Technical Papers UNIX Concepts: Unix Process Model and IPC

1. Brief about the initial process sequence while the system boots up. While booting, special process called the 'swapper' or 'scheduler' is created with Process-ID 0. The swapper manages memory allocation for processes and influences CPU allocation. The swapper inturn creates 3 children: The process dispatcher, vhand and dbflush with IDs 1, 2 and 3 respectively. This is done by executing the file/etc/init. Process dispatcher gives birth to the shell. Unix keeps track of all the processes in an internal data structure called the Process Table (listing command is ps-el).

2. What are various IDs associated with a process? Unix identifies each process with a unique integer called ProcessID. The process that executes the request for creation of a process is called the 'parent process' whose PID is 'Parent Process ID' Every process is associated with a particular user called the 'owner' who has privileges over the process. The identification for the user is 'UserID' Owner is the user who executes the process. Process also has 'Effective User ID' which determines the access privileges for accessing resources like files. Getpid () -process id getppid () -parent process id getuid () -user id geteuid () -effective user id

3. Explain fork () system call. The 'fork ()' used to create a new process from an existing process. The new process is called the child process, and the existing process is called the parent. We can tell which is which by checking the return value from 'fork ()' The parent gets the child's pid returned to him, but the child gets 0 returned to him.

4. Predict the output of the following program code main () { fork (); printf ("Hello World!"); } Answer: Hello World! Hello World! Explanation: The fork creates a child that is a duplicate of the parent process. The child begins from the fork (). All the statements after the call to fork () will be executed twice (once by the parent process and other by child). The statement before fork () is executed only by the parent process.

5. Predict the output of the following program code main () { fork (); fork (); fork (); printf ("Hello World!"); } Answer: "Hello World" will be printed 8 times. Explanation: 2^n times where n is the number of calls to fork ()

6. List the system calls used for process management: System calls Description fork () To create a new process exec () To execute a new program in a process wait () To wait until a created process completes its execution exit () To exit from a process execution getpid () To get a process identifier of the current process getppid () To get parent process identifier

nice () To bias the existing priority of a process brk () To increase/decrease the data segment size of a process

7. How can you get/set an environment variable from a program? Getting the value of an environment variable is done by using 'getenv ()' Setting the value of an environment variable is done by using 'putenv ()'

8. How can a parent and child process communicate? A parent and child can communicate through any of the normal inter-process communication schemes (pipes, sockets, message queues, shared memory), but also have some special ways to communicate that take advantage of their relationship as a parent and child. One of the most obvious is that the parent can get the exit status of the child.

9. What is a zombie? When a program forks and the child finishes before the parent, the kernel still keeps some of its information about the child in case the parent might need it- for example, the parent may need to check the child's exit status. To be able to get this information, the parent calls 'wait ()' In the interval between the child terminating and the parent calling 'wait ()' the child is said to be a 'zombie' (If you do 'ps' the child will have a 'Z' in its status field to indicate this.)

10. What are the process states in Unix? As a process executes it changes state according to its circumstances. Unix processes have the following states: Running: The process is either running or it is ready to run. Waiting: The process is waiting for an event or for a resource. Stopped: The process has been stopped, usually by receiving a signal. Zombie: The process is dead but have not been removed from the process table.

11. What Happens when you execute a program? 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. Each process has process context, which is everything that is unique about the state of the program you are currently running. Every time you execute a program the UNIX system does a fork, which performs a series of operations to create a process context and then execute your program in that context. The steps include the following: Allocate a slot in the process table, a list of currently running programs kept by UNIX. Assign a unique process identifier (PID) to the process. iCopy the context of the parent, the process that requested the spawning of the new process. Return the new PID to the parent process. This enables the parent process to examine or control the process directly. After the fork is complete, UNIX runs your program.

12. What Happens when you execute a command? When you enter 'ls' command 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: The shell has UNIX perform a fork. This creates a new process that the shell will use to run the ls program. The shell has UNIX perform 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.

13. What is a Daemon? A daemon is a process that detaches itself from the terminal and runs, disconnected, in the background, waiting for requests and responding to them. It can also be defined as the background process that does not belong to a terminal session. Many system functions are commonly performed by daemons, including the sendmail daemon, which handles mail, and the NNTP daemon, which handles USENET news. Many other daemons may exist. Some of the most common daemons are: Init: Takes over the basic running of the system when the kernel has finished the boot process. Inetd: Responsible for starting network services that do not have their own stand-alone daemons. For example, inetd usually takes care of incoming rlogin, telnet, and ftp connections. Cron: Responsible for running repetitive tasks on a regular schedule.

14. What is 'ps' command for? The ps command prints the process status for some or all of the running processes. The information given are the process identification number (PID), the amount of time that the process has taken to execute so far etc.

15. How would you kill a process? The kill command takes the PID as one argument; this identifies which process to terminate. The PID of a process can be got using 'ps' command.

16. What is an advantage of executing a process in background? The most common reason to put a process in the background is to allow you to do something else interactively without waiting for the process to complete. At the end of the command you add the special background symbol, & This symbol tells your shell to execute the given command in the background. Example: Cp *. *. /backup& (cp is for copy)

17. How do you execute one program from within another? The system calls used for low-level process creation are execlp () and execvp (). The execlp call overlays the existing program with the new one, runs that and exits. The original program gets back control only when an error occurs. Execlp (path, file_name, arguments.);//last argument must be NULL A variant of execlp called execvp is used when the number of arguments is not known in advance. Execvp (path, argument_array);//argument array should be terminated by NULL

18. What is IPC? What are the various schemes available?

The term IPC (Inter-Process Communication) describes various ways by which different process running on some operating system communicate between each other. Various schemes available are as follows:

Pipes:

One-way communication scheme through which different process can communicate. The problem is that the two processes should have a common ancestor (parent-child relationship). However this problem was fixed with the introduction of named-pipes (FIFO).

Message Queues:

Message queues can be used between related and unrelated processes running on a machine.

Shared Memory:

This is the fastest of all IPC schemes. The memory to be shared is mapped into the address space of the processes (that are sharing). The speed achieved is attributed to the fact that there is no kernel involvement. But this scheme needs synchronization.

Various forms of synchronisation are mutexes, condition-variables, read-write locks, record-locks, and semaphores.