Lab Assignment 2

Title: Process Management (Process control)

Problem statement:

Write a program using a fork to create a child process.

a) The parent process should sort elements in ascending order and child process should sort elements in descending order.

b) Orphan Process

c) Zombie process

Objectives:

- To understand the concept of Process, Process Image
- To understand System calls
- To use system call to create a new process
- To understand orphan and zombie processes

Theory:

A **process** is an instance of a running program. A process is said to be born when the program starts execution, it remains alive as long as the program is active. After the execution is complete, the process is said to die or terminate. Program and process are two different entities, when two users run the same program; there is only one program on disk but two processes in memory.

Kernel is responsible for the management of the processes. It determines the time and priorities that are allocated to processes so that multiple processes are able to share CPU resources. It provides a mechanism by which a process is able to execute for a finite period of time and then relinquish control to another process. Kernel allocates space memory not only for process image but also for control information that enables it to switch the processes.

Every process has the various attributes associated with it, and they are maintained in memory by the kernel, in a separate structure called process table. Some important attributes are:

1) Process_id (PID) : Each process is identified by a unique integer called Process_id, that is allocated by the kernel when the process is born. We need PID to control the process, e.g. to kill the process.
2) Parent_id (PPID) : The PID of the parent is also available as a process attribute. It is also required to control the process. E.g. if several processes have the same PPID, we just kill the parent, rather than to kill children separately.
3) State of Process: whether running, ready, zombie, waiting etc.

System Call:

A routine defined in the kernel which performs the basic operations of the computers, like opening a file and creating a process. All UNIX commands and library functions are written in terms of system calls. Processor switches from user mode to kernel mode when executing a system call.

Mechanism to create a process: Creation of a new process, makes use of three important system calls or functions – fork, exec and wait.

Fork: Fork system call creates a copy of the process that invokes it. The process image is identical to that of the calling process, except for a few parameters like PID. When a new process is created, it gets a new PID. PIDs, 0,1,2,and 3 are reserved and created by the kernel only. After fork, parent and child have different PID and PPIDs. At this point there are two processes (parent and child), with practically identical constituents. They continue execution at the statement following fork (code before fork ignored by child). When a fork is invoked, the kernel replicates the address space of the current process (its text, data, stack etc.) It also creates a separate entry in the process table containing several fields copied from the entry of the parent. This includes the file descriptors, current directory etc. Because a child runs into its own address space, changes made to these parameters don't affect the parent. The size of the process table places a restriction on the total number of processes that a machine can support. If an attempt to fork a process violates this restriction, fork returns -1.

Exec: It is a system call that gives a call to a brand new existing executable program. eg: Program A (parent process) creates a child process B, and child process B gives a call to program C through exec. Thus program C overlays the address space of child process B. Thus, exec never returns any value.

Wait: The parent executes a wait system call to wait for the child process to complete. Kernel clears the slot in the process table that was allocated to the child. Parent then picks up the exit status of the child and continues with statements following the wait. A pointer to integer is passed to the wait system call, it not only contains the exit status but other things such as process state. Exit status is stored in least eight significant bits and WEXITSTATUS is the macro used to fetch the exit status.

Algorithm:

i) Read the number of elements n.
ii) Create a child process using fork, which returns zero to the child process and some non zero positive integer to parent.
iii) In the child process print all the odd numbers from 1 to n.
iv) In the child process use exit to terminate the child.
v) In the parent process print all the even numbers from 1 to n.
vi) In the parent process, use a wait system call to collect the exit status of the child.
vii) Terminate parent

Input : Number of elements n

Output : Even numbers and odd numbers from 1 to n.

Conclusion: Thus we have studied the concept of process control, orphan and zombie processes and different system calls

FAQs

1. Explain orphan process concept. Who takes care of the orphan process and how?
2. Explain the zombie process with a diagram.
3. Explain sleep() function.