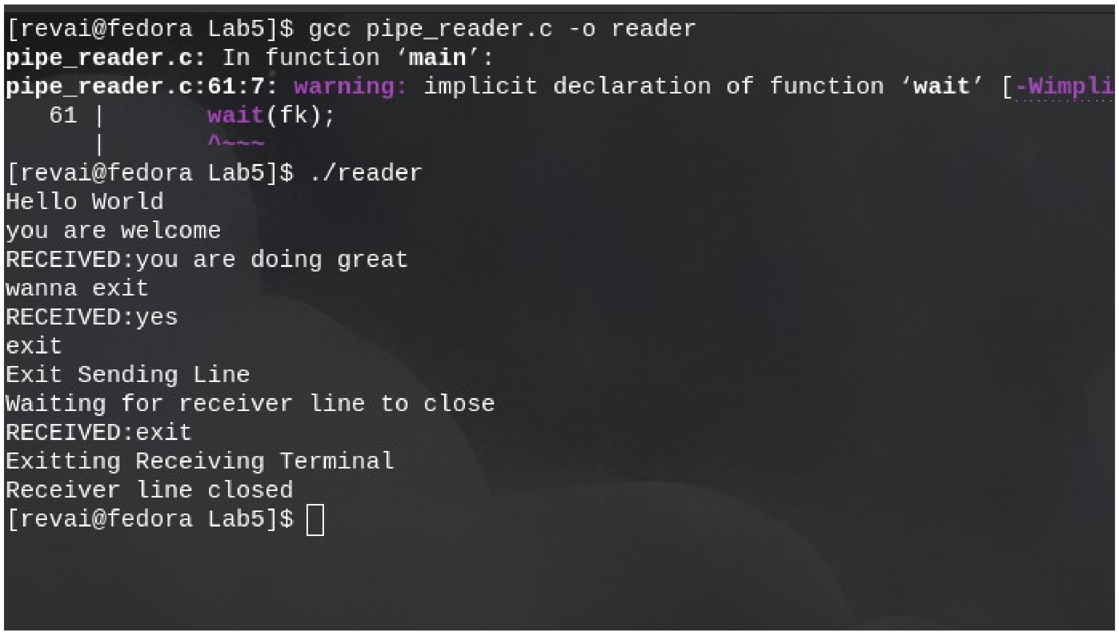
Lab 5

Operating System

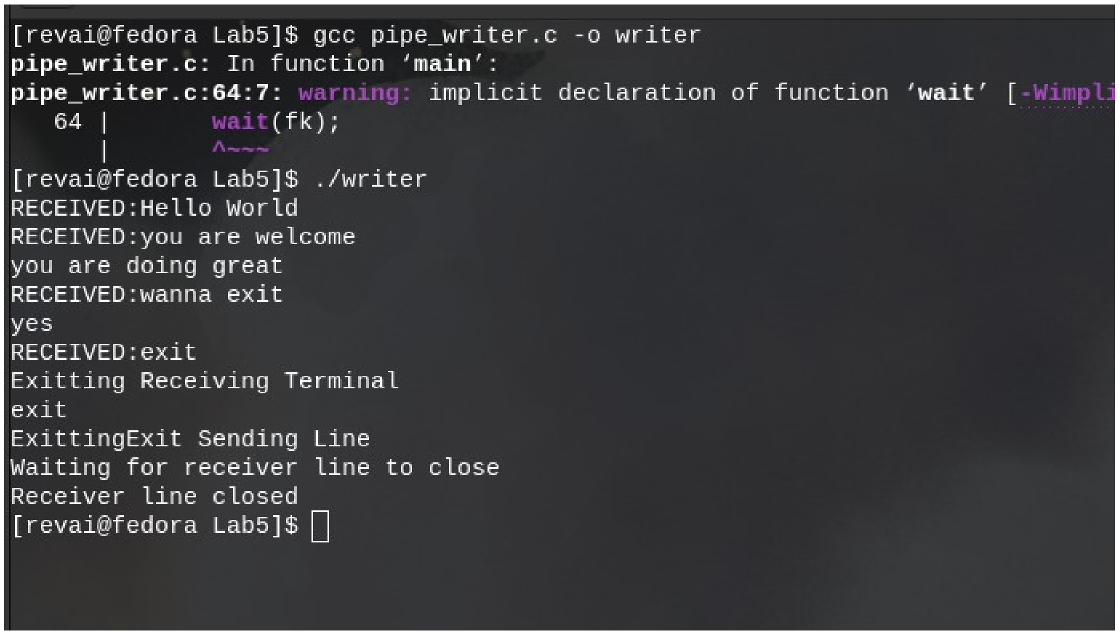
Name: Prakhar Gupta
Roll No: 2103126
Department: CS
Instructor: Dr. Shitala Prasad

Part B:
1)
Reader Program:



```
[revai@fedora Lab5]$ gcc pipe_reader.c -o reader
pipe_reader.c: In function ‘main’:
pipe_reader.c:61:7: warning: implicit declaration of function ‘wait’ [-Wimpli
   61 |       wait(fk);
      |       ^~~~
[revai@fedora Lab5]$ ./reader
Hello World
you are welcome
RECEIVED:you are doing great
wanna exit
RECEIVED:yes
exit
Exit Sending Line
Waiting for receiver line to close
RECEIVED:exit
Exitting Receiving Terminal
Receiver line closed
[revai@fedora Lab5]$ 
```

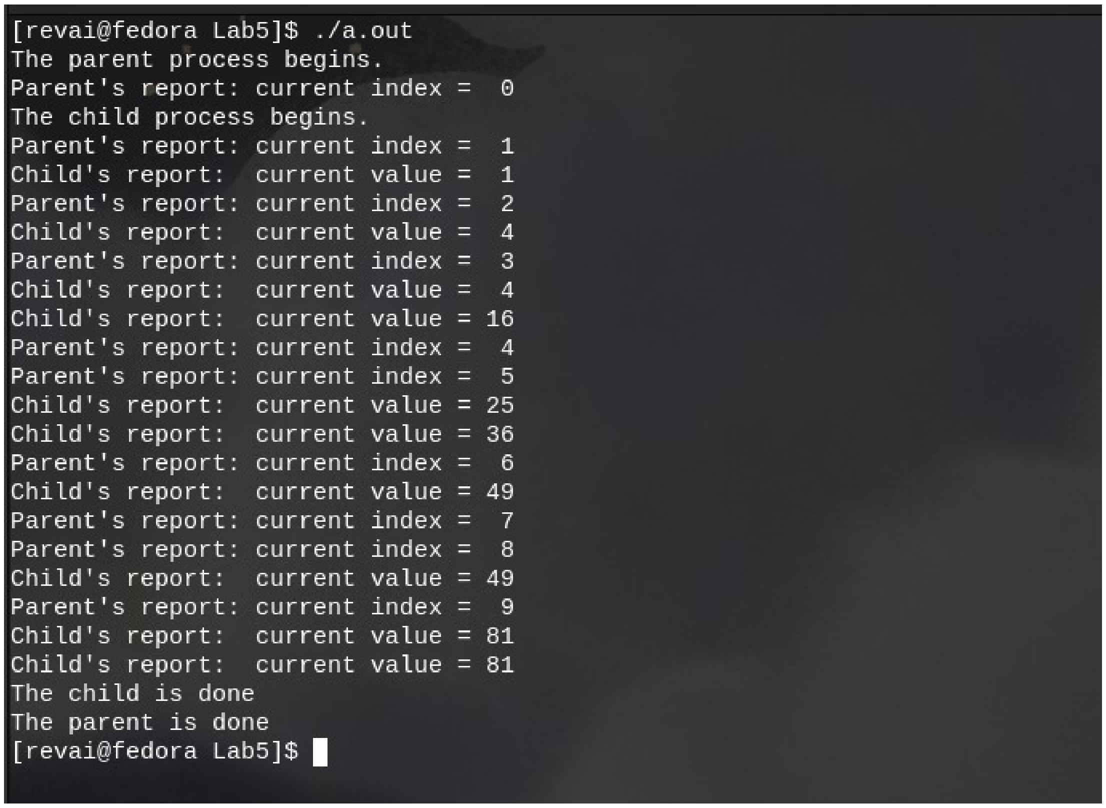
Writer Program:



```
[revai@fedora Lab5]$ gcc pipe_writer.c -o writer
pipe_writer.c: In function ‘main’:
pipe_writer.c:64:7: warning: implicit declaration of function ‘wait’ [-Wimpli
   64 |       wait(fk);
      |       ^~~~
[revai@fedora Lab5]$ ./writer
RECEIVED:Hello World
RECEIVED:you are welcome
you are doing great
RECEIVED:wanna exit
yes
RECEIVED:exit
Exitting Receiving Terminal
exit
ExittingExit Sending Line
Waiting for receiver line to close
Receiver line closed
[revai@fedora Lab5]$ 
```

In our code we created two processes for each reader and writer file. Each process associated with file was responsible for receiving data. We created 2 pipes and pipe works best for unidirectional data flow. So one pipe (fifo) was associated for sending data from reader to writer and one pipe was associated for vise versa.
After typing exit at either end that send line and receiver receiving lines are closed but data can still flow unidirectinally. Therefore for stopping the processses (stopping the whole chat application) we need to type exit at each process.

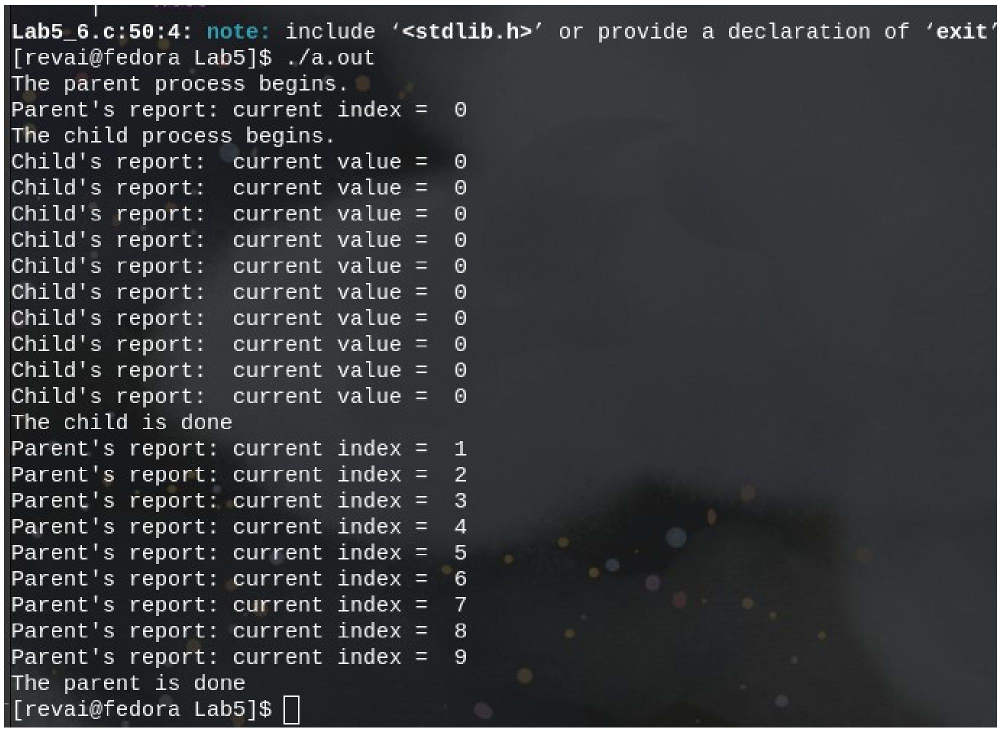
2.)



```
[revai@fedora Lab5]$ ./a.out
The parent process begins.
Parent's report: current index =  0
The child process begins.
Parent's report: current index =  1
Child's report:  current value =  1
Parent's report: current index =  2
Child's report:  current value =  4
Parent's report: current index =  3
Child's report:  current value =  4
Child's report:  current value = 16
Parent's report: current index =  4
Parent's report: current index =  5
Child's report:  current value = 25
Child's report:  current value = 36
Parent's report: current index =  6
Child's report:  current value = 49
Parent's report: current index =  7
Parent's report: current index =  8
Child's report:  current value = 49
Parent's report: current index =  9
Child's report:  current value = 81
Child's report:  current value = 81
The child is done
The parent is done
[revai@fedora Lab5]$ 
```

Here the code between parent process and child process has been synchronized by using sleep system call. First the child process is made to sleep for parent process to first initial shared memory. Then parent process is made to sleep (wait) for child process to use that data and calculate (here square of that number). Then same procedure happens again where child and parent and child process are made to wait one by one so other can either use or change data.

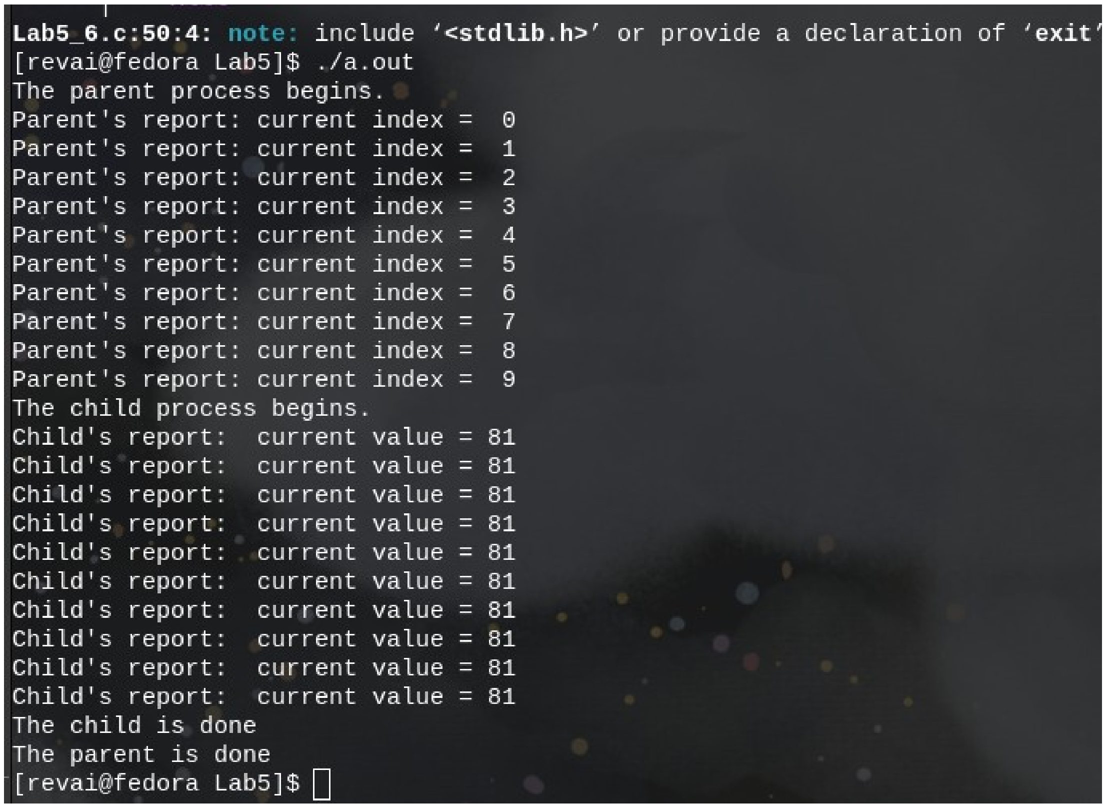
3.)



```
Lab5_6.c:50:4: note: include ‘<stdlib.h>’ or provide a declaration of ‘exit’
[revai@fedora Lab5]$ ./a.out
The parent process begins.
Parent's report: current index =  0
The child process begins.
Child's report:  current value =  0
Child's report:  current value =  0
Child's report:  current value =  0
Child's report:  current value =  0
Child's report:  current value =  0
Child's report:  current value =  0
Child's report:  current value =  0
Child's report:  current value =  0
Child's report:  current value =  0
Child's report:  current value =  0
The child is done
Parent's report: current index =  1
Parent's report: current index =  2
Parent's report: current index =  3
Parent's report: current index =  4
Parent's report: current index =  5
Parent's report: current index =  6
Parent's report: current index =  7
Parent's report: current index =  8
Parent's report: current index =  9
The parent is done
[revai@fedora Lab5]$ 
```

After removing sleep from child process, all the code for child is executed at an instant, and parent did not have time to change the data. Therefore with preinitialised value of shared memory to zero, child process complete all the loops with zero value. Zero square is zero. The parent process took its time in executing its code with sleep in each loop.

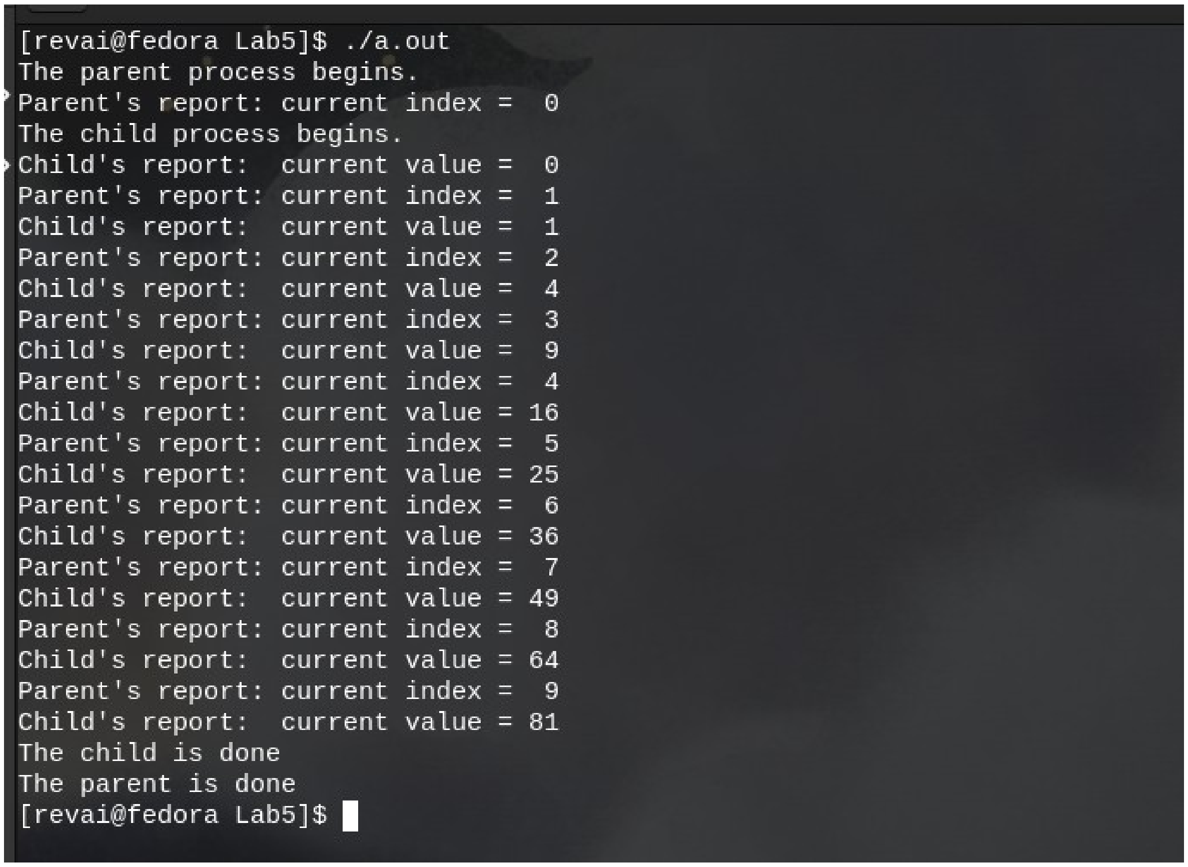
4.)



```
Lab5_6.c:50:4: note: include ‘<stdlib.h>’ or provide a declaration of ‘exit’
[revai@fedora Lab5]$ ./a.out
The parent process begins.
Parent's report: current index =  0
Parent's report: current index =  1
Parent's report: current index =  2
Parent's report: current index =  3
Parent's report: current index =  4
Parent's report: current index =  5
Parent's report: current index =  6
Parent's report: current index =  7
Parent's report: current index =  8
Parent's report: current index =  9
The child process begins.
Child's report:  current value = 81
Child's report:  current value = 81
Child's report:  current value = 81
Child's report:  current value = 81
Child's report:  current value = 81
Child's report:  current value = 81
Child's report:  current value = 81
Child's report:  current value = 81
Child's report:  current value = 81
Child's report:  current value = 81
The child is done
The parent is done
[revai@fedora Lab5]$
```

Here, instead of child being completed at instant as earlier, parent process got executed at an instant. They child process, because of sleep cmd started quite some time after parent process end and use the value of shared memory (which is now 9) to do calculation on it. Now child process is not manipulating shared memory therefore same value (9) is reused again and again.
Also because of wait command parent process waited for child process to terminate before it terminate itself.

5.)



```
[revai@fedora Lab5]$ ./a.out
The parent process begins.
Parent's report: current index =  0
The child process begins.
Child's report:  current value =  0
Parent's report: current index =  1
Child's report:  current value =  1
Parent's report: current index =  2
Child's report:  current value =  4
Parent's report: current index =  3
Child's report:  current value =  9
Parent's report: current index =  4
Child's report:  current value = 16
Parent's report: current index =  5
Child's report:  current value = 25
Parent's report: current index =  6
Child's report:  current value = 36
Parent's report: current index =  7
Child's report:  current value = 49
Parent's report: current index =  8
Child's report:  current value = 64
Parent's report: current index =  9
Child's report:  current value = 81
The child is done
The parent is done
[revai@fedora Lab5]$ 
```

Here we have spinlock to synchronized the code rather than sleep. So unlike 3 and 4 question answer code, our code is synchronized and does not take too much time to execute unlike Lab5_6.c default file (which takes a lot of time to fully execute). This was made possible by creating a new shared memory and using it as a signal for both parent and child process whether original shared memory was ready to be used or be manipulated. After parent process manipulate (change) the value of shared memory we set spinlock value to 1, signally child that memory is ready to be used. After using value child, set spinlock value to -1 signally that value is ready to be manipulated again. This goes on and on till needed.