

Assignment 2 :

Process Control System Calls :

→ Aim :

Process Control system calls to demonstrate, FORK, ~~EX~~ EXEC and WAIT system call along with zombie & orphan states.

1. Implement a C program that accepts integers to be sorted. Main program uses the FORK system call to create a new process called a child process. Parent process sorts the integer using sorting algorithm & waits for child process using WAIT system call to sort the integers using any sorting algorithm. Also demonstrate zombie & orphan processes.
2. Implement a C program to accept an integer array. Main program uses the FORK system to create a new process called a child process. Parent process sorts an integer array & passes the sorted array to child process using ~~EXEC~~ EXEC system call. The child process then uses this sorted array to perform binary search to find a particular item in the array.

→ Objective :

This assignment covers the UNIX system process control, commonly called for process creation, execution & termination. It also covers process model - zombie and orphan processes.

→ Theory :

A. Process :

A process is the basic active entity in most operating system models.

B. Process IDs :

Each process in a Linux system is identified by its unique process ID, sometimes referred to as pid. Process IDs are 16-bit numbers that are assigned sequentially by Linux as new processes are created.

C. Creating Processes :

1. Using system() function :

It provides an easy way to execute a command from within a program. The system creates a subprocess running

the standard Bourne shell and hands the command to that shell for execution.

2. Using fork() system call :

A process can be created by calling fork. The calling process becomes the parent, and the created process is called the child process. The fork function copies the parent's memory image so that the new process receives the copy of address space of the parent. Both ~~instructions~~ processes continue at the instruction after the fork statement.

Syntax: `pid_t fork(void);`

The wait function

When a function creates a child both parent & child proceed with execution from the point of the fork. The parent can execute the wait block until the child finishes. The wait function causes the caller to suspend execution until a child's status becomes available or until the caller receives a signal.

Syntax: `pid_t wait(int* status)`

Status Values

The status argument of wait is not NULL then the function stores return status of the child in this location.

A zero return value indicates EXIT_SUCCESS any other value indicates EXIT_FAILURE.

POSIX specifies six macros for testing the child's return status. Each takes the status value returned by a child to wait as a parameter.

Ex. WIFEXITED(int statval)
WEXITSTATUS(int statval)

The exec Function

The fork function creates a copy of the calling process, but many applications require the child process to execute the code that is different from that of the parent. The exec family of functions provides a facility for overlaying the process image of the calling process with a new image. The traditional way to use the fork-exec combination is for the child to execute the new program while the parent continues to

execute the original code.

1. exec() & execle() :

exec() : Permits us to pass a list of arguments to the program to be executed.

execle() : Does the same JOB except it will use environment variable PATH to determine which executable to process.

2. execv() & execvp() :

execv() : It works same as exec() except that command line argument can be passed to it in the form of array of strings.

execvp() : Does the same except it will use environment variable PATH to determine which executable to process.

3. execve() : It executes the program pointed to by the filename. filename must be either a binary file or a script starting with a line of the form: arg0 is the array of argument list.

Process Termination :

Normally a process terminates when a program calls `exit()` function or the main function returns. The exit code is the argument passed to the `exit()` or the value returned by main.

Zombie Process :

If a child process terminates while its parent is calling `wait` function, the child process vanishes and its termination status is passed to the parent via `wait` call. A zombie process is a process that has terminated but has not been cleaned up yet. It is the responsibility of the parent to clean up its zombie children. If the child process finishes before the parent calls `wait`, the child process becomes zombie.

Orphan Process.

As soon as the parents of any process are dead, re-parenting occurs. Re-parenting means the orphaned processes are immediately adopted by special `init` process.

A process can be orphaned either intentionally or unintentionally. Sometime a parent process exits/terminates or crashes leaving the child process still ~~so~~ running, and then they become orphans. Also a process can be intentionally orphaned just to keep it running.

• Killing an orphan process.

Use the kill command to kill any process - including orphaned process.

Daemon Process :

• It is a process that runs in the background, rather than under the direct control of user, they are usually initiated as background process.

→ Conclusion :

Topics covered :

1. Process Creation
 2. Process Termination
 3. Zombie Process
 4. Orphan Process.
-

Q

Q