Allen Mrazek amrazek@csu.fullerton.edu
Jeremy Rico jjrico@csu.fullerton.edu
Saytu Singh saytusingh23@csu.fullerton.edu
Trenton Jansen trenton@csu.fullerton.edu

Design of Sender and Receiver

Table of Contents

General Description

This project consists of two separate programs that communicate using the System V IPC (Inter Process Communication) protocols. Two protocols are used: a message queue and a shared memory segment.

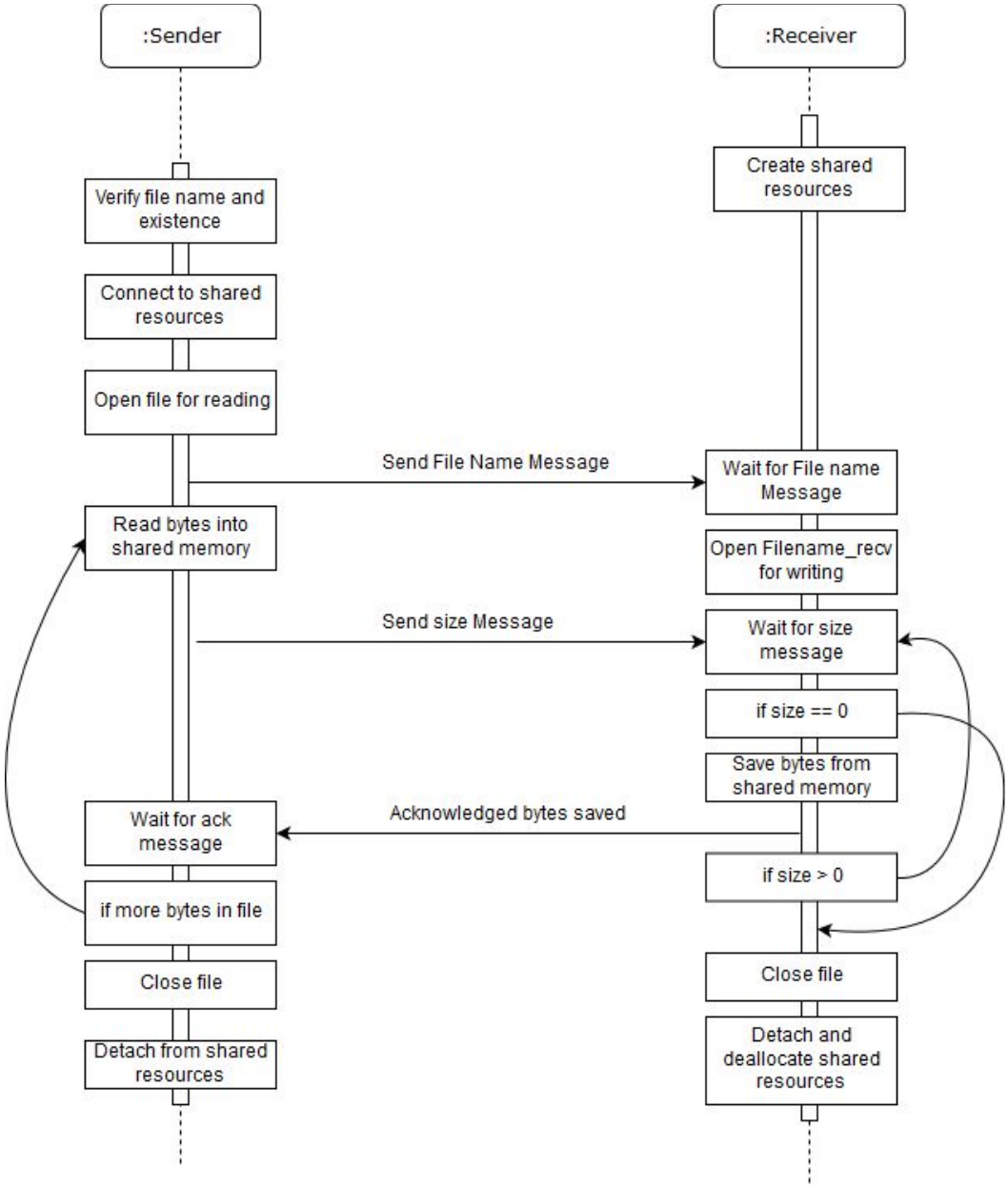
The receiver program sets up the shared resources and waits for an incoming file name message of type FILE_NAME_TRANSFER_TYPE from the sender program. When it has received this message, it prepares to write to a new file of the sent filename with "__recv" appended to it. Once the output file is ready to receive output, the receiver waits for additional

messages from the sender program consisting of the next chunk size. If this value is zero, the sender has sent the entirety of the file and the receiver may close the file and exit. If it is a positive nonzero value, the receiver reads the specified number of bytes from the shared memory segment and writes this data to disk. It then sends an acknowledgement message of type RECV_DONE_TYPE to the sender to indicate it is ready for the next file chunk.

The sender program receives an input filename as a command line argument. If the file exists and is opened successfully, the sender connects to the shared resources created by the receiver program. It sends a FILE_NAME_TRANSFER_TYPE message to the receiver program containing the file name. Once this message arrives, the receiver is ready to begin accepting chunks of the file. The sender reads the next file chunk into the shared memory segment. A SENDER_DATA_TYPE message to the receiver containing the chunk's size is dispatched, and the sender waits for acknowledgement before transmitting the next chunk. Once the entirety of the file has been read, the sender program transmits a final SENDER_DATA_TYPE message with size field set to zero to indicate that the file is complete and then exits.

System Diagram

This diagram depicts the flow and interactions of both sender and receiver:



Function Pseudocode

The following sections describe the purpose and pseudocode for the major functions in each program.

Sender Program

void init()

Purpose: Connect to the shared resources (message queue and shared memory segment) created by the receiver program. Note that the sender program does not create these resources; they are the responsibility of the receiver program.

Def init():

Generate unique key using keyfile.txt

Use key to attach to shared memory segment

Use key to attach to message queue

void sendFileName(const char fileName)*

Purpose: Send name of the file to the receiver

Def sendFileName(fileName):

If file name length exceeds MAX_FILE_NAME_SIZE:

Error

Let fMsg be a default-constructed fileNameMsg struct

Set fMsg mtype field to FILE_NAME_TRANSFER_TYPE

Copy contents of fileName to fMsg fileName field

Insert fMsg into message queue

void sendFile(const char fileName)*

Purpose: Send the contents of the file to the receiver program in chunks. The sender waits for acknowledgement of each chunk before preparing the next one.

Def sendFile(fileName):

Let sndMsg be a message struct

Let rcvMsg be an ackMessage struct

Let numBytesSent be an unsigned long initialized to 0

Open fileName for reading

If failed to open file:

Error

While not end of file:

Read bytes from file into shared memory segment

Set sndMsg size field to num bytes read from file

Increment numBytesSent by sndMsg size

Set sndMsg mtype field to SENDER_DATA_TYPE

Place sndMsg on message queue

Wait for an acknowledgement message from receiver

Close file

void cleanUp()

Purpose: Detach from shared memory segment

Def cleanUp():

Detach from shared memory segment

Receiver Program

void init()

Purpose: Create shared resources that will be used by both the receiver program and the sender program. Note that the receiver owns these resources, so it will be responsible for destroying them as well.

Def init():

Generate a unique key using keyfile.txt

Create an exclusive shared memory segment using generated key

Attach to the shared memory segment

Create message queue using key

string recvFileName()

Purpose: Receives name of file from sender program, in the form of a FILE_NAME_TRANSFER_TYPE message from the message queue.

Def recvFileName():

Let fileName be a string

Wait for a FILE_NAME_TRANSFER_TYPE message

Copy the message's fileName field into fileName

Return fileName

unsigned long mainLoop(const char fileName)*

Purpose: Receive file chunks from the sender program, saving them into the specified file name (with __recv appended)

Def: mainLoop(fileName):

Let msgSize = -1

Let numBytesRecv = 0

Let recvFilenameStr be a string = fileName + "__recv"

Open recvFileNameStr for writing

While msgSize is not zero:

Wait for a SENDER_DATA_TYPE message

Set msgSize to the message's size field

If msgSize is not zero:

Add msgSize to numBytesRecv

Write msgSize bytes from shared memory segment to file

Place acknowledgement message into message queue

Else: // msgSize is zero

Close file

End While

Return numBytesRecv

void ctrlCSignal(int signal)

Purpose: Allow user to exit receiver program without receiving a file using a Ctrl+C interrupt.

Def ctrlCSignal(signal):
Call cleanUp() function

void cleanUp()

Purpose: Destroy shared resources created in *init*

Def cleanUp():

If attached to shared memory segment:

Detach from shared memory segment

If created shared memory segment:

Destroy shared memory segment

If created message queue

Destroy message queue