Jaypee Institute of Information Technology, Noida

Operating System and System Programming lab [15B17CI472]

Assignment- 3

Instruction:

- For executing following program you can use online C compiler in case you don't have Linux operating system installed. Please make sure it has to be a GDB compiler e.g https://www.onlinegdb.com/
- Commands for Process Management

```
top       # view top consumers of memory and CPU (press 1 to see per-CPU statistics)
who       # Shows who is logged into system
w         # Shows which users are logged into system and what they are doing
ps        # Shows processes running by user
ps -e     # Shows all processes on system; try also '-a' and '-x' arguments
ps aux | grep<user_name> # Shows all processes of one user
ps ax --tree      # Shows the child-parent hierarchy of all processes
ps -o %t -p <pid> # Shows how long a particular process was running.
          # (E.g. 6-04:30:50 means 6 days 4 hours ...)
Ctrl z <enter>    # Suspend (put to sleep) a process
fg          # Resume (wake up) a suspended process and brings it into foreground
bg          # Resume (wake up) a suspended process but keeps it running
            # in the background.
Ctrl c      # Kills the process that is currently running in the foreground
kill <process-ID> # Kills a specific process
kill -9 <process-ID> # NOTICE: "kill -9" is a very violent approach.
           # It does not give the process any time to perform cleanup procedures.
kill -l     # List all of the signals that can be sent to a proccess
kill -s SIGSTOP <process-ID> # Suspend (put to sleep) a specific process
```

1. Write Linux C Program to display the process attributed as follows:

   a. The process ID or PID
   b. The parent process ID or PPID
   c. Nice number
   d. Terminal or TTY
   e. User name of the real and effective user (RUID and EUID).

   Sort the output of the above into files called *procatt0.txt*and *procatt1.txt* depending on process creation time and CPU utilization time.

2. Write Linux C Program to create the child process using *fork* command. Display the output in following format

```
before fork : 10539
I am the father: 10539 of
child: 10540
I am  the child: 10540
```

Note: `10539 and 10540 are parent ID and Child ID`

3. Write a Program to create a Zombie process.

4. Write a program where you call fork() system call thrice. Find out how many process are created.

5. Solve the arithmetic of ADD, SUB, MUL and DIV by forking child process for each. Make sure your parent is waiting for all child process to complete their task.

6. Write Linux C Program illustrating inter-process communication using a *pipe* to read and write for at least two pair of child process.

HINT:

- Inter-process communication in Unix may be accomplished at a low level using pipes. A Unix pipe is a circular buffer maintained by the operating system. Typically, one or more processes write data into the buffer; another process reads the data from the pipe.
- While Unix handles the internals of pipe communication, programs communicate through pipes at a slightly more conceptual level. More specifically, the use of pipes involves four main steps

1. An array of two integers is declared.
2. The array is initialized by the pipe procedure. With this initialization, the first array element (subscript 0) provides an appropriate file descriptor for reading, while the second array element (subscript 1) provides an appropriate file descriptor for writing.
3. After the fork() operation, both the reading and writing ends of the pipe are available to both processes. Traditionally, however, pipes are available for one-way communication only. Thus, the reading end of the pipe should be closed off in the writing process, and the writing end of the pipe should be closed off in the reading process.
4. Data are moved byte-by-byte through a pipe, using read and write system calls.
    - The write call requires three parameters, the output file number, a base address (e.g., a character string or an array of characters), and the number of characters to be written.
    - The read call requires three parameters, the input file number, a base address (e.g., a character string), and a number of bytes. Reading retrieves the next number of byes -up to the size of the buffer.