Q1) Write a program that does the following :

a) It should accept two command line arguments.

(These two are file names. One is the *source file* and the another is the *destination file*) b) It creates a pipe.

c) It creates a child process

d) Now the parent reads the data in the source file, sends it over the pipe. The child reads the data, over the pipe and writes it in the destination file.

Whenver there is no more data to read, the child exits. The destination file becomes a copy of the source file.

(Do error handlings and proper indentation of your source code)

2) Write a program which will concatenate "n" files given as command line arguments***.***

(Say your final executable is myconcat and you have invoked it as "*myconcat f1 f2 f3 f4*". *Here f4 will contain the data of f1,f2 and f3 in that order.)*

1. You need to create a shared memory, of suitable size.
2. You need to create a child process, which will store the contents of the file in that order, in the shared memory.
3. The parent process needs to wait for the completion of the child process. Once the child gets completed, it reads the data from shared memory and writes it to the output file.

(Note the parent needs to know how much data, it is going to read from the shared memory)

3) You are required to develop 3 types of processes

* **Publisher**   
  Publisher process reads the messages from the user input *stdin* and writes them into a message queue. The input by the user can be assumed to be an integer and a string. (The integer can be used as the type of message, corresponding to a message structure applicable for System V Message Queue)<http://man7.org/linux/man-pages/man2/msgrcv.2.html>
* **Dispatcher**  
  Dispatcher process reads the messages from the message queue to which the publisher processes write. It then identifies the message (based on the Type of Message) and invokes the particular subscriber to that message.   
  For example, if the message received is of type **1**, then it dispatches this event to the corresponding subscriber of that message, by sending a signal to that subscriber process (Need to use ***kill( )*** function here <http://pubs.opengroup.org/onlinepubs/009695399/functions/kill.html> )
* **Subscriber**   
  Subscriber is the process which handles a particular message. It registers itself with the dispatcher (by sending its process id to the dispatcher through another message queue). After registration it sleeps (pauses its execution) and will return to receive a message only when it is interrupted by a signal from the Dispatcher Process.

Please note the assumptions given below:

* There can be *any number* of publishers in the system.
* There is only one dispatcher.
* The number of subscribers can be assumed to be fixed (minimum of 3). Therefore the user input to the publisher can be **1**, **2** or **3**.
* The dispatcher when started, first waits for all the (3) subscribers to register with it (it receives the process identifiers of the subscribers through another message queue). Once all the (3) subscribers are registered, it proceeds to read the Publisher message queue and depending on the message it sends a signal to the registered process (You need to decide which signal you are going to use like SIGALARM, SIGUSR1.
* It can be assumed that there is only one subscriber per one type of message.