File Transfer

COMP4981

Assignment2

A00907822

Moon Eunwon

DESIGN

Structure of message

```
typedef struct my_msg {
        long type;
        int length;
        int sender_id;
        char msg[MAXMSGDATA];
         int end_flag;
}qMsg;
```

Type : when receive message from queue, use to identify a certain message to receive.

- Server -> Client
  - use client pid to send back to client.
- Client -> Server
  - Client read priority from standard input, and use for a type of structure
  - Server defined PRIORITY as a '-5' to get a lowest number of type message first between 1 and 5.

Length : the msg length in the structure.

- Client define length by calculating use priority value.
  - MAXMSGDATA – 800*(priority-1)
  - If choose priority 1, the length will be MAXMSGDATA(4080)
  - Else the length will be shorter depending on the priority
- Server use this length when reading each filestream to make message.

Sender_id: the sender side process id.

- Server id : Client will send back to kill the process when client interrupted and quit the program while receiving message.'
- Client id : Sender will be use this when return file stream or error message to client as a type value

Msg : the message to communicate

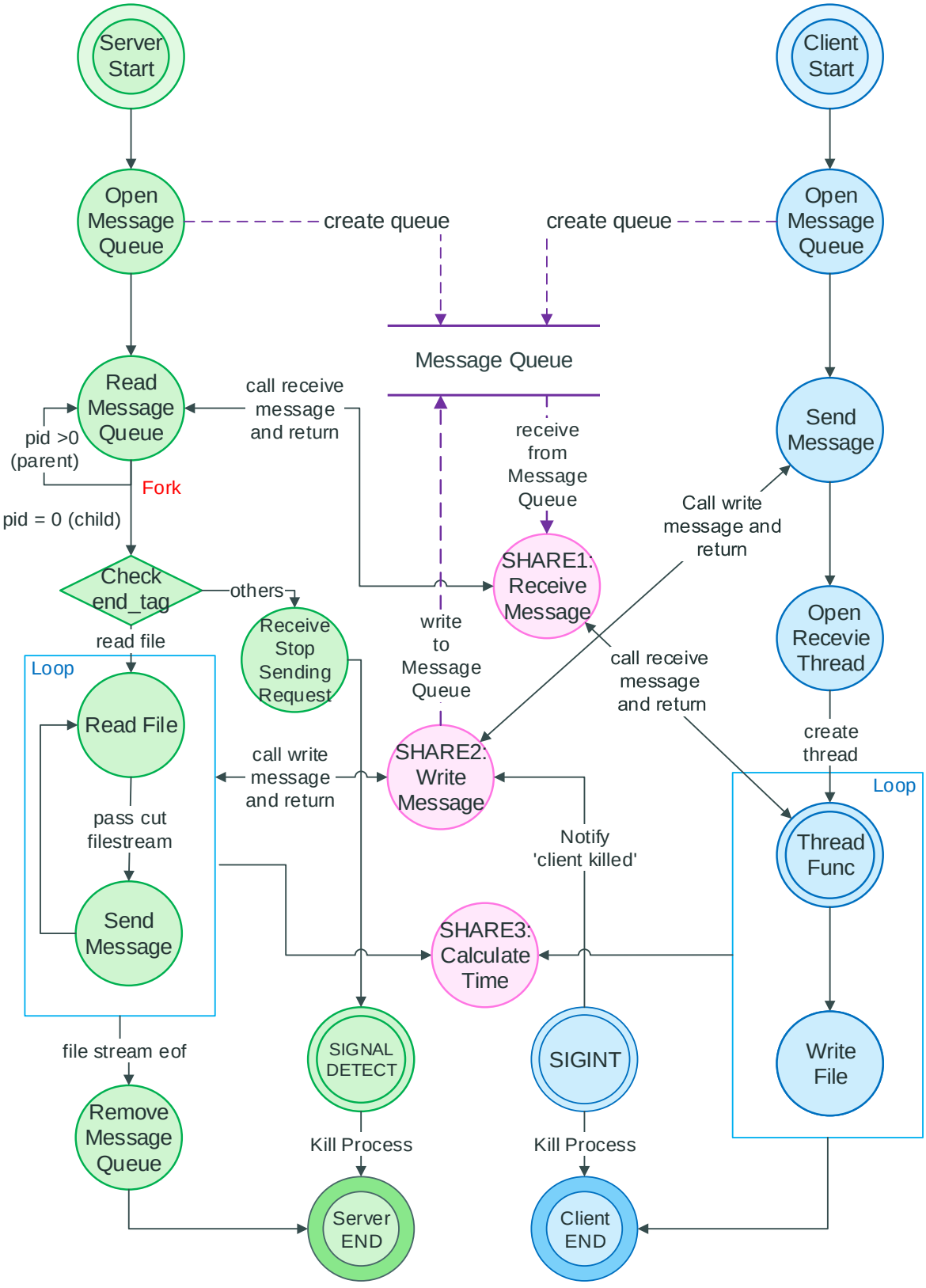
- Server -> Client
  : the file stream that client request to send, or send error message if it is not existing
- Client -> Server
  : send filename that client want to receive.

end_flag : the flag to help communication

- 0 : there is more message to send
- 1(Default): This is last message to send or client send filename.
- 2: Error message to notify non-existing file.
- Other : server process id to request stop sending message.

DESIGN

State Diagram



State Diagram Detail [Share]

SHARE1 – Receive Message

Receive message from Message queue using 'msgrcv' function with queue id, type and message structure. Depending on the 'type' of message, receive different message in the queue. Using point type of the structure, pass the received value by connecting reference. When quite the program return received message byte.

SHARE2 – Write Message

Send message to Message queue using 'msgsnd' function. The parameter is queue id, type and message structure pointer. Using queue id find message 'type' in that message queue. This function return the result of msgsnd.

SHARE3 – Calculate Time

Calculate time spending to transfer file. For sending side, check begin time just before start sending first stream of file, and check end time right after sending last one. On the contrary, the client side check the time when receive first packet and after receiving last one. Last one can be aware using 'end_flag' in the structure.

DESIGN

Psudocode

[Share1]

Function recv_msg(qid, type_val , message_struct pointer)

In qid message queue, read message which type is 'type_val'.

Save in new message structure and link the structure's address to struct pointer

Return result of msgrcv which is byte of reading message.

[Share2]

Function send_msg(qid, message_struct pointer)

Send message structure to message queue - identifier is specified by qid.

Return result of msgsnd.

[Share3]

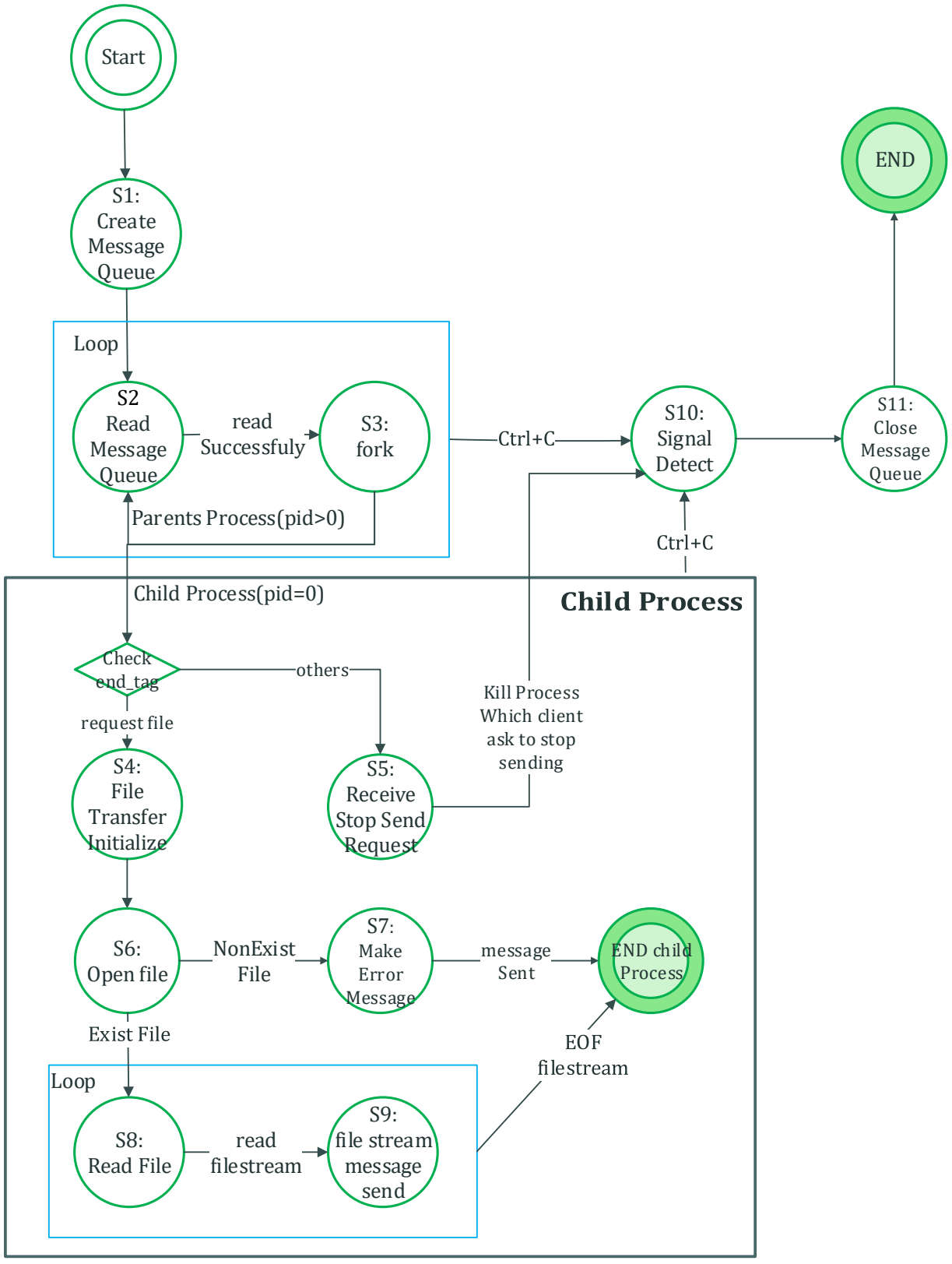
Function Calculate_Time(begin , end)

Duration of transfer time is End time – Begin time.

Print time duration.

DESIGN

State Diagram (SERVER)



State Diagram Detail (SERVER)

S1 – Create message queue

Create message queue using argument using msgget and get message queue identifier.

S2 – Read message queue

Read message queue by calling 'mesg_rcv' function with message queue id. When receiving message, user type '-5' which means to get lowest priority message first which type is between 1 to 5.

S3 – fork child processes

If the message is received successfully, fork process and check the 'end_flag' of the received message in the child process. If it is 1, execute cmove to file transfer function *[S4]*.
Otherwise, execute stop sending request *[S5].*
The parent process keep read message queue and fork *[S3].*

S4 – File transfer initialize

Before transfer file, read message which is received from queue in S2 state, and make a new my_msg structure type message to return message to client. For example, new message type is the received message sender_id, new message sender id is current process id.

S5 – Receive stop sending request

If received message's end_tag is not 1, it is client pid which request stop sending. The client quit the program while receiving message from server because of interrupting. Therefore, send message of server-side child process id to kill the processor to block sending and keep receiving all left messages in message queue using client id.

S6 – Open file

Using filename which is stored received message variable, open file. If the file is open without error, move to Read file *(S7)*. Otherwise move to Send error message state *(S6)* to notify error to client.

S7 – Make error message

Write error message on new message msg part, and pass structure to 'mesg_snd' function. When send message set end_flag to 2.

S8 – Read file

Read the stream of the file. The length of each file stream read is as much as the 'length' value in client message structure. The file stream will store in new message 'msg' variable. If a file stream is last one, change 'end_flag' to notify last part of this file and pass the message structure to 'mesg_send' function to send.

DESIGN

S9 – Send file stream message

When a file stream is read, pass the message structure to 'mesg_send' function to send. Right after sending both first message and last message, check the time to calculate transfer time using function. *[Share3]*

S10 – Signal Detect

Signal is defined when start this program. When user press 'Ctrl+C', detect the signal and kill all child processes and message queue. Finally terminate program.
Also when client request stop *[S5]* detect signal by killing the process using pid number and SIGKILL.

S11 – Close message queue

When finishing the program or killing all processes by detecting signal, remove everything in the queue and quit it. Closing queue is only possible at the server.

DESIGN

Psudocode (SERVER)

[S1]

Function **main**

Create message queue using current directory
If there is error, display error.
Otherwise, read queue.

[S2]

* Loop S2, S3 - Parent queue is keep reading message queue and fork process to send back.

Function **read_queue**

Create structure and malloc memory
Call receive function with message queue id and message structure pointer***[SHARE1].***
[S3]
If queue is read, fork process.
For child process: check end_flag of received message structure
If end_flag is 1, do file_transfer function to send back to client ***[S4]***
Otherwise, do stop_request to block send message to that client***[S5]***
parent process keep reading message queue***[S2]***
At the end of read_queue free the memory.

* For reading queue in parent process, type is '-5' for priority which means read between 1 and 5 from smallest number and early coming number.

[S5]

Function **Stop_request**

Make signal to kill process (which is received from client)
Define temp message structure
[loop] keep reading message using client id in received message until nothing left to read.

DESIGN

[S4]

Function **file_trans**

Make new structure for return message

initialize new message using received message or default value.

Change file name to file path

[S6]

Open file using modifying filename :fread().

[S7]

If file cannot open,

change end_flag to 2 and add error message in msg field.

Call send_message function***[SHARE2].***

Quit file_trans function

[S8]

* Loop S8 and S9 front keep reading until end of the file.

Else,

keep reading file stream from the open file. : fread

*Each message size is the length in received message.

If the message is last stream of file(check feof or shorter than reading stream length)

change new message end_flag to 1.

[S9]

Each time read message from the file, store in the structure msg portion,

and call send_message function. *[Share2]*

If send all file, print complete message and close file.

[S10]

Function sig_handler

If signal is SIGINT

If process is parent,

Keep wait until nothing left and close queue*[S11]*

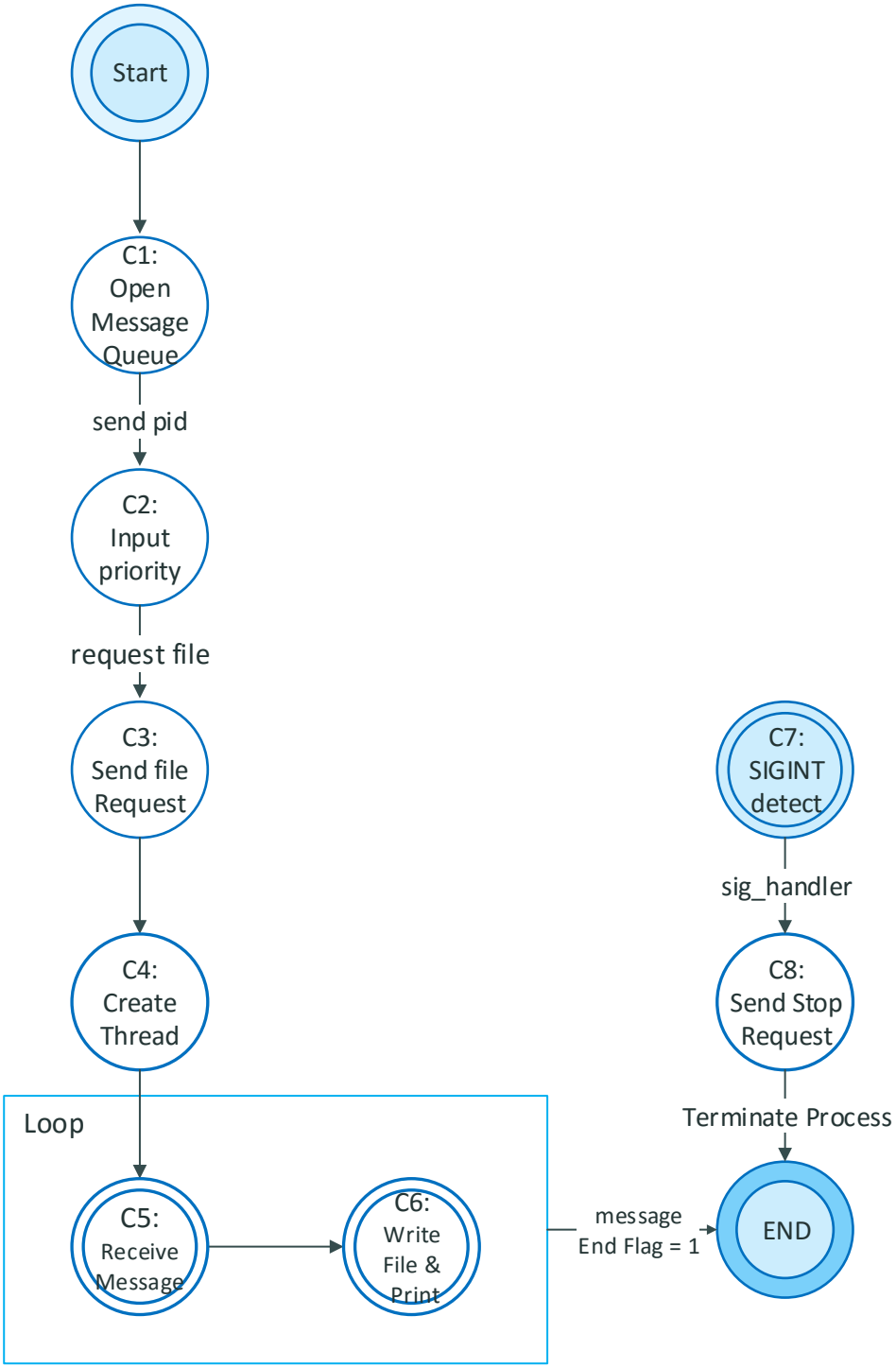
If not process signal initialize to null and kill this process using SIGTERM

[S11]

Function close_queue

Remove queue from the kernel using msgctl function : IPM_RMID

State Diagram (Client)



State Diagram Detail (Client)

C1 –Open message queue

Check if there is not enough argument in command line, display error and quit program.
Otherwise, create message queue using current directory or open existing queue.
If there is error, display error.
Otherwise move to the state related to request file [C2, S3]

C2 – Standard Input for priority

Read standard input to get priority.

C3 – Create message and send file request

Using input information and default value(ex: flag) make a message with process id, and send message to client. Priority is using as a type.

C4 – Create Thread

Before receiving message, create send to receive message.

Create thread information structure and put file name to create file using receiving data.

C5 – Receive message (Thread)

Make new file name using original name to store received file and open file to write.

If there is no error, keep reading message using receive_message function and pass the message to C6 to write and print message until get end flag(1) or error flag(2).

If finish receive message close file stream and if receive error flag delete file.

C6 – Write file and Print(Thread)

If reading anything, write on the file and print.

C7 – SIGNAL detect (Ctrl+C)

Signal is defined when start this program. When user press 'Ctrl+C', detect the signal and kill the process. Before killing process, send 'stop sending request'*[C8]*

C8 – Send stop request

When signal detected, client read 1 more message to get information of communication, and send 'stop sending request' to prevent blocking message queue .

DESIGN

Psudocode

Function **main**

Check argument contain filename,
if not print error message, quite program
[C1]
Otherwise, create message queue using current directory and return message queue identification number. Also check error of creating queue.
Without error
call file request file_request function*[C2]*
call msg_rcv function *[C4]*

Function **file_request**

Initialize message queue structure and memory allocate.
[C2]
Ask priority level and read standard input .
Calculate length of message using priority value.
Set the other message structure value to request file.
[C3]
Send message using send_msg function*[Share2]*
Free message queue structure.

* Priority input value should be between 1 and 5.

[C4]

Function **msg_rcv**

Create thread with threadfunc method and filename
Join thread

Function **ThreadFunc**

Initialize message queue structure and memory allocate.
Modify filename : cp_filename
Open file using filename to write in binary : fopen(wb+)
[C5] *** loop C5, C6 until end of the message**
Read message if end_flag is 0 or nothing read.
If it is first message, set begin time.
[C6]
If message is read, write the file and print it
Set end time finish read
Close file stream.

If end_flag =2, delete created file
Otherwise print total packet and call timegap function to check transfer time
Free message queue buffer memory.

DESIGN

[C7]

Function **sig_cl_handler**

Initialize and memory allocate for message structure
Read message using recv_msg from the queue [Share1]
change type to 1 (high priority)
change end_flag value to sender_id
change sender_id to getpid()

initialize msg

send message to server : send_msg[Share2]

set signal to null
kill process and terminate program