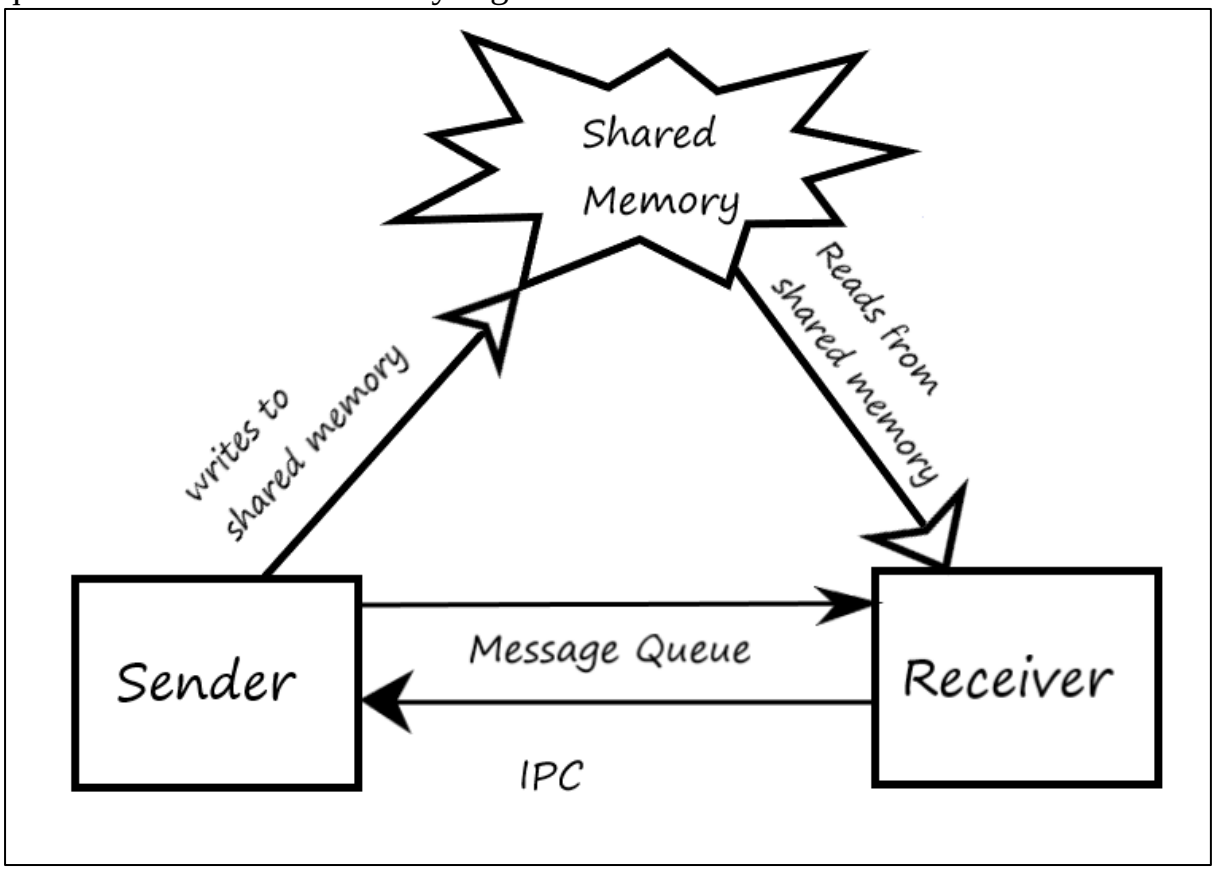
Assignment 1

Basic Idea:

The main idea of this assignment is to design 2 programs that will communicate with each other by shared memory and message queues. This mechanism will synchronously transfer files between two processes. Here we have designed sender and receiver process. Both the program should connect to the message queue and the shared memory segment.



The message queue helps the sender to communicate with the receiver about the size of the message to be fetched from shared memory. The sender process breaks the large message into small chunks and stores it into the shared memory segment. The sender then sends a message to the receiver with the help of a message queue to indicate how many bytes to read from the shared memory segment. As this is a synchronous process the sender waits in the message queue to receive confirmation from receiver about successful transfer of the message. After getting acknowledgment from the receiver, the sender sends messages until there is no message left for transfer. At the end of the transfer, the sender will send a message to the receiver with size 0 and that will be the end of communication between the 2 processes.

Design:

Below is the pseudo code of the 2 processes.

Sender:

// Sets up the shared memory segment and message queue. //

Void **init**{
A file called keyfile.txt containing the message to be transferred is being created.(we are assuming file already created in the specified directory);
Generate key;
Get the id of the shared memory segment;
Attach to the shared memory;
Attach to the message queue;
}

//Clean-up procedure after message transfer
void **cleanUp**
{
Detach from shared memory region;
Exit;
}

//Main send function:
Void **send**{
Open the file for reading;
A buffer to store message we will send to the receiver;
A buffer to store message received from the receiver;
Read at most SHARED_MEMORY_CHUNK_SIZE from the file and store them in shared memory;
Send a message to the receiver telling it that the data is ready;
Wait until the receiver sends a message telling that it has finished saving the memory chunk into recvfile(output file);
After no more data to send, send a message with size field 0;
Exit;
}

Int main{
Call **init** function to connect to shared memory and the message queue;
Call **send** function to Send the file;
Call **Cleanup** function.
}

Receiver:

// Sets up the shared memory segment and message queue. //
void **init**{
//Create keyfile.txt similar with sender file;
Generate key;
Get the id of the shared memory segment;
Attach to the shared memory;
Attach to the message queue;
}

```
//Main receive function
Void mainLoop{

Open output file for writing the message received from sender;(Output file name: recvfile)
Error check;
Receive the message and get the message size.
Keep receiving until the sender set the size to 0 indicating that there is no more data to send.
while (msgSize != 0){
        if(msgSize != 0)
                {
                        Save shared memory to file;
                        Tell sender ready for next file chunk;
                }
        }
}
Close file after done.


//Clean-up procedure after message transfer
void cleanUp
{
Detach from shared memory;
Deallocate the shared memory chunk;
Deallocate the message queue;
Exit;
}


//Exit Signal
void ctrlCSignal{
Call cleanup to free system resource;
}

int main{
//Install Signal handler
Call signal;( Install a signal handler using signaldemo.cpp)
Call init function to connect to shared memory and the message queue;
Call mainLoop  function to receive the file;
Call Cleanup function to Detach from shared memory segment, and deallocate shared memory and
message queue;
Return;
}
```