Pseudo Code

Tyler trepanier - A00850517

Client Server

Linux internal message queues

Table of Contents

[Server 2](#_Toc442272308)

[SERVER 2](#_Toc442272309)

[SEARCH FOR CLIENTS 2](#_Toc442272310)

[CREATE NEW CLIENT PROCESS 3](#_Toc442272311)

[CLEAN UP RESOURCES 3](#_Toc442272312)

[Client Process 4](#_Toc442272313)

[PROCESS CLIENT 5](#_Toc442272314)

[DESIGNATE PRIORITY 6](#_Toc442272315)

[OPEN REQUESTED FILE 6](#_Toc442272316)

[SEND MESSAGES 6](#_Toc442272317)

[PACKETIZE DATA 7](#_Toc442272318)

[END (process) 8](#_Toc442272319)

[Client 9](#_Toc442272320)

[CLIENT 9](#_Toc442272321)

[CREATE READ THREAD 9](#_Toc442272322)

[READ SERVER RESPONSE 10](#_Toc442272323)

[DISPLAY MESSAGE 10](#_Toc442272324)

[READ COMMAND-LINE ARGUMENTS 10](#_Toc442272325)

[SEND MESSAGE TO SERVER 11](#_Toc442272326)

[WAIT FOR READ COMPLETION 12](#_Toc442272327)

[CLEAN UP RESOURCES 12](#_Toc442272328)

# Server

This section 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 the message queue (Goto Open Queue)

IF Open Queue fails

Attempt to close application

Goto CLEAN UP.

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

Goto SEARCH FOR CLIENTS

## SEARCH FOR CLIENTS

WHILE the user does not want to quit the application

IF the user wishes to quit the application

Leave this while loop.

Check for any incoming messages from clients.

* Failure no new messages: RESTART SEARCH FOR CLIENTS
* Message found in the queue: proceed to next step.

Message was found and there is a new client to be processed.

Fork a new process to handle the client’s connection.

Goto CREATE NEW CLIENT PROCESS

Parent process has finished created the new client process

RESTART the while loop.

The while loop has ended because the user wants to end the application.

Release some resources that have been used by the application.

Goto CLEAN UP RESOURCES.

## CREATE NEW CLIENT PROCESS

Get the message received from the Server.

Fork a child process.

* Child:
  + Assign that process the task of process client
  + The new process will diverge and perform the PROCESS CLIENT function
  + After the PROCESS CLIENT task is finished this child process will exit.
* Parent:
  + RESTART this while loop

## CLEAN UP RESOURCES

Free all used resources.

Restore normal signal functionality to the terminal.

End program.

Goto END.

# Client Process

Whenever a server detects a new message in the message queue, a new client process will be created for each individual client that makes a request for a message.

The purpose of separate process for each individual client is to ensure that there is a disconnection between a server itself and instead, handle the client’s request. This request involves the opening of a file and sending the contents of a file to the client who made the request. If there is an error in opening a file, an error will be send instead to the Client who made the request and this process will immediately terminate.

## PROCESS CLIENT

Get the client’s message.

Prepare a blank message with the client’s priority and their assigned message type.

Goto DESIGNATE PRIORITY.

IF Designate Priority fails

Terminate the process.

Goto END (process).

Get the file’s name from received message.

Attempt to open the file.

Goto OPEN FILE

IF Open File succeeds

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.

IF Open File fails to open a file

Fill a message with the data indicating that there is an error opening the file.

Send the message to the Client

Goto SEND MESSAGE and return to this function when completed.

Terminate this process regardless of send message failure or success.

Goto END (process)

## DESIGNATE PRIORITY

Get the Client’s message.

Parse the Client’s priority and PID from the message.

IF we cannot parse the Client’s priority and PID

Return failure to PROCESS CLIENT

Return success to PROCESS CLIENT

## 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

Make the size of the message to be (MAXIMUM / priority).

Fill the shared content of all messages to be sent (size of message and type of message).

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

Copy the number of characters (specified by size of message) into the message.

Pass the message to SEND MESSAGES

Pass the PID to SEND MESSAGES

Pass the Message Queue to SEND MESSAGES

Goto SEND MESSAGES and Return to this function.

Restart loop

File has finished reading

Send another message indicating a finished communication.

Goto SEND MESSAGE and return to this function.

Close the file that was open.

Goto END (process)

## 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.