$wait 227   # waits till the PID 227 process to be completely executed
```

13. Job Control:

- A command or a command line with a number of commands put together or a script is referred as a job
- In UNIX, as one can run commands in the background, there could be a number of commands, that is, processes, running in the background
- Also there could be command - a process – running in the foreground.

The jobs command:

- A list of all the current jobs is obtained using the jobs command.

Example:

```
[ksh] jobs                  # The {ksh} prompt has been used intensionally
[1] + Running vi sample.sh &
[2] - Running sleep 30  &
[ksh]
```

- Here, a + (plus) and a - (minus) that appear after the job number mark the current and previous jobs, respectively
- The word Running indicates that the job is currently being executed.

The fg command:

- This command is used to bring a job that is being executed in the background currently to the foreground
- This command can be either used without any argument or with a job number as its argument.

Example1:

```
[ksh] fg      #Brings the most recent background process to the foreground
```

Example2:

```
[ksh] fg %2   #Brings job number 2 to the foreground
```

Example3:

```
[ksh] fg %sort  #Brings the job the name of which begins with sort to the foreground
```

- Whenever a job number is used as an argument with a job-control command, it must be preceded by a percent sign (%)
- The current job may be referred by using %1 (or) %+ (or) %%

The bg command:

- A new job can be made to run in the background by using the & (ampersand) at the end of a command line.
- The currently running foreground process is first suspended by using the <ctrl-z> keys, and then making it run in the background by using the bg command.

Example:

[ksh] bg %1 #resumes job number 1 in the background

Important Questions:

1. a) What is meant by Process. Explain about Parent and Child Processes
   b) What are the types of Processes?
2. a) Explain about Foreground and Background Processes
   b) Explain about Process Creation
3. Explain about a) trap b) stty c) kill d) Job Control