Operating Systems - Assignment 3

Topic : IPC
Total Marks : 100
Submission Date : 24th September, 11:59pm

General Instructions :

1.There will be no extension of deadline what-so-ever, unless some critical situation arises(like alien invasion!)

2.There will be zero-tolerance for copy cases. A list of students indulged in copying(from net/friends/seniors) will be sent and they will not be called for evaluation.

3. Upload format :
a.)Make folder of your roll no : 2015XXXXX.
b.)Inside this folder make 4 folders(one for each question) : Q1, Q2, Q3, Q4
c.)Inside each of these folders put your one or more .c files of respective questions.
d.)Create 2015XXXX_Assignment3.tar.gz and upload on courses portal.

Question 1 : Fork [Marks : 20]

Consider a 5*3 array -

```
int arr[5][3] = {
                        {1, 2, 3},
                        {4, 5, 6},
                        {7, 8, 9},
                        {10, 11, 12},
                        {13, 14, 15}
                };
```

It has 5 rows. You need to find summation of all the numbers in a given row.
However for doing so, you will create 5 children using fork().

Each child will find the summation of a row & print 3 things -
a. its own PID

b. the row number
c. the summation of the row.

This is the required output -
>./a.out
Child pid : 13488, Sum of row : 0 is : 6
Child pid : 13489, Sum of row : 1 is : 15
Child pid : 13491, Sum of row : 3 is : 33
Child pid : 13490, Sum of row : 2 is : 24
Child pid : 13492, Sum of row : 4 is : 42

NOTE#1 The row number will not necessarily come in order.
NOTE#2 Each child pid has to be unique.

You have to solve this problem for any given matrix of any size.
You need to create row-count number of children, each of them printing every row's sum.

Input Format :
First line contains 2 numbers : row-count & column-count.
This will be followed by row-count number of lines, each containing column-count number of numbers, separated by single space.

Example :
5 3
1 2 3
4 5 6
7 8 9
10 11 12
13 14 15

You can use one or more from the following the only-
fork()
getpid()
exit()
wait()

Question 2: (Single Level Pipe) [Marks : 30]

The problem is to simulate a single level PIPE (" | ") used for two way communication.
The output of one command will be treated as the input for the other.
The expected output should be the exactly as the output on the linux terminal.

TIP : If possible make your implementation extensible i.e. for any number of pipes.(because this will be required in the next assignment)

Input format : ./a.out <entire_command>

Examples :

./a.out "ls -l | wc -l"

./a.out "ls -l /home/loki/Desktop/ | grep "a" "

./a.out "cat /usr/share/dict/words | grep "and" "

Note :

1.You CANNOT use any data-structure or external text file to store the intermediate result of the command1.

2.The <entire_command> will be given in double quotes as a string. Parse it properly to extract two commands on LHS & RHS of the pipe.

Useful System Calls:

exec() family

fork()

pipe()

close()

wait()

waitpid()

dup()

Question 3 : (Shared Memory Access) [Marks : 25]

The problem is to implement a producer consumer user-level application using shared-memory. Producer writes a fibonacci number in shared memory and waits for the consumer to read it and reply back with the next fibonacci number. Then this goes in cycle between both of them.

The starting two numbers of the fibonacci series will be given as input to both producer & consumer.

Example :

Producer process: ./a.out 3 10

Consumer process: ./a.out 5 10

This means the fibonacci series starts from 3 and goes upto next 10 numbers -

3 5 8 13 21 34 55 89 144 233

NOTE : The smaller number will be always given to the producer & the greater number to the consumer.

Sample Run :

Producer process	**Consumer process**
1.)./a.out 3 10	2.) ./a.out 5 10
3.)Received 5 from consumer. Sending 8 to consumer.	4.)Received 8 from producer. Sending 13 to producer.
5.)Received 13 from consumer. Sending 21 to consumer.	6.)Received 21 from producer. Sending 34 to producer.
...	...
9.)Received 89 from consumer. Sending 144 to consumer.	10.)Received 144 from producer. Sending 233 to producer.

NOTE :

a.) If you just see the numbers under producer (or consumer) column, they form a continuous fibonacci series.

b.) The numbers(1.), 2.), 3.), 4.) etc) above indicate the flow, you don't have to print numbers in your output.

Testing :

Part 1 : Run producer & consumer code on two different terminals in same laptop.

Useful System calls :

shmget()

shmat()

wait()

shmdt()

shmctl()

Question 4: (Socket Programming) [Marks : 25]

The problem is to implement a client-server user-level application using socket programming in C. Server accepts strings from clients (even multiple strings from each client) and replies with the sum of the ASCII values of the characters in the string.

Example : When client sends - "!Hello", server replies with "533".

Explanation:

!	=	33
H	=	72
e	=	101
l	=	108
l	=	108
o	=	111

sum = 33+ 72+101+108+108+111 = 533

Both server and client(s) have to output both sending & receiving strings on the terminal. The server and client processes should be run on different machines.

Note: During evaluation, you will be asked to setup client processes on different machines and show the outputs.

Sample Run :

NOTE : The numbers below indicate the flow, you don't have to print numbers in your output. Both the client & server run indefinitely until killed by <Ctrl-C>.

Client Side	**Server Side**
2.)./a.out<server_IP_addr> <server_portno>	1)./a.out
3.)Type string to send : !Hello	4.)String received : !Hello
6.)Sum received : 533	5.)Sum sent : 533

7.)Type string to send : IIIT-H	8.) < and the cycle continues >

Useful system calls:

sock()

bind()

sendto()

recvfrom()

send()

recv()

listen()

connect()