Design of Assignment1

I: Purpose:

The main objective of Assignment1 is to gain a better understanding of shared memory as well as message queues. Here we have designed a basic message queue. Via the programs sender.cpp and recv.cpp, we are able to send a message between the two programs in sync as opposed to running them in a single program. This is done through utilization of commands from the IPC library, allowing for the creation of the message queue. Using commands from the shm library, allows us to create a segment of shared memory from which the message queue will be able to place data into.

II: Files and Functions:

The program consists of several .cpp/.h files which will be listed below:

recv.cpp: Handles the creation of the shared memory segment as well as the message queue. Recv then waits on standby for either a message to be received via sender.cpp or the signal to cancel execution(CTRL-C). Once a message has been received, recv sends the message to 'recvfile' and communicates with sender via sending messages to signal when it is ready to read in more of the message. The process repeats until the message has been fully read, at which point recv frees the shared memory and destroys the message queue. This program is executed first using the command ./recv.

sender.cpp: Handles attaching to the shared memory segment and then sending the message passed as a parameter to the recv.cpp function via the message queue. Once the whole message has been sent, it will then clean up, detach from the shared memory and close. This program is executed second using the command ./sender [file to be sent].

msg.h: Contains a struct which will handle the information connected to the message to be sent such as message size as well as basic print functions to display the contents of the file.

Other files included:

Makefile: A simple makefile in charge of compiling both sender.cpp and recv.cpp. This will be run initially to allow recv and sender to execute.

SampleExecution.png: Screenshots of sample execution showing test runs of the program along with their outputs. If running differently then expected outcomes, please contact us.

Design of Assignment1.pdf: You are here!

III: Flow of the Program:

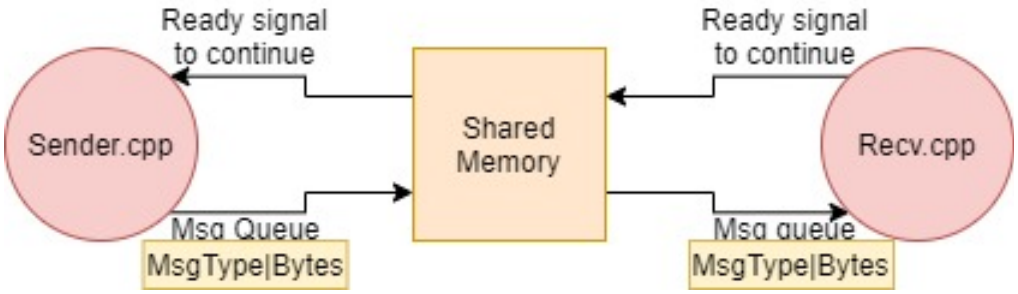

Figure 3.1

Explanation:
Recv waits on standby to receive messages from sender which will send them towards recv via the shared memory segment. Attached with the message will be the number of bytes as well as the message type. Recv will keep processing the message so long as the number of bytes is not 0. When the number of bytes hits 0, recv will signal Sender.cpp that it is ready to receive more of the message. Once the message has been fully sent, “clean up” happens, destroying the message queue and freeing up the memory being used from the shared memory segment.

Structured English:

```
Recv.cpp:
{
        Set up shared memory segment;
        Set up message queue;



        while( SizeofMessage != 0)
        {
                Wait until message received from sender;

                Read in number of bytes saved to shared memory;

                Save the bytes from shared memory to recvfile;

                Signal sender to continue sending message;

        }
```

```
        Detach shared memory;

        Destroy message queue;

        //this will be done via the cleanup functions
        //NOTE: Calling cleanup twice leads to errors, so it is exclusively called in sender
        //unless CTRL-C is used to end the program early.

        exit;

}


Sender.cpp
{
        Attach to shared memory segment to allow communication with recv;

        Connect to message queue;


        while(msgSize != 0)
        {
                Transfer message from file passed in parameter to shared memory;

                Send message from shared memory to receiver;

                Wait for signal from receiver to continue processing message;

        }



        Signal receiver to finish its while loop;

        Display received message

        Call cleanup functions

        Detach shared memory;

        Destroy message queue;

        Exit;
```

```
}
```