LAB 2(8- 14 March 2021)

Process and IPC

1. Write a program where the parent process writes in the pipe and child process reads from the pipe continuously. To send data to the child the parent takes it from the user. When parent process receives the string "exit" from the user, it sends "exit" on the pipe, closes the write end of the pipe and exits. Upon receiving "exit", the client (child) terminates.

HINT:

- Interprocess 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.

2. 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.

3. Write a program which goes to sleep and is woken up by its child.

HINT: There are 2 ways to solve this problem.

- Wait() – this has some limitations, have a look at them.
    - Ref: http://linux.die.net/man/3/wait
- Using signals –pause()
    - Ref: http://linux.die.net/man/7/signal
    - Ref: http://linux.die.net/man/2/pause
    - Ref: http://www.thegeekstuff.com/2012/03/linux-signals-fundamentals/
    - Ref: http://www.thegeekstuff.com/2011/02/send-signal-to-process/

4. Spawn a child process running a new program. PROGRAM is the name of the program to run; the path will be searched for this program. ARG_LIST is a NULL-terminated list of character strings to be passed as the program's argument list. Return the process ID of the spawned process.

HINT: The exec functions replace the program running in a process with another program. When a program calls an exec function, that process immediately ceases executing that program and begins executing a new program from the beginning, assuming that the exec call doesn't encounter an error. Functions that contain the letter *p* in their names (execvp and execlp) accept a program name and search for a program by that name in the current execution path; Functions that contain the letter v in their names (execv, execvp, and execve)accept the argument list for the new program as a NULL-terminated array of pointers to strings.

5. Write a program which acts like a timer. It asks the user for a specific time, allows the user to execute other programs while it waits and popsup with an alert on finishing the time.

HINT: Look at how the sleep() and wait() commands work. (Can also be done using other IPC techniques e.g. signals)
Ref: http://linux.die.net/man/3/sleep

Process Scheduling Algorithms

1. Write C program in Linux operating systems for implementing the first come first serve scheduling algorithm.

   HINT: Read the no. of process from user, Read burst time for all the processes; apply FCFS Scheduling, print PID, Burst time and Gantt chart
2. Write a C program for implementing the priority scheduling using Linux operating system.

   HINT: Read no. of process from user, Read burst time and priority for all processes. Apply Priority Scheduling algorithm, Print PID, Burst Time, Priority of process and Gantt chart.
3. Write a Linux C Program for the Implementation of Shortest Job First Scheduling Algorithm.

   HINT: Read the number of process from user, Read the burst time for all process, Print: Process, Waiting time, and Turn-around time, also print the average waiting time and the average turnaround time in sec.
4. Write a Linux C Program for the Implementation of Round Robin Scheduling Algorithm.

   HINT: Create few processes. Read the process name and apply burst time of 20 ns for all the process, apply Round Robin scheduling, print: Process name, Remaining Time, Total consumed Time.

5. For the above scheduling algorithm, write menu driven Linux c program and give average waiting time and the average turnaround time in sec.