Pseudo Code

Tyler trepanier - A00850517

Client Server

Linux internal message queues

Table of Contents

[Client Server (main program entry) 2](#_Toc442041767)

[START 2](#_Toc442041768)

[PARSE COMMAND LINE 2](#_Toc442041769)

[Server 3](#_Toc442041770)

[SERVER 3](#_Toc442041771)

[SEARCH FOR CLIENTS 3](#_Toc442041772)

[CREATE NEW CLIENT PROCESS 3](#_Toc442041773)

[START (process) 4](#_Toc442041774)

[DESIGNATE PRIORITY 4](#_Toc442041775)

[OPEN REQUESTED FILE 4](#_Toc442041776)

[Open requested file 4](#_Toc442041777)

[SEND MESSAGES 4](#_Toc442041778)

[PACKETIZE DATA 5](#_Toc442041779)

[END (process) 5](#_Toc442041780)

[Client 7](#_Toc442041781)

[CLIENT 7](#_Toc442041782)

[OUTPUT FUNCTION 7](#_Toc442041783)

[DISPLAY MESSAGE 7](#_Toc442041784)

[PROMPT USER INPUT 8](#_Toc442041785)

[READ SERVER RESPONSE 8](#_Toc442041786)

# Server

This sections comes into effect when a user has selected the server option when running this application. The purpose of this section is to separate the functionality of the client and server functions. The Server’s responsibilities include creating the message queue where all clients will connect to. From there, the server will respond to all client file requests and return either error information or the file contents itself.

## SERVER

Redirect all signals to the Signal Handler.

Create ClientServer msg queue

Assign Message-Type 1 to be the messages designated Client to Server

Goto SEARCH FOR CLIENTS

## SEARCH FOR CLIENTS

Check for any incoming messages from clients.

* Failure no new messages: RESTART SEARCH FOR CLIENTS

Pass message to CREATE NEW CLIENT PROCESS.

Goto CREATE NEW CLIENT PROCESS

## CREATE NEW CLIENT PROCESS

Check message data for client’s PID

Check message for client‘s priority

IF no priority specified, set priority to maximum

Fork a child process

IF Process is the child

Pass client information to this process.

Assign the task START (process)

Goto START (process)

ELSE

This is the Parent Process

Goto SEARCH FOR CLIENTS

## PROCESS CLIENT

Get the client’s message.

Prepare the series of messages with the client’s priority and their assigned message type.

Goto DESIGNATE PRIORITY.

Get the file’s name from received message.

Attempt to open the file.

* Success:
  + Pass the File, client’s message type and priority to PACKETIZEDATA.
  + Use PACKETIZEDATA to split the file’s contents and send them to the client.
  + Goto PACKETIZE DATA.
* Failure:
  + Send a message using the client’s specified message type back to the client indicating that there was an error with their file request.

The message(s) have been sent successfully.

Terminate this process.

## DESIGNATE PRIORITY

Get Client’s Priority and PID

Designate new length of message to be MAXSIZE / PRIORITY

Goto OPEN REQUESTED FILE

## OPEN REQUESTED FILE

Get message data from client

Read desired Filename from message data

Open requested file

* Failure: File cannot be opened for reading
  + Send “Error: Cannot open file” message to SEND MESSAGE
  + Goto SEND MESSAGE
* Success: File has been read
  + Send this file to PACKETIZE DATA
  + Goto PACKETIZE DATA

## SEND MESSAGES

Grab assigned PID to indicate message type (destination of message)

Grab the message queue

Grab the message

Grab the “more” flag to indicate more messages

Send message to the message queue using the message type as its address.

IF “more” flag has been specified

Return to PACKETIZE DATA (restart its file read loop)

Goto PACKETIZE DATA

## PACKETIZE DATA

Grab file contents

Grab the message queue

Grab the PID

Grab client priority

Read file until designated length of message

WHILE the file can be read

Check if file can be read

IF file cannot be read

Get out of while loop

Send the portion of the message to SEND MESSAGES

Send the PID to SEND MESSAGES

Send the Message Queue to SEND MESSAGES

Send the “more” flag to SEND MESSAGES

Goto SEND MESSAGES

Restart loop

File has finished reading

Goto END (process)

## CLEAN UP RESOURCES

Free all used resources.

Restore normal signal functionality to the terminal.

End program.

Goto END.

## END (process)

Free all used resources.

Terminate this child process.

# Client

This section comes into effect when a user has selected the client option when running this application. The purpose of this section is to separate the functionality of the client and server functions. The Client is responsible for reading file requests, made by the user from Standard Input, and requesting those files from the existing server. These requests will be made by taking advantage of the Linux message queue to allow for interprocess communication.

## CLIENT

Open or create the Client-Server message queue.

Redirect all signals to the Signal Handler.

Prompt for user input.

Goto USERINPUT

Arrange the user request into the format of “<filename> <(if available)priority> <client ID>”

Read from the Server to all clients message queue on the server’s response.

Goto READ SERVER RESPONSE.

Assign the task of reading to the DisplayMessage which will create a new thread.

Goto DISPLAYMESSAGE.

## CREATE READ THREAD

Get the message read from the server’s response.

Create a detached thread.

Pass the message queue to the thread’s parameters.

Assign the task of READ SERVER RESPONSE to the thread.

* The new thread: Goto READ SERVER REPONSE
* The main thread: Goto READ COMMAND-LINE ARGUMENTS

## READ SERVER RESPONSE

This is a thread function dedicated to reading a server’s messages from the message queue.

Get this process’ ID

Assign the desired message type to be this process’ ID

In a continuous loop that will end ONLY when the server indicates there are no more messages

Read the next message from the queue using the desired message type (process ID)

Read the message into a temporary message.

IF the message is an empty message from the server

Leave the continuous loop.

ELSE

Get message data from the message (only the string, not any other attributes).

Goto DISPLAY MESSAGE

The message has been successfully printed out and there are more messages to read.

RESTART the loop.

Release all used resources.

Tell the main thread of this process that reading is complete.

Kill this thread.

## DISPLAY MESSAGE

Get the message’s data (only the characters)

Print out the message’s data to standard output.

Return to SERVER RESPONSE loop

## READ COMMAND-LINE ARGUMENTS

GET the command-line arguments

Create a blank message for sending.

Treat the first arguments as a filename.

Put the first argument into the message.

IF there is another argument

Check if the argument is a number.

IF it isn’t a number or it is a number less than 1

Set the client’s priority as 1, the maximum priority.

ELSE IF the number is higher than 1000

Set the priority as 1000

ELSE the number is in between 1 and 1000

Set the priority as the number specified.

ELSE there isn’t a second argument

Set the priority as 1.

Include the client’s process ID inside of the message contents.

Set the message type to the Client\_to\_Server (which is defined as 100).

Message contents have been filled.

Goto SEND MESSAGE TO SERVER

## SEND MESSAGE TO SERVER

Get the filled message.

Send the message to the queue using the Client\_to\_Server message type.

IF the message was sent successfully

Wait for the server to finish sending all of the data.

Goto WAIT FOR THREAD

ELSE the message was not sent properly

Goto CLEAN UP RESOURCES

## WAIT FOR READ COMPLETION

In a continuous loop that checks if the thread has finished completing

IF thread is not finished

RESTART this loop

ELSE the thread has finished reading

Break out of the continuous loop

Goto CLEAN UP RESOURCES.

## CLEAN UP RESOURCES

Free all used resources.

Stop the Read thread from reading.

Restore normal signal functionality to the terminal.

End program.

Goto END.