System Programming and Operating Systems Lab

ASSIGNMENT 13

Name: Shrirang Mhalgi
Roll No: 322008
Batch: B 1

1 Date of Completion:

29/03/2019

2 Aim:

Implement Unix system calls like ps, fork, join, exec family, and wait for process managment(use shell script/java/c programming).

3 Objectives:

To implement Unixs commands using java/c programming/shell script.

4 Theory:

Processes are a very important piece in the UNIX world. Basically, almost every program that you execute is running in a process. A process is defined as instance of a program that is currently running

A unix processor system or single core system can still execute multiple processes giving the appearance of a multi-core machine. Whenever you issue a command in Unix, it creates, or starts, a new process. When you tried out the ls command to list the directory contents, you started a process. A process, in simple terms, is an instance of a running program

The operating system tracks processes through a five-digit ID number known as the pid or the process ID. Each process in the system has a unique pid.

Pids eventually repeat because all the possible numbers are used up and the next pid rolls or starts over. At any point of time, no two processes with the same pid exist in the system because it is the pid that Unix uses to track each process.

When you start a process (run a command), there are two ways you can run it Foreground Processes

Background Processes

1.Foreground Processes: By default, every process that you start runs in the foreground. It gets its input from the keyboard and sends its output to the screen.

2.Background Processes: A background process runs without being connected to your keyboard. If the background process requires any keyboard input, it waits. The advantage of running a process in the background is that you can run other commands; you do not have to wait until it completes to start another!

Commands in Unix:

1.PS command:

The ps command can be used to list only the processes you own, or all of the active processes on a system. The syntax for the ps command is:

$ps[options]

Running ps without any options will display the status of the active processes you own: $ps

PID TTY TIME CMD 7505 pts/3 00:00:00 ksh 8078 pts/3 00:00:00 ps

This output displays four pieces of information for each process. The process' s ID (PID), the terminal controlling the process (TTY), how long the process has been running (TIME), and the command or program the process is running (CMD).

Option Description

-a : Displays all processes on a terminal, with the exception of group leaders.

-c : Displays scheduler data.

-d : Displays all processes with the exception of session leaders.

-e : Displays all processes.

-f : Displays a full listing.

-glist: Displays data for the list of group leader IDs.

-j : Displays the process group ID and session ID.

-l : Displays a long listing

-plist: Displays data for the list of process IDs.

-slist: Displays data for the list of session leader IDs.

-tlist: Displays data for the list of terminals.

-ulist: Displays data for the list of usernames.

2.Fork Command:

System call fork() is used to create processes. It takes no arguments and returns a process ID. The purpose of fork() is to create a new process, which becomes the child process of the caller. After a new child process is created, both processes will execute the next instruction following the fork() system call. Therefore, we have to distinguish the parent from the child. This can be done by testing the returned value of fork():

If fork() returns a negative value, the creation of a child process was unsuccessful.
fork() returns a zero to the newly created child process.
fork() returns a positive value, the process ID of the child process, to the parent. The returned process ID is of type pid t defined in sys/types.h.

Normally,the process ID is an integer. Moreover, a process can use function getpid() to retrieve the process ID assigned to this process. Therefore, after the system call to fork(), a simple test can tell which process is the child. Please note that Unix will make an exact copy of the parent's address space and give it to the child. Therefore, the parent and child processes have separate address spaces.

Let us take an example to make the above points clear. This example does not distinguish parent and the child processes.

```
#include   <stdio.h>
#include   <string.h>
#include   <sys/types.h>

#define    MAXCOUNT   200
#define    BUF_SIZE   100

void   main(void)
{
      pid_t   pid;
      int     i;
      char    buf[BUF_SIZE];

      fork();
      pid = getpid();
      for (i = 1; i <= MAXCOUNT; i++) {
            sprintf(buf, "This line is from pid %d, value = %d\n", pid, i);
            write(1, buf, strlen(buf));
      }
}
```

3.Exec command:

Exec is a functionality of an operating system that runs an executable file in the context of an already existing process, replacing the previous executable. This act is also referred to as an over-

lay.It is especially important in Unix-like systems, although other operating systems implement it as well.

1.execl:

Take the path name from the binary execution as its first argument, the rest of argument are command line argument encoding with null.

2.execv:

Takes the path name of a library executable as its first argument and array of argument consist of second argument.

Like all of the exec functions, execv replaces the calling process image with a new process image. This has the effect of running a new progam with the process ID of the calling process. Note that a new process is not started; the new process image simply overlays the original process image. The execv function is most commonly used to overlay a process image that has been created by a call to the fork function.

4.wait command:
All of these system calls are used to wait for state changes in a child of the calling process, and obtain information about the child whose state has changed. A state change is considered to be: the child terminated; the child was stopped by a signal; or the child was resumed by a signal. In the case of a terminated child, performing a wait allows the system to release the resources asso- ciated with the child; if a wait is not performed, then terminated the child remains in a "zombie" state.
wait() and waitpid():

The wait() :
system call suspends execution of the current process until one of its children terminates. The call wait is equivalent to:

The waitpid() :
system call suspends execution of the current process until a child specified by pid argument has changed state. By default, waitpid() waits only for terminated children, but this behaviour is modifiable via the options argument, as described below.

5.Join command:

join - join lines of two files on a common field. It is similar to join operator used in database.

Syntax: join [OPTION]... FILE1 FILE2
The join command takes as input two text files and a number of options. If no command-line argument is given, this command looks for a pair of lines from the two files having the same first field (a sequence of characters that are different from space), and outputs a line composed of the first field followed by the rest of the two lines.
The program arguments specify which character to be used in place of space to separate the

fields of the line, which field to use when looking for matching lines, and whether to output lines that do not match. The output can be stored to another file rather than printing using redirection.s

5 Conclusion:

In this assignment we learn in detail the concept of Unix System calls .