33231

CLASSMATE
Date :
Page :

ASSIGNMENT 2

Title : Process control system calls

Problem Statement : The demonstration of FORK, EXECVE and WAIT system calls along with zombie and orphan states.

a) Implement the C program in which main program accepts the integers to be sorted. Main program uses the FORK system call to create a new process called child process. Parent process sorts the integers using sorting algorithm and waits for child process using WAIT system call to sort the integers using any sorting algorithm. Also demonstrate zombie and orphan states.

b) Implement the C program in which main program accepts an integer array. Main program uses the FORK system call to create a new process called a child process. Parent process sorts an integer ~~using~~ array and passes the sorted array to child process through the command line arguments of EXECVE system call to load new program that uses this sorted array for performing the binary search to search the particular item in the array.

Theory :-

Fork Call - used for creating a new process, which is called child process, which runs concurrently with the process that makes the fork() call (parent process).

After a new child process is created, both processes will execute the next instruction following the fork() system call. A child process uses the same pc (program counter), same registers, same open files which are used in the parent process.

It takes no parameters and returns an integer value.

Below are the different values returned by fork()
- Negative value - creation of child process was unsuccessful
- Zero - returned to the newly created child process
- Positive value - Returned to parent or caller. The value contains process ID of newly created child process.

EXECVE call : executes the program pointed to by filename. Filename must be either a binary executable or a script with a line of the form "#! interpreter [arg]"
On success, execve() does not return, on error -1 is returned and errno is set appropriately.

WAIT call : blocks the calling process until one of its child processes exits or a signal is recieved. After child process terminates, parent continues its execution after wait system call instruction.

Child process may terminate due to any of these :
- It calls exit();
- It returns (an int) from main
- It recieves a signal (from the OS or another process) whose default action is to terminate.

If only one child process is terminated then return a wait() returns process ID of the terminated child process.

If more than one child processes are terminated than wait() reap any arbitary child and returns a process ID of that child process. When wait() returns they also

33231

CLASSMATE
Date :
Page :

Algorithm :

Part 1

1) Accept the integer array to be sorted
2) Call the fork function
3) In child process call the sorting algorithm function (Quick Sort)
4) In parent process call another sorting algorithm (Merge Sort)
5) Using WAIT call, wait for child process to complete its sorting
6) Print the result of sorting from both parent process and child process
7) Exit

Part 2

1) Accept the size and content of array which has numbers to be sorted
2) Call the fork function
3) Sort the array using Bubble Sort in parent process
4) Pass the sorted array to child process using EXECVE system call command line arguments
5) In child process, load binary search program file using EXECVE system call ~~command line arguments~~ and perform
6) ~~In child pro~~ binary search on this array to search required elements.
6) Print the sorted array and the position of the element if found else print not found.
7) Exit.

Conclusion : I have successfully implemented FORK, EXECVE and WAIT system calls in C along with Zombie and Orphan states.