Programming Tools - II

LAB ASSIGNMENT for Week # 9-10

Q.1. Use the following *Hare* and *Turtle* process descriptions to implement IPC solutions by defining cooperating processes –

```
Turtle()
{
      while (not finish)
      {
      moveleg1();
      moveleg2();
      moveleg3();
      moveleg4();
      }
}

Hare()
{
   while (Turtle is far behind)
       Sleep(for_a_while);
   RunLikeCrazy_A_bit();
}

God()
{
  wait for keyboard;
  choose hare or turtle from keyboard;
  reposition hare or turtle;
}

Report()
{
   while (game not over)
     report positions of hare and turtle;
     /* use ncurses for graphics if you are ambitious, printf otherwise */
}

Main()
{
  fork {
     Turtle();
     Hare();
     God();
     Report();
  }
  loop until somebody wins;
}
```

a) First use fork() and execve() to create the processes and develop complete applications using the following LINUX IPC (message passing) mechanisms separately:

(i) *pipes* and (ii) *FIFO files*.

b) Develop the same applications using Linux *pthreads* (shared memory). Notice that you have shared memory between threads so you do not need explicit communication channels. You can communicate using shared variables. However, you will need to ensure that different threads synchronize their access to shared variables to avoid inconsistencies. In particular, you should use *mutexes* to protect all accesses to shared variables.

Note: 1) The current outcome of the race should be displayed after fixed intervals.

2) Try varying the speed and the sleeping time of the hare to see different outcomes.

3) Compare the performance (running time) of shared memory communication versus pipes by transferring a large amount of data between two processes using these two mechanisms. Report your results. Optionally compare the performance of other IPC mechanisms too.

Q.2. Implement the following problem using UNIX *pipes* and *signals* –

There are *N* players (numbered *0* to *N-1*) that sit in a circle and play the following game. The first person (player 0) starts with a token of initial value *Tok_value*. If the value of the token is not 0 it decrements the value of token by one and passes it to the next person (player 1). The receiver (player 1) checks if the token is 0 or not, if not the receiver just decrements the value of the token by one and passes it to the next person (player 2) and so-on. Otherwise, if the value of token was zero the receiver collects a point and passes a new token with initial value of *Tok_value* to the next player. The first player to collect *Pt* points wins and then kills all the others.

For example: If *N = 9, Tok_value = 4* and *Pt = 2*, the players are numbered 0 to 8. Player 4 gets the first point then player 0 gains a point and so on. In this example players gain points in the following order *4, 0, 5, 1, 6, 2, 7, 3, 8, 4* and 4 wins because it is the first one to get *Pt=2* points.

Write a program to simulate the above game. You program should get the inputs *N*, *Tok_value* and *Pt* from the user. It then spawns *N* processes and then links the last process to the first. The linking is being done through unnamed pipes (i.e. there should be a pipe between two consecutive persons). The i^{th} process represents the i^{th} person in the circle. Note that the first process has to generate two pipes, one to communicate with the second process and one with the last process. The token is passed using UNIX unnamed pipes. The winner kills the losers by sending the signal SIGTERM along the circle. (Refer to *kill()* system call for this). Before each loser is killed, he has to close the pipes which he uses (i.e., you have to write a signal handler for SIGTERM).

You must output the progress of the game. E.g. when a receiver gets a token it prints something like the following: -
I am player number <X>. I have received a token with value <Y> and I am now sending token to player <Z>. My current points are <W>.

If a player wins it prints:
I am player number <X>. I have <Y> points. I have won!!!

Each player that is killed prints before exiting:
I am player number <X>. I have been killed.

Last Date and Time of Submission: 21/04/2016, 02:00 PM