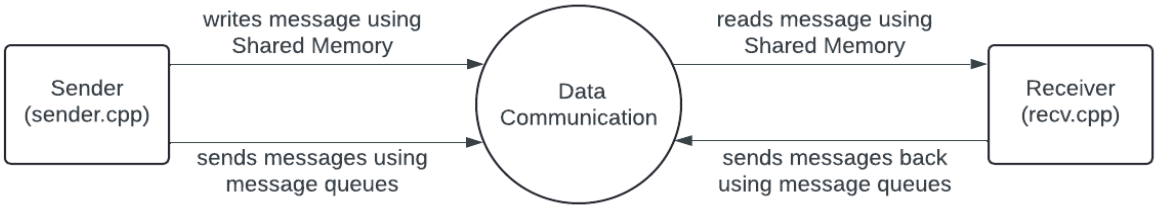
Design of Sender and Receiver

Anthony Seng, Sagarkumar Patel, Edmarck Sosa Pineda, Afnan Al

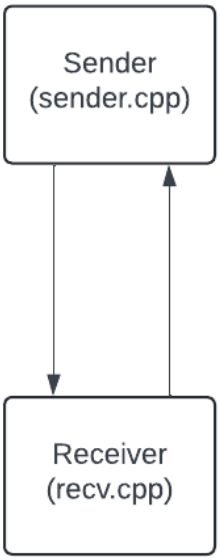
Explanation of Sender and Receiver:

The program's main purpose is to depict shared memory and message queues between Sender (sender.cpp) and Receiver (recv.cpp). Shared memory enables information communication between processes by reading/writing data to/from a shared memory region. Message queue allows information communication between processes by exchanging messages utilizing a queue.

Data Flow Diagram (Level 1)



Structure Chart



Pseudo Code

recv.cpp

```
String recvFilename()
{
//Gets file name from sender
//Declare fileNameMsg
//Recieve the file
//returns the received filename
}

Void init(int& shmid, int& msqid, void*& sharedMemPtr)
{
//Create a key file
//Allocate s shared memory
//Attach to the shared memory
// Create a message queue
//Store the ID and pointer to shared memory
}

unsigned long mainloop(const char* filename)
{
//This function is for opening the received file and reading it
}

Void cleanUp(const int& shmid, const int& msqid, void* sharedMemPtr)
{
//Detach from shared memory
//deallocate the shared memory
//deallocate the message queue
}

Void ctrlCSignal(int signal)
{
//Free resources
}

Int main(int argc, char** argv_
{
//install a signal handler
```

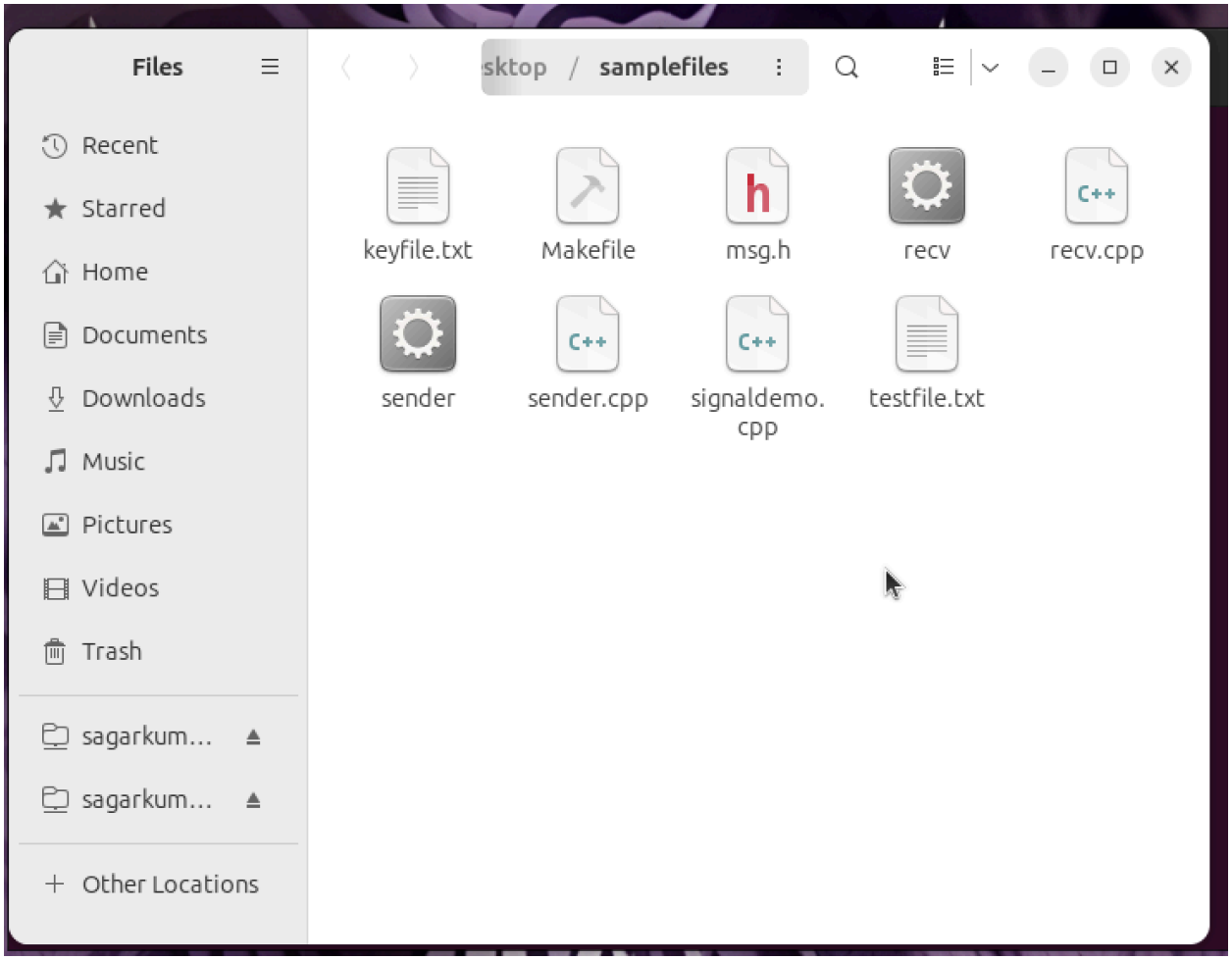
```
//initalization
//receive the file name
//go to main loop
//call cleanup
}
```

sender.cpp

```
void init (int& shmid, int& msqid, void*& sharedMemPtr)
{
//create file called keyfile.txt containing string "hello world"
//generate the key using ftok
//get id of the shared memory segment
//attach to the shared memory
//Get id of the message queue
}
Void cleanUp(const int& shmid, const int& msqid, void* sharedMemPtr)
{
//Detach from shared memory
//deallocate the shared memory
//deallocate the message queue
}
unsigned long sendFile(const char* fileName)
{
//function for sending the file
}
void sendFileName(const char* fileName)
{
//sends the filename
}
int main(int argc, char** argv)
{
//check the command line arguments
//connect to shared memory and the message queue
//send the name of the file
//send the file
//cleanup
}
```

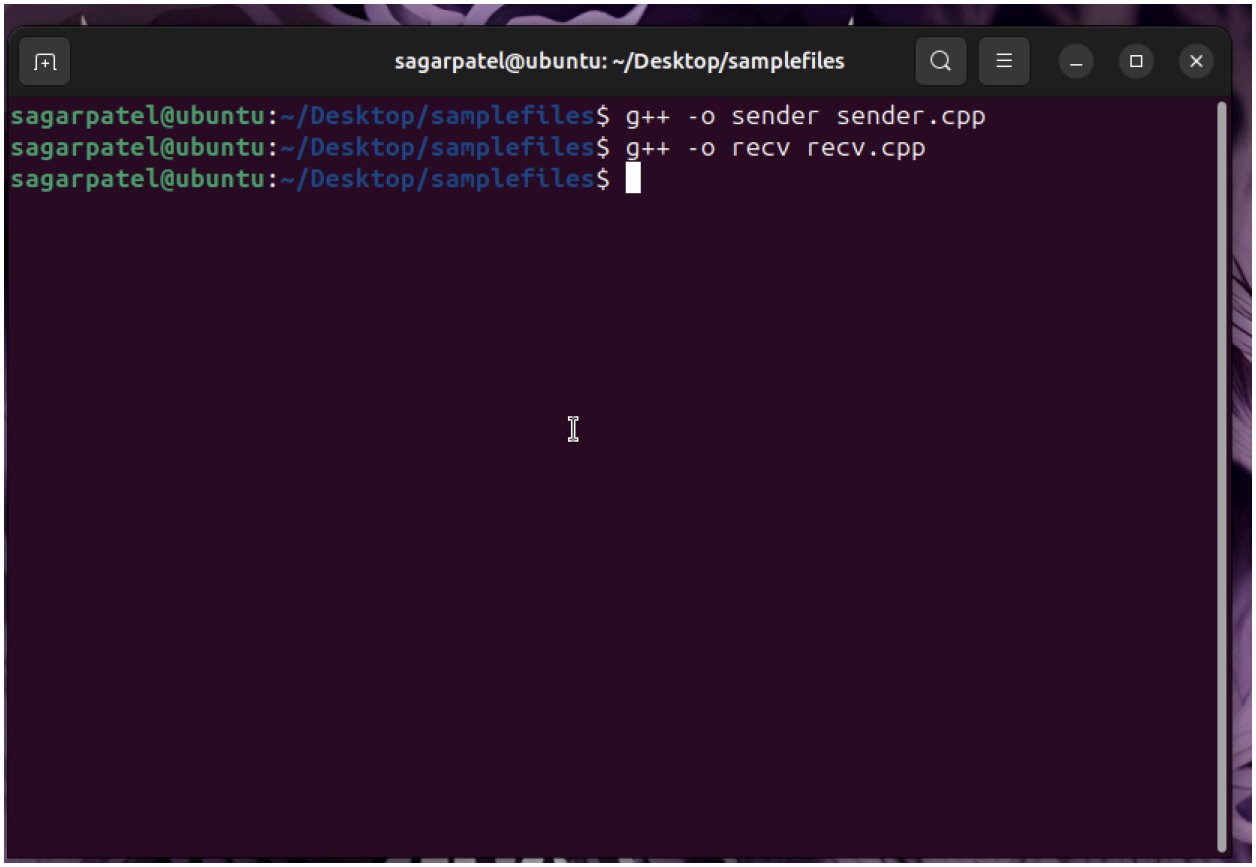
How to run the program

1. Make sure all the cpp files are in the same folder



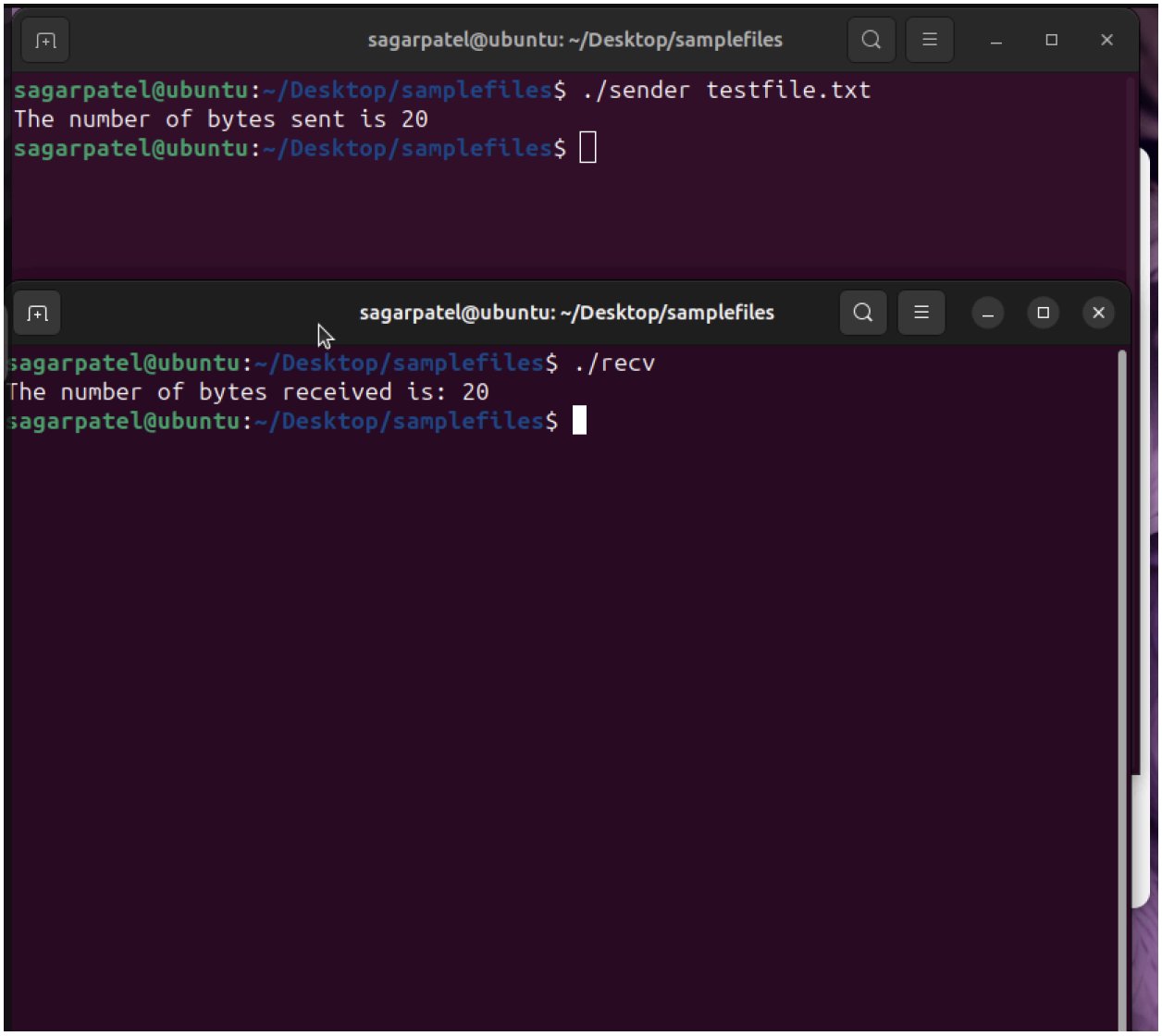
2. Create a .txt file and name it testfile.txt or you can use keyfile.txt as well (We decide to make a new file.)
3. Write “This is test file.” into testfile.txt

4. Open the terminal and go to the directory where your sample files folder is located, then run the g++ command for sender.cpp and recv.cpp
Command: g++ sender -o sender sender.cpp and g++ -o recv recv.cpp

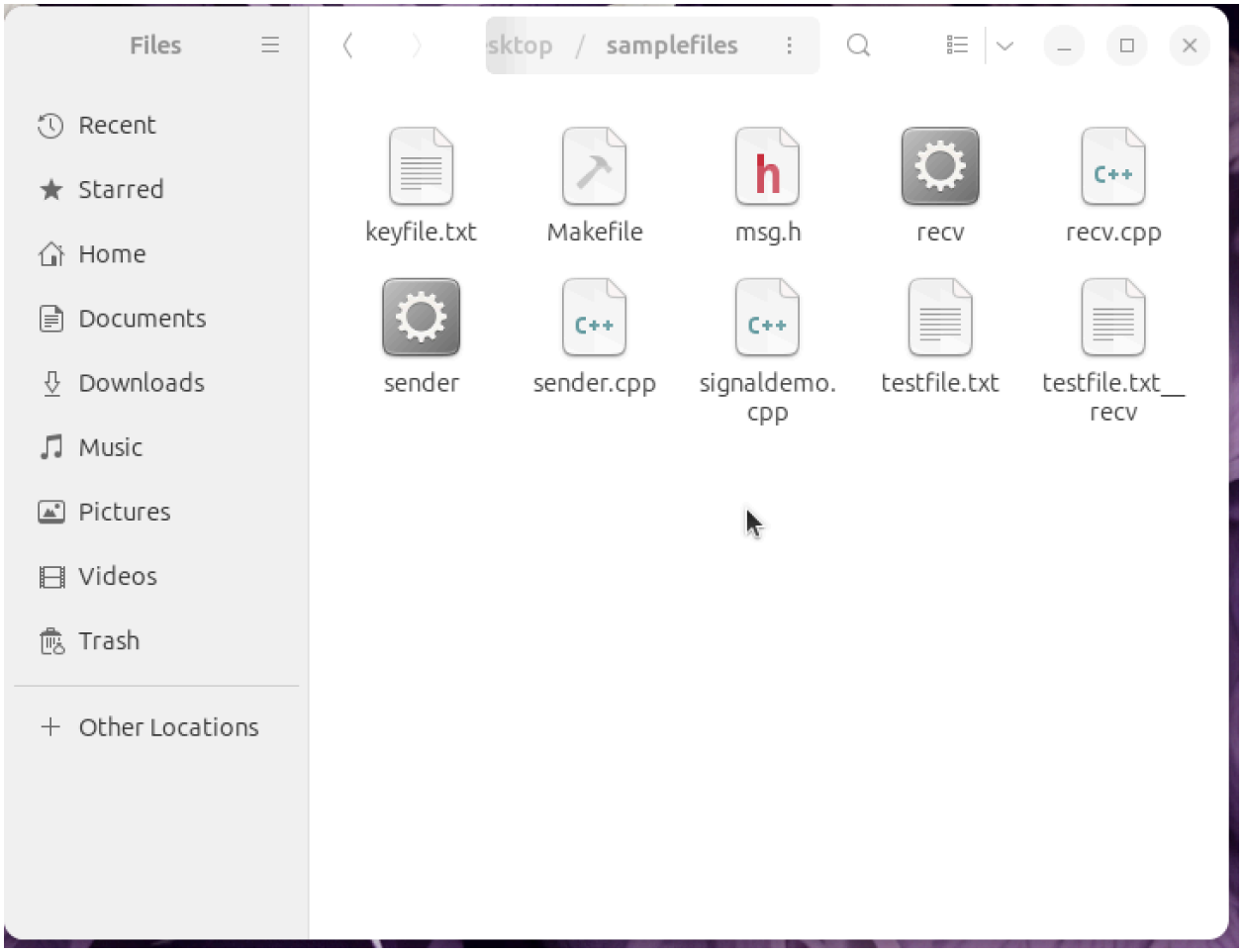


Once you are done, go back to your folder, and it should have two program debugged names: sender and recv.

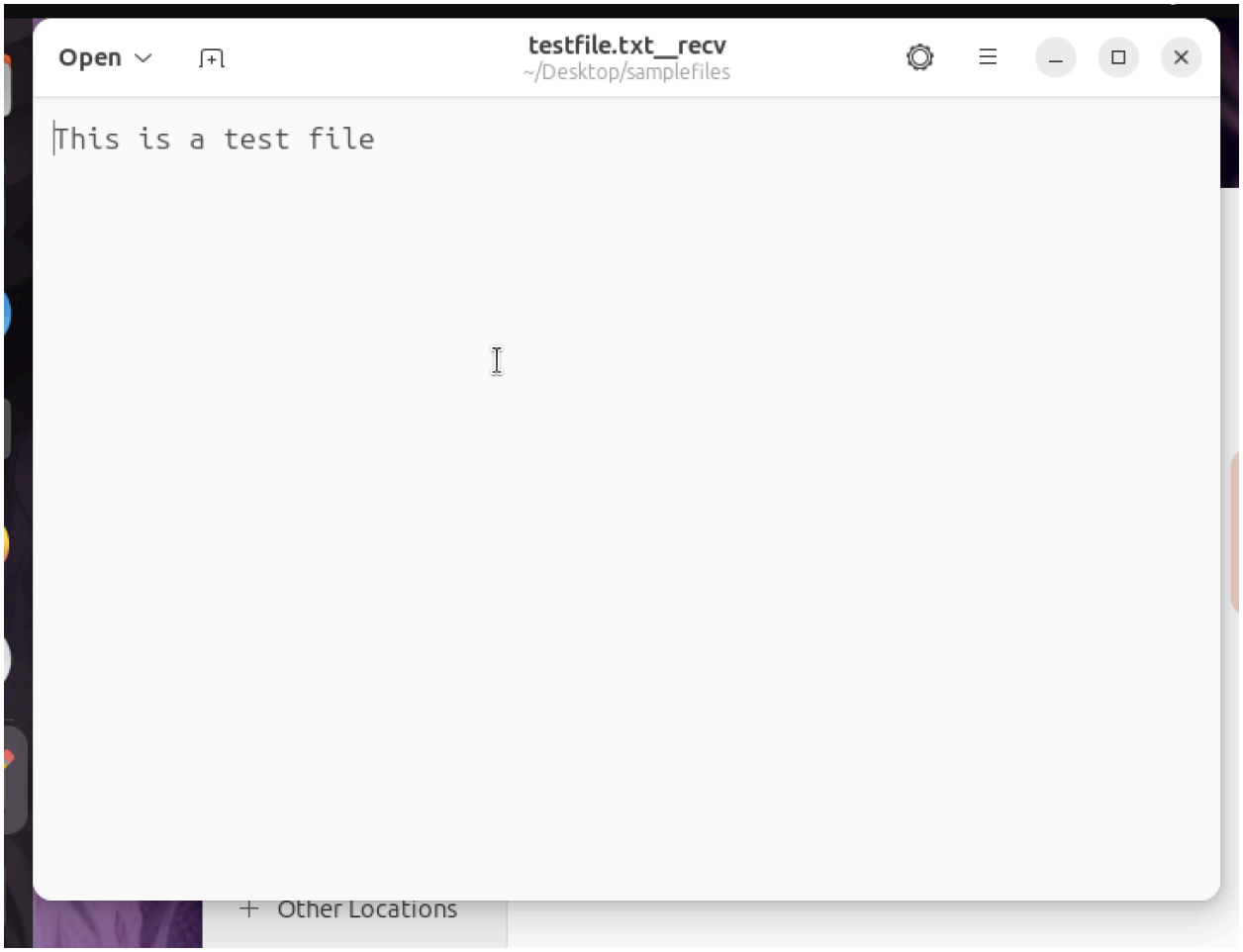
5. Open two terminals:
6. First one for the sender, type: ./sender testfile.txt in terminal
7. A second one for recv, type: ./recv in terminal



If the bytes number match in both terminals, the receiver got the file, so go back and check in your samplefiles folder.

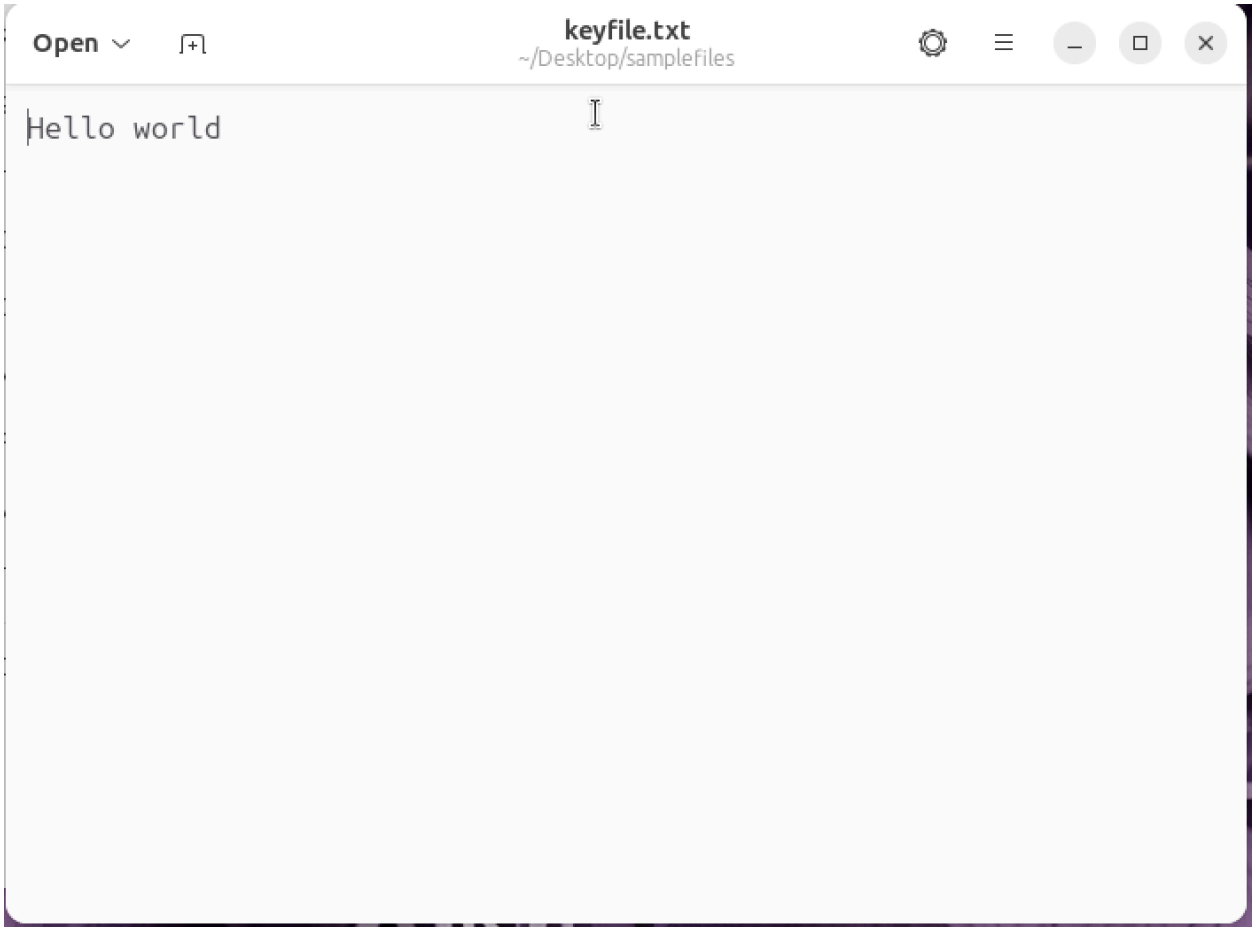


Testfile.txt__recv shows that the file was received; open it, and it should say the same sentence as testfile.txt.



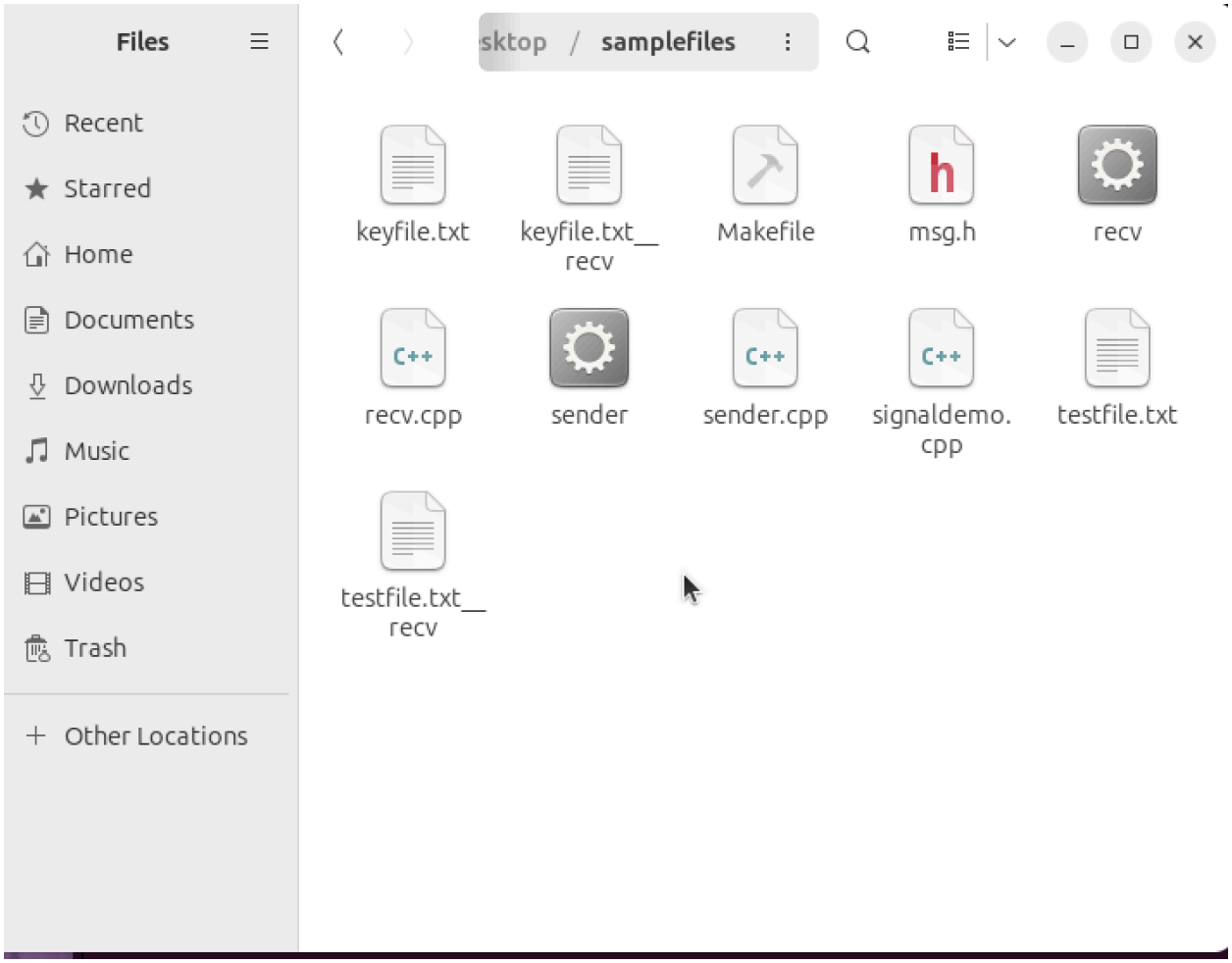
This shows your program worked successfully.

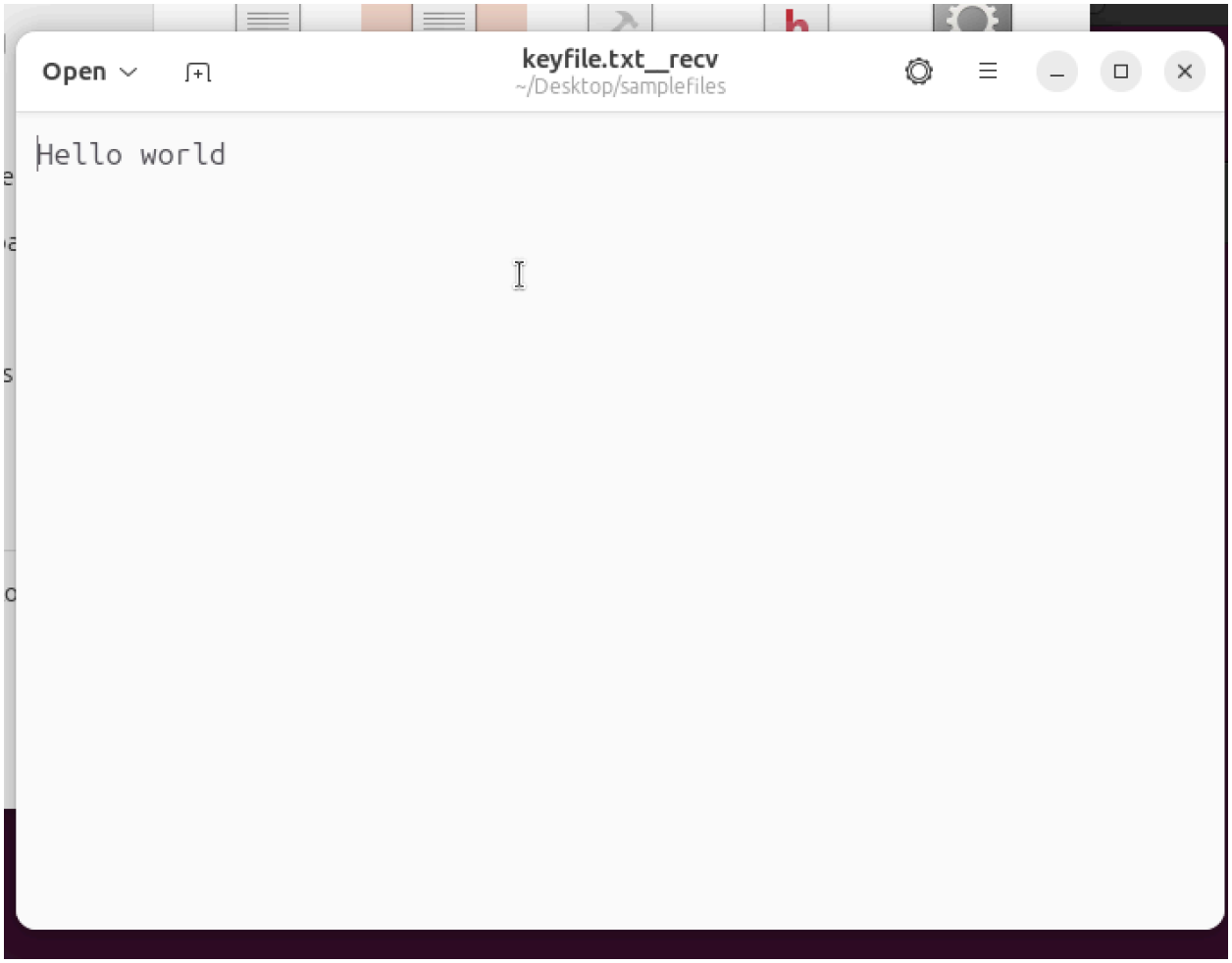
Hello world version



```
sagarpatel@ubuntu: ~/Desktop/samplefiles
sagarpatel@ubuntu:~/Desktop/samplefiles$ ./sender keyfile.txt
The number of bytes sent is 11
sagarpatel@ubuntu:~/Desktop/samplefiles$
```

```
sagarpatel@ubuntu: ~/Desktop/samplefiles
sagarpatel@ubuntu:~/Desktop/samplefiles$ ./recv
The number of bytes received is: 11
sagarpatel@ubuntu:~/Desktop/samplefiles$
```


Files
Recent
Starred
Home
Documents
Downloads
Music
Pictures
Videos
Trash
Other Locations
sktop / samplefiles
keyfile.txt
keyfile.txt__recv
Makefile
msg.h
recv
recv.cpp
sender
sender.cpp
signaldemo.cpp
testfile.txt
testfile.txt__recv


Open
keyfile.txt__recv
~/Desktop/samplefiles
Hello world