TEAM NAME: hatStripesCamo

Assignment 1

Software Design Document

Chris Weichlein, Miguel Cabrera, Stanley Shi, and Nate Ruppert

March 26, 2020

TABLE OF CONTENTS:

1 Introduction

1.1 Purpose

This document describes a program which synchronously transfers data between two processes

1.2 Scope

This application deals with two files, a sender and receiver program, and I/O text documents, and is meant as a demonstration.

2 Overview

At a high level, this project consists of two main modules: a sender module and a receiver module. They were implemented simultaneously as program functionality is entirely dependent upon both modules sending and receiving signals via a shared memory space. Message queues are implemented as well to indicate how many bytes were read from the file.

3 System Architecture

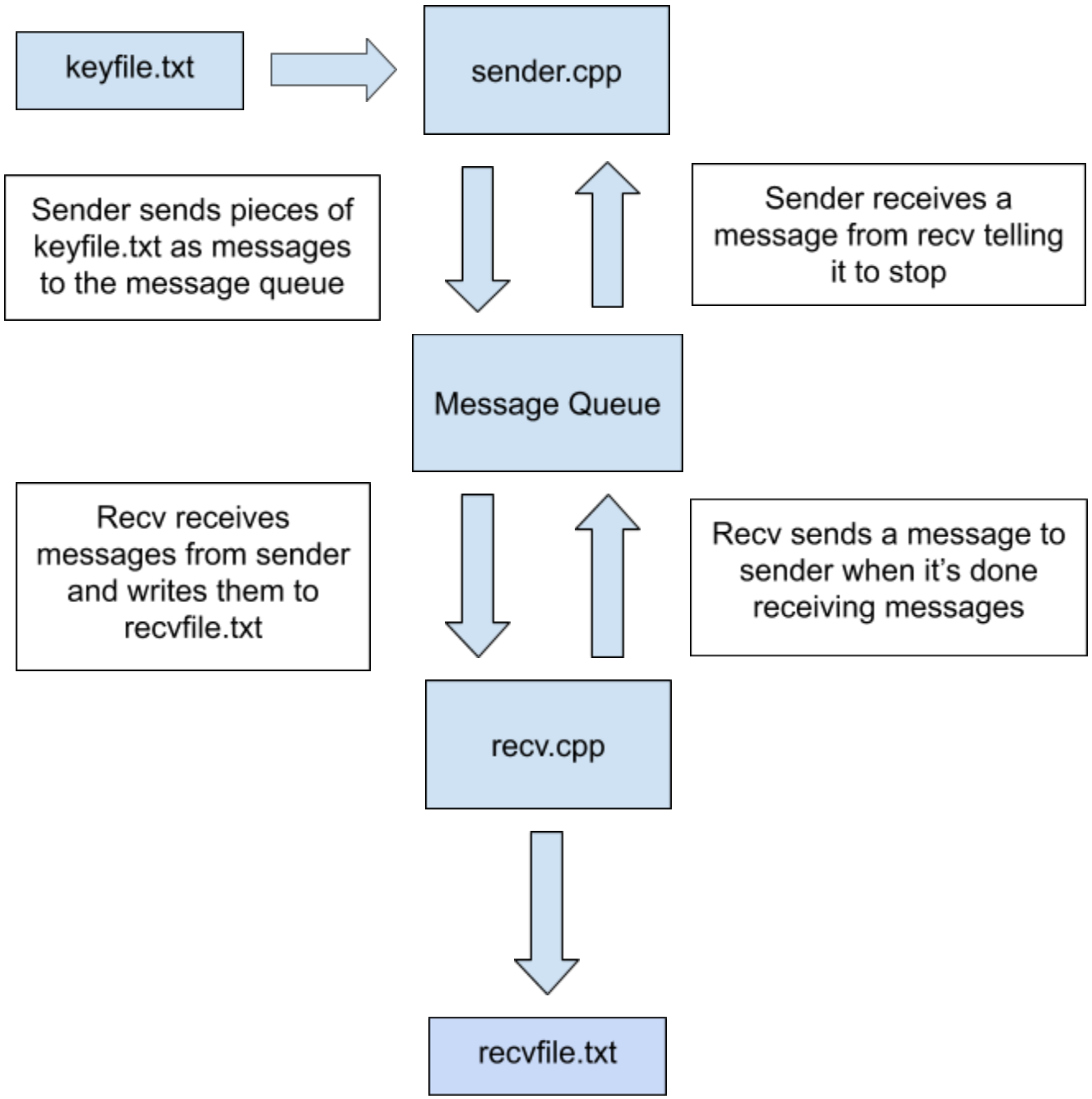
3.1 Architectural Design

The program is divided into two main modules, sender and receiver, which run simultaneously. What follows is the basic algorithm design as described in the assignment document:

1. Receiver sets up a shared memory space and a message queue for both components to operate off of.
2. Sender attaches to said memory space and connects to that message queue.
3. Receiver waits for a signal from Sender to appear in the message queue.
4. Sender reads some bytes of data from its text file and sends them to the shared memory space. If there are no bytes, it doesn't send anything to the shared memory space.
5. Sender sends a signal to the message queue in the form of a variable that indicates the number of bytes it read from the file and placed into the shared memory space. If there were no bytes, this number is zero, and Sender closes the file, detaches shared memory, and exits. If this is the case, skip step 6.

6. Sender waits for a return signal to appear back from the Receiver in the message queue, indicating the Receiver has received the Sender's signal.
7. Back on the Receiver's end, the Receiver gets the Sender's signal on the message queue and begins reading the number of bytes indicated in that signal, from the shared memory space. If the signal indicates that there are zero bytes to be read, however, Receiver will instead close the file, detach shared memory, deallocate this shared memory along with the message queue, and exit. This being the case, the program will terminate successfully.
8. The Receiver puts the data into its respective text file, and sends a signal back to Sender via the message queue.
9. The receiver waits for Sender to signal back.
10. Sender receives the signal from the message queue. Go back to step 4.

3.2 Decomposition



4. Data Design

4.1 Data Description

The following describes basic component to data structure mapping:

i. Sender and Receiver are implemented via the sender.cpp and recv.cpp files, respectively, and consist of main functions that call their own private functions.

ii. Shared memory and the message queue(a linked list in the kernel) are set up by Receiver.

iii. Bytes of data are stored in shared memory by Sender, and fields named size are input into the message queue to indicate the amount of bytes Sender last placed into the shared memory.

iv. The signal that Sender and Receiver emit to the message queue are the same type of structure called “message”, but Receiver does not have a size field (implemented as an int).

v. The makefile allows both Sender and Receiver to be built and run at once.

4.2 Data Dictionary

Sender: sender.cpp. A file that consists of a main and several helper functions.
Receiver: recv.cpp. A file that consists of a main and several helper functions.
message.h: contains the message structure, utilized by Sender and Receiver for the message queue. Receiver does not utilize *size*. “size” is an int, while the message type “mtype” is a long int, indicating whether it is a Sender/Receiver message.
Message Queue: a linked list in the kernel.
keyfile.txt: contains the message for Sender to read from.
recv.txt: contains the message for Receiver to write to.

5. Component Design

Receiver:

In main() call init().

In init(), open the keyfile.txt. Get the id of the shared memory segment, attach to it, and attach to the message queue.

Back in main, call mainLoop().

In mainLoop(), open/create the file for writing.

Set message size (msgSize) = 1 so we can enter the following loop...

While msgSize is not zero, do the following:

Get message from message queue, updating msgSize.
If (msgSize != 0){ save the shared memory to the recv.txt file, and send a received message to Sender.} Else {there is nothing to be read, and the recv.txt file is closed.}

Go back to main, and call cleanUp()

In cleanUp(), detach from shared memory, deallocate it and the message queue.

Sender:

In main(), call init().

In init(), create keyfile.txt. Create a key, and use it to get the id of the shared memory segment. Attach to this shared memory and the message queue. Save the id's to the message queue and the shared memory segment in variable msid and shmid, respectively. Go back to main().

In main(), call send(), passing the file name(pointer to a char) as a parameter.

In send(), open the file for reading. Create two buffers to store messages sent and messages received to/from the message queue, named sndMsg/rcvMsg.

While(!endOfFile):

Read a chunk of size SHARED_MEMORY_CHUNK_SIZE from the file.
Create a message object of type SENDER_DATA_TYPE and send it.
While(rcvmsg != RECV_DONE_TYPE): //effectively waiting for Receiver.
Update rcvmsg buffer with the message in the queue.

Send message with size set to 0, telling Receiver that the file has been read, then close the file and return to main(), where cleanUp() is then called, detaching shared memory.

6. Human Interface Design

6.1 Overview of Human interface

The user need only to open a terminal in the project's root directory and type in make. The program will create a tar file containing the contents of the directory.

6.2 Screen Images

```
miguels-MBP:hatStripesCamo miguelcabrera$ make
rm -f sender.out receiver.out recvfile.txt
g++ -o sender.out sender.cpp
g++ -o receiver.out recv.cpp
tar cvf CPSC351_Cabrera_Ruppert_Shi_Weichlein.tar *.cpp *.h makefile *.txt
a recv.cpp
a sender.cpp
a signaldemo.cpp
a msg.h
a makefile
a keyfile.txt
a msg.txt
./sender.out msg.txt & ./receiver.out
[Sender] Calling main
[Receiver] Calling main
[Sender] Sending message
[Sender] Waiting for response...
[Receiver] Receiving message
[Receiver] Sending response
[Receiver] Receiving message
[Sender] Sending message
[Sender] Waiting for response...
[Sender] Sending message

[Receiver] Receiving message
[Receiver] Sending response
[Receiver] Receiving message
[Receiver] Sending response
[Receiver] Receiving message
[Receiver] Sending response
[Receiver] Receiving message
[Receiver] Sending response
[Receiver] Receiving message
[Receiver] Sending response
[Receiver] Receiving message
[Receiver] Sending response
[Receiver] Receiving message
[Receiver] Sending response
[Receiver] Receiving message
[Receiver] Sending response
[Receiver] Receiving message
[Receiver] Sending response
[Receiver] Receiving message
[Receiver] Sending response
[Receiver] Receiving message
[Receiver] Detaching from memory
[Receiver] Deallocating from shared memory
[Receiver] Deallocating message queue
```