MQ MULTIPLEXER

DESIGN AND TESTING DOCUMENT

Abstract

A document explains the design of a client server message queue multiplexer with state diagrams and pseudocode. Test cases are highlighted at the end of the document.

Justin Cervantes

Cervantes.jfa@gmail.com

Contents

About the MQ Multiplexer

The objective of the MQ Multiplexer is to demonstrate that you can set priority by reading a file at different speeds using a single IPC resource. In this particular application, this is done by ensuring that an equally proportional number of messages for each client are put on the message queue and manipulating the size of the message for each client based on priority. This is managed with the use of a semaphore. The higher the number of chars read per message, the higher priority the client will have. Each client is not scanning for a priority, but instead its own PID.

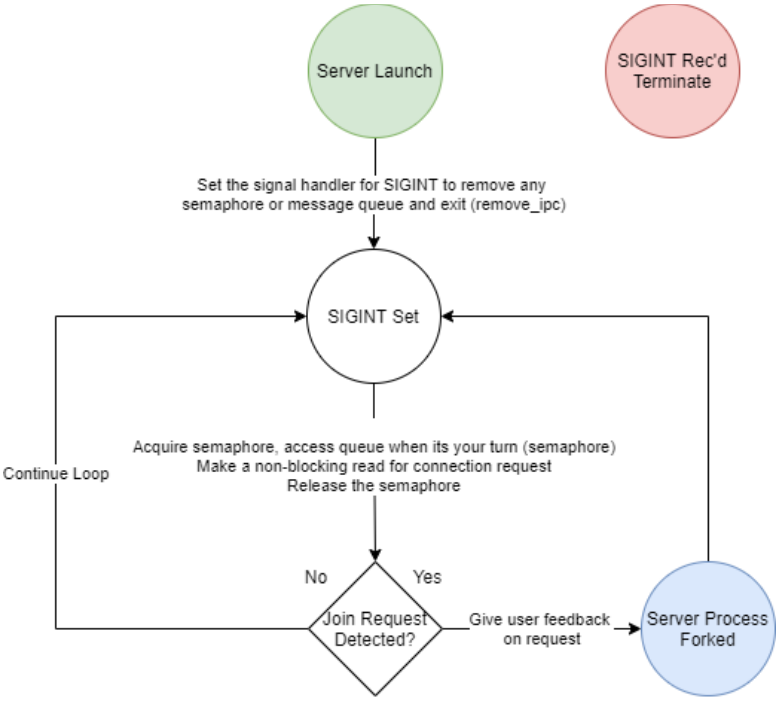
The MQ Multiplexer is a client server application which has one executable for clients and one executable for the server. Written in C++, the application takes advantage of a message queue and a semaphore to multiplex multiple clients requesting a server to read a requested text file and send a message back in increments specified by the client.

Design

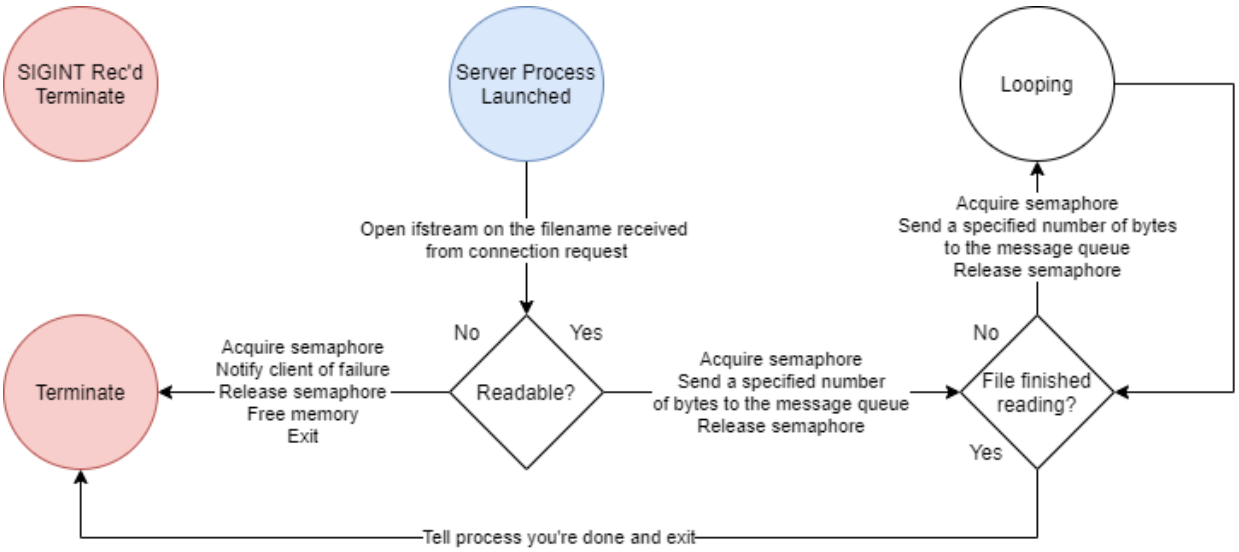
The design relies on the use of a shared semaphore which guards the message queue API calls. At any given time, anyone who wants to use the message queue will lock everyone else out, forcing all those who want to use it to enter a lineup, thus waiting for the resource. You can't re-enter the lineup for the message queue resource until every other application has had an opportunity to line up. This lineup is managed atomically by the operating system, and so you can be relatively sure that if processes of the client all have the same priority from the OS, then you should have proportional messages being entered into the message queue for reading by clients.

State Diagrams

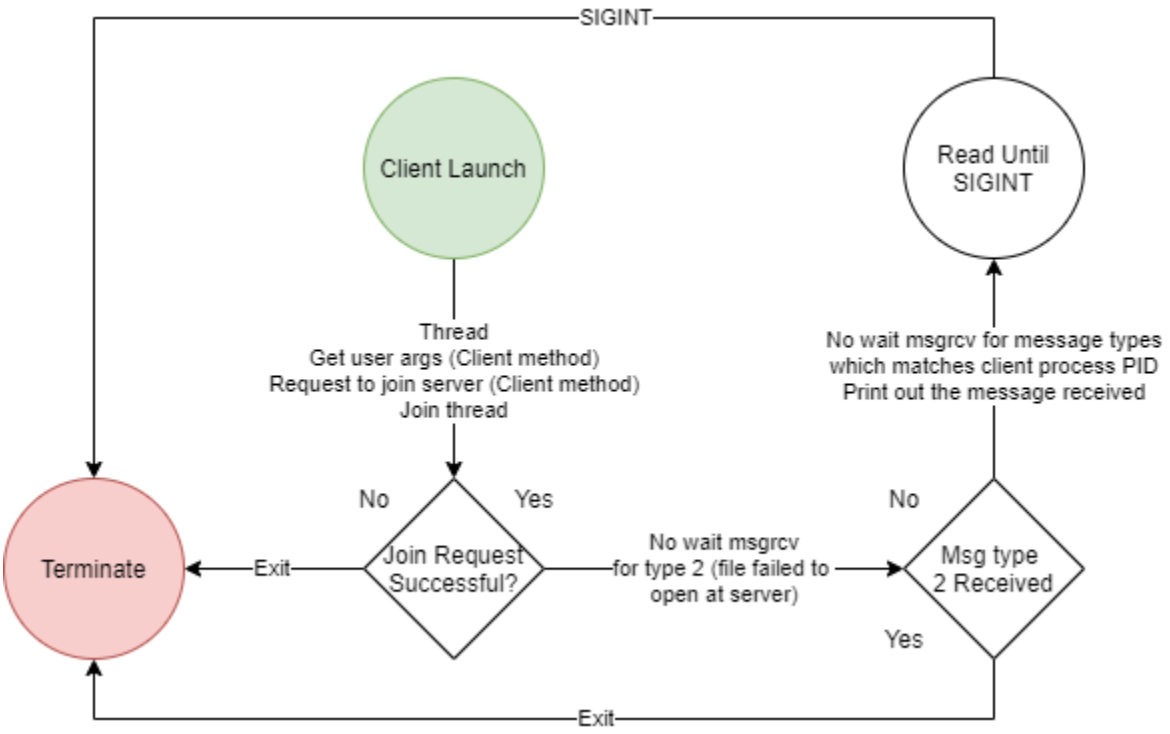
Server



Server Child Processes



Client



Pseudocode

Server

main

Create a signal handler to watch for a SIGINT signal (signal interrupt, ie ctrl+c) which will delete the IPCS

In a loop, makes a non-blocking read on the message queue looking for priority 1 messages (requests to join the server). These requests should be captured in a semaphore to ensure that it isn't starved out of getting onto the message queue due to competition from the client processes.

If a request to join is received, fork a child process that calls the server method and continues to loop looking for more type 1 messages.

server

Read the message contained from the parent process and extract the file name that it wants to have read back to it.

Attempt to open the file using an ifstream; if it fails to open, send a message (guarded by a semaphore to make sure it has the opportunity to line up for the resource) which lets the client know an error occurred and that it can terminate. Use the pid of the client to send messages to it via the message type. Terminate yourself afterwards.

If the ifstream creation is successful, in a loop, constantly reset the message values and send a message with the characters from the text file specified (the number of chars to specified will be passed to you from the parent process).

If the end of the file is reached, terminate yourself after letting the user know the text file was successfully read.

Client

main

Launch a thread which makes an initial request to join the server.

In a loop, check for message types which are either 2 (received immediately after a file open fails). This should be guarded in a semaphore.

If a 2 message type is detected, terminate yourself.

Make another non-blocking read which is guarded in a semaphore and look for messages with a message type that matches your PID. Echo out the contents once you receive something.

Client (thread which requests to join the server)

Get user input on your priority (number from 1 – 32). This signifies the amount of bytes to get.

Get user input for what text file to open.

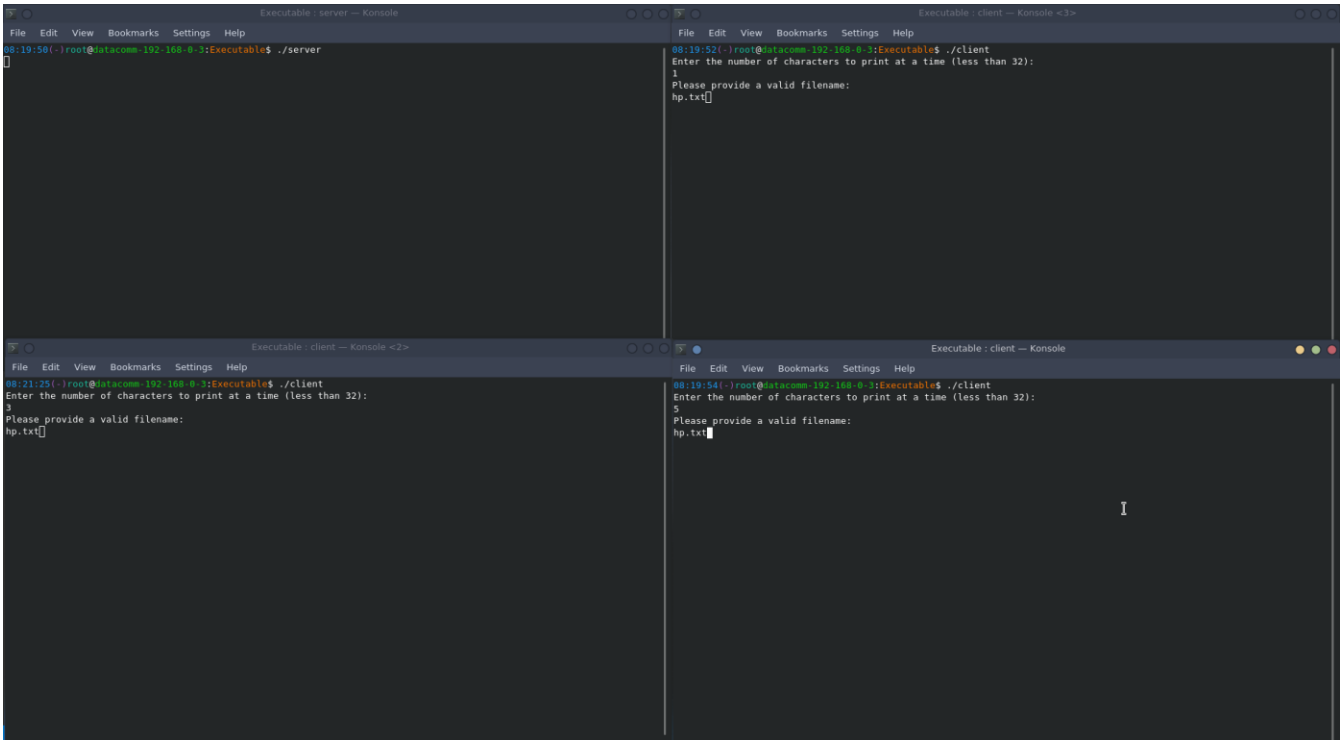
Send a message with this information along with your PID with a message type of 1. If message send fails, inform client and terminate.

Test Cases

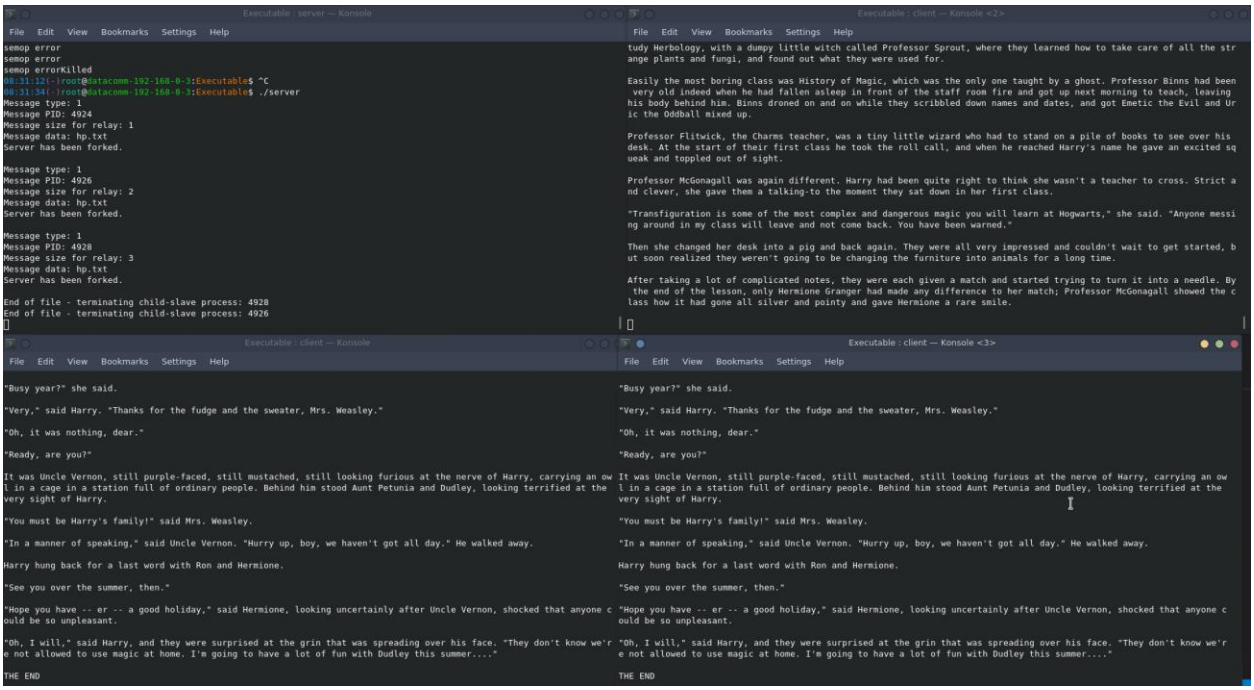
1 char, 3 char, 5 char scroll speeds

Description: The user, if they run a program with the same text file with different priorities, should have file transfers complete with order according to priority (ie higher priorities finish faster given the same text file)
Success criteria: If you launch a client with 1, 3 and 5 priorities on the same text file at the same time, priority 5 should finish first, followed by 3 then 1.
Status: Passed

Precondition



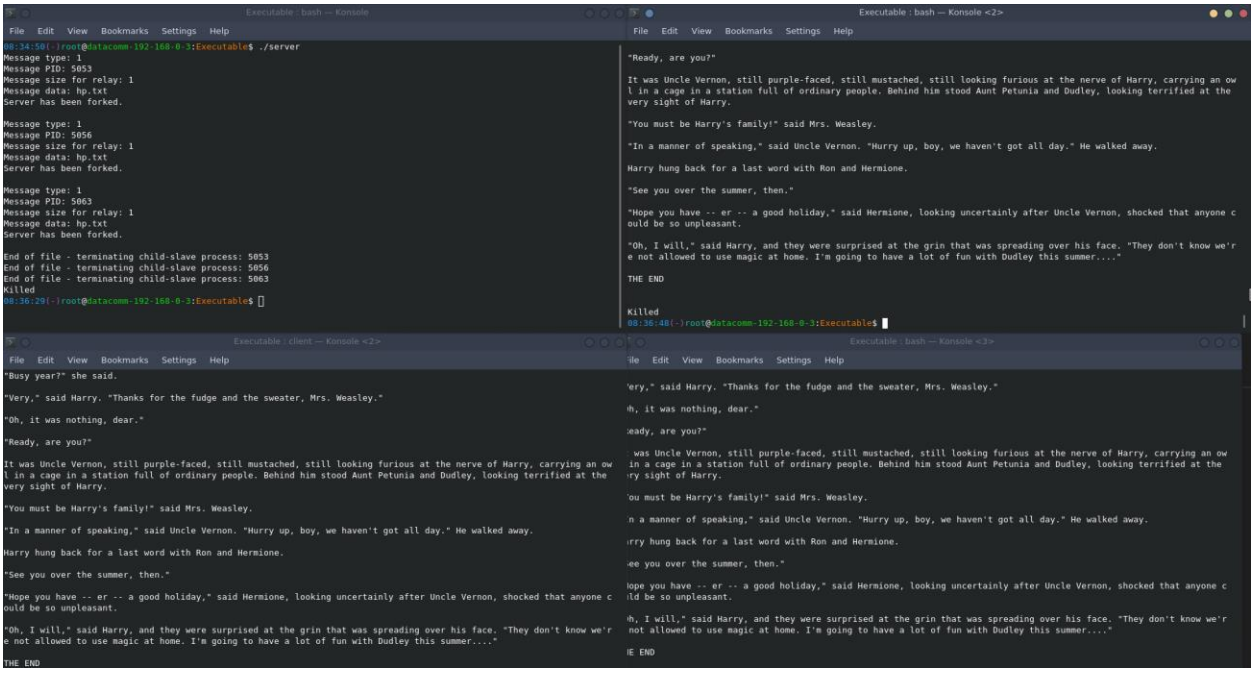
Postcondition



1 char per client on all clients, each ends in the order they started

Description: With equal priority, the clients should finish in the order in which they started, demonstrating that the semaphores force the clients to line up to access the requested resource.
Success criteria: Processes finish in the order in which they were started.
Status: Passed

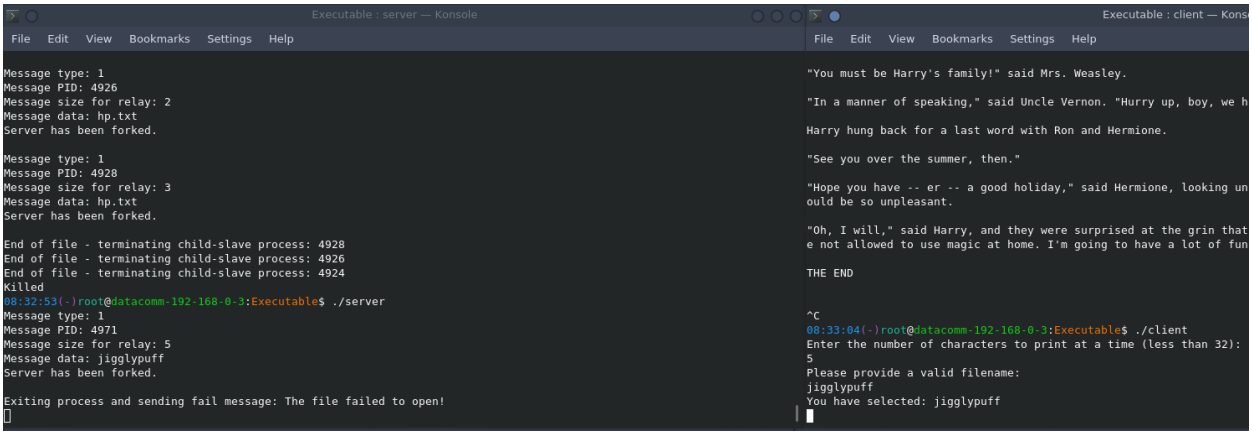
Each end in order they started



File fails to open

Description: If a file fails to open, the client and server process should terminate.
Success criteria: The system out message for the error should appear when the ifstream returns an error message upon trying to open a text file. This should be seen on both client and server side as the server lets the client know it failed to open it.
Status: Passed

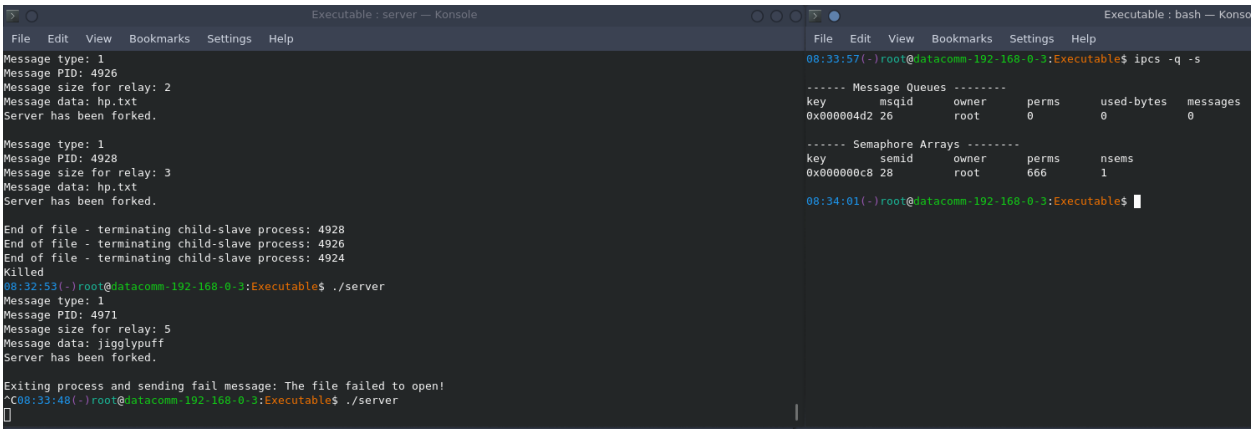
Failure message



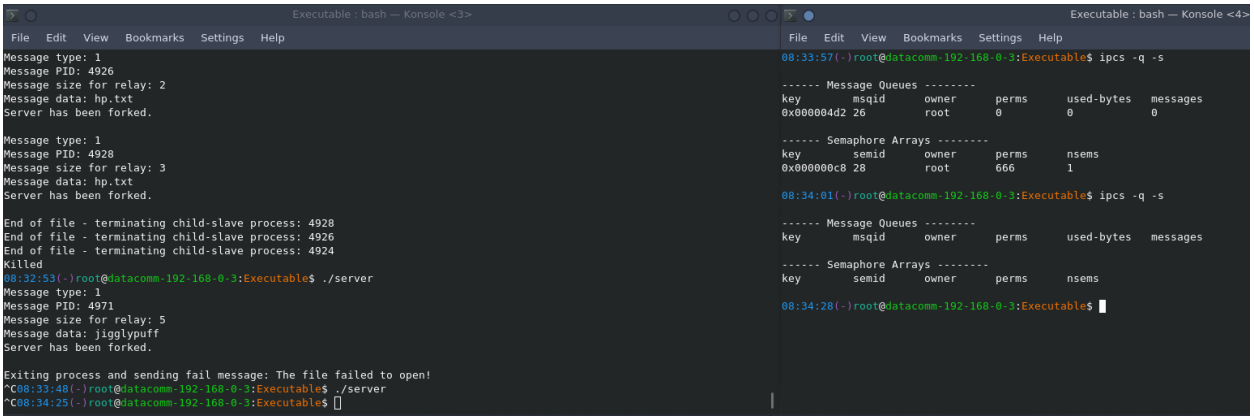
SIGINT received on server

Description: IPC channels should all be removed at the end of the program (intended to be SIGINT).
Success criteria: Checking ipcs -q -s should show no semaphores nor message queues left in the system after invoking the SIGINT signal.
Status: Passed

Pre SIGINT



Post SIGINT


Executable : bash — Konsole <3>
File Edit View Bookmarks Settings Help
Message type: 1
Message PID: 4926
Message size for relay: 2
Message data: hp.txt
Server has been forked.
Message type: 1
Message PID: 4928
Message size for relay: 3
Message data: hp.txt
Server has been forked.
End of file - terminating child-slave process: 4928
End of file - terminating child-slave process: 4926
End of file - terminating child-slave process: 4924
Killed
08:32:53(-)root@datacomm-192-168-0-3:Executable$ ./server
Message type: 1
Message PID: 4971
Message size for relay: 5
Message data: jigglypuff
Server has been forked.
Exiting process and sending fail message: The file failed to open!
^C08:33:48(-)root@datacomm-192-168-0-3:Executable$ ./server
^C08:34:25(-)root@datacomm-192-168-0-3:Executable$
Executable : bash — Konsole <4>
File Edit View Bookmarks Settings Help
08:33:57(-)root@datacomm-192-168-0-3:Executable$ ipcs -q -s
------ Message Queues --------
key msqid owner perms used-bytes messages
0x000004d2 26 root 0 0 0
------ Semaphore Arrays --------
key semid owner perms nsems
0x000000c8 28 root 666 1
08:34:01(-)root@datacomm-192-168-0-3:Executable$ ipcs -q -s
------ Message Queues --------
key msqid owner perms used-bytes messages
------ Semaphore Arrays --------
key semid owner perms nsems
08:34:28(-)root@datacomm-192-168-0-3:Executable$