Objective:

This part of the lab focuses on process manipulation in C.

Activities - Part A

1. Write a recursive C function called `fib(int n)` that calculates the n_{th} fibonacci number. The fibonacci numbers are a sequence given by:

$$x_1 = 1,\ x_2 = 1,\ x_n = x_{n-1} + x_{n-2};\ \text{for } n \geq 3$$

n	0	1	2	3	4	5	6	7	8	9	10	11	12	...
x_n	0	1	1	2	3	5	8	13	21	34	55	89	144	...

2. In your program from 1. write another function called `printFib(int n)` that prints the fibonacci sequence from `1` to `n`. For example,

   If n = 3 the output should be:
   ```
   1 1 2
   ```

   (Tip #1: For information on printing and loops in C, see: <http://www.tenouk.com/cpluscodesnippet/simpleforstatement.html>)

3. Modify your program from 2. to accept `n` as an input parameter from the command line. Your program should perform necessary error checking to ensure that `n` is a positive integer.

   (Tip #2: For information on converting a string to an integer value, see: <https://www.tutorialspoint.com/c_standard_library/c_function_atoi.htm>)
   (Tip #3: For information on accepting input values, see: <http://crasseux.com/books/ctutorial/argc-and-argv.html>)

4. Modify your program from 3. to use the `fork()` system call to generate the fibonacci sequence in a child process. The child process should also print the sequence. Have the parent invoke the `wait()` system call to wait for the child process to complete before exiting the program.

Objective:

This part of the lab focuses on message passing between processes in C using the shared-memory model. Write a complete C program on Ubuntu to demonstrate interprocess communication (IPC) using the shared-memory model. The system calls can be used are `shmget()`, `shmat()`, `shmdt()`, `shmctl()`.

Activities - Part B

Reference program from Lecture Notes. Research the system calls online for more usage information.

```
#include<stdio.h>
#include<sys/shm.h>
#include<sys/stat.h>

int main(void){

   int segment_id; //identifier for shared memory segment
   char *shared_memory; //pointer to shared memory segment
   const int size=4096; //size in bytes of shared memory segment

   //1.allocate a shared memory segment
   segment_id=shmget(IPC_PRIVATE,size,S_IRUSR|S_IWUSR);

   //2.attach the shared memory segment
   shared_memory = (char*)shmat(segment_id, NULL, 0);

   //3.write a message to the shared memory segment
   sprintf(shared_memory, "Hi there!”);

   //3b.now print out the string from shared memory
   printf(“*%s\n”,shared_memory);

   //4. now detach shared memory segment
   shmdt(shared_memory);

   //5.now remove shared memory segment
   shmctl(segment_id, IPC_RMID, NULL);

   return 0;
}
```

1. Write a program called `server.c`
   (a) Create a 1K shared memory segment, identified by the key ‘5678’, with read and write permissions allowed for all processes (IPC_CREAT | 0666).
   (For more information on `shmget`, see: http://man7.org/linux/man-pages/man2/shmget.2.html)
   (b) Attach the shared memory segment to the program using the `shmat(..)` system call.
   (c) Write a message in the shared memory segment and then print the message by

retrieving it from the shared memory segment.

(d) Write code to make `server.c` sleep if the data in the memory segment does not start with a special character e.g. *. If the data has the special character, then the program should exit.

2. Write a program called `client.c`

(a) Locate the shared memory segment identified by the key in server.c using the `shmget` system call.

(b) Attach the program to the shared memory segment using the `shmat` system call.

(c) Read the message in the memory segment and print the message.

(d) Modify the message in the segment by appending the '*' character to the beginning of the message.

(e) Detach from memory segment.

(Tip: You need to run both programs in order to test and proceed with Step 2. (`Ctrl + a, c`). See the manual page for more information - `man screen`)

Extra: Modify `client.c` to accept a message from the command line to write to shared memory.