

LINUX IPC: Msg QUEUE

ASSIGNMENT 2

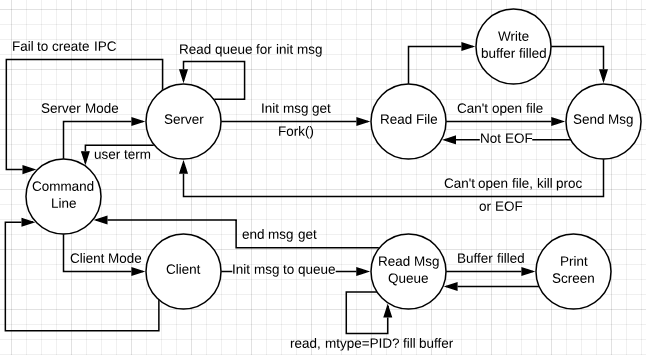


February 27, 2019

Jacky Li

A00823377

# Design: State Diagram



# Design: Pseudocode

**Open\_Queue**

Gets a key\_t

Calls msgget with key\_t

Creates a msg queue, and returns the id for the message queue

**Read\_Message**

Gets msg queue id

Gets mtype to pay attention for

Gets a pointer to custom message struct to be filled from incoming msg

Calls msgrcv with above parameters

Option is IPC\_NOWAIT (important)

Returns number of bytes written, or -1 for failure

**Send\_message**

Gets msg queue id

Gets a pointer to custom message struct to be sent out to msg queue

Calls msgsnd with params

Returns 0 if success, -1 when msgsnd fails

**Client**

Gets msg queue id

Gets filename

Gets priority

Inits a custom message struct, populate it with

filename, priority, initialization mtype, self PID

Sends to a message queue, an init message with mtype = initialization mtype

Keeps reading from msg queue for mtype = self PID

Print whenever incoming messages fill the buffer

Ends when incoming message has priority = -1

**clientThread**

Client thread function

Gets a message queue ID, what to do with it, TBA

**Server**

Gets msg queue id

Forever loops with msg queue id to look for mtype = initialization mtype

init msg get -> fork a process to be used to deal with init message request

Forked child runs **Server\_transfer\_proc**

**Server\_transfer\_proc**

Gets msg queue id

Gets a copy of a custom message struct to be used when this function sends to msg queue

Tries to read file from filename passed from message struct

Failed to open file

Send msg with error message back to PID specified in msg struct passed in

Return -2

Succeed open file

Keep reading until EOF or Buffer in send message is filled

Send message to PID

Return -1 if send failed

Reset send buffer

When EOF reached, and send message buffer not filled yet

Set send msg struct priority to -1

Send last message to PID

Return -1 if send failed

Close file

Return from function

**Main**

Reads from STDIN for options

if type == server

Run server function

if type == client

Requires filename + priority to be specified in args

Run client function

# Priority Design

Priority will be given to clients depending on number assigned

* Larger priority number param for STDIN will signal to the server to allocate less bytes per msg
* at MAX\_PRIORITY the server will attempt to send a message with a body of 4096 bytes, including null-terminated character
* The size of the message is determined by **MAX\_PRIORITY / priority from STDIN**
* eg. if the -p option was 1024
  + Then the server will send **4096/1024 bytes per message**
* Clients will expect the same size from the server per message, and will wait until it has received MAX\_PRIORITY bytes to print to the console to tell the user a “full” block of data has arrived

# Unique ID design

For the initial mtype that is observed by the Server for initial Client requests, it will be both known by the Server and Client, with it being whatever is **/proc/sys/kernel/pid\_max + 500**. For this application, dynamic pid\_max querying is not implemented yet, so a value of **32768 + 500** is used instead.

This is to make sure no process will ever be the same as this initial mtype, to avoid init messages colliding with mtype = Client PID messages.

# End messge design

The last message sent by the server to denote to the client that the file request has been completed will be inputted to the priority param of the Mesg struct, with it given a value of -1

typedef struct Mesg

{

long mtype;

int mesg\_len;

int pid;

int mesg\_priority;

char mesg\_data[MAXMESSAGEDATA];

} Mesg;