GTU Department of Computer Engineering
CSE 344 – Spring 2023
Midterm Project Report

Hasan Mutlu
1801042673

Implementation Details and General Work Flow

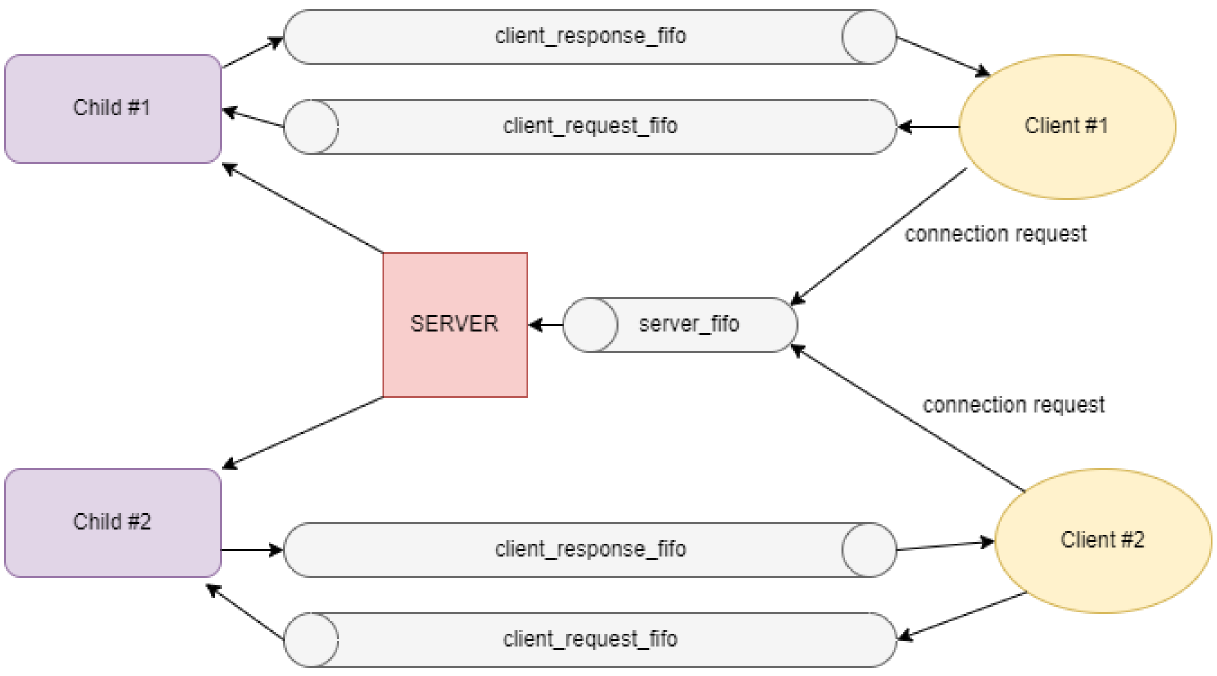
The server-client program make use of a lot of different features in order to satisfy the needs. FIFO's, semaphores and shared memory is used.

To generally describe the work flow of the program, firstly, the server starts after checking the inputs and reserving necessary resources. After that, server waits for clients to send connection requests. When a request arrives, if there is an available slot, creates a new child and forwards this client to this new process. If there isn't an available slot, puts the request on queue or rejects it if it calls with try.

When client is connected to a child process, they start sending messages back and forward. Clients sends a request message and the child process handles it until client sends quit request. After that, both processes free their resources and terminate.

The Queue data structure is taken as external source and little modifications are made.

Inter-Process Communication



IPC is established using **FIFO**'s with the logic above. All messages are sent as fixed size in order to keep a clean connection. There is one server fifo that accepts the requests from clients. Clients sent their PIDs followed by their connection request (Connect or tryConnect). According to the request, server puts the client's PID in queue or informs that the queue is full, if they request with tryConnect.

After that, if there is an available place, the server pops the next PID from the queue and forks a child process. This child process then sends “connect” message to the client using client_response_fifo. After that, they exchange messages. Client sends one request message to child using client_request_fifo and the child sends the output messages using client_response_fifo, followed by a terminating message.

Every process is responsible for creating and unlinking the FIFO that they get their messages from. Server is responsible from server_fifo, child is responsible from client_request_fifo and the client is responsible from client_response_fifo.

Synchronization

In order to synchronize the processes, **semaphores** are used. Busy waiting is strictly avoided. The use of semaphores is parallell with the use of FIFO’s.

There are 3 types of semaphores: server semaphore, client response semaphore and client request semaphore. All semaphores are initilized with the value of 0. Firstly, the server waits on server semaphore to accept and redirect requests. Whenever a client wants to connect, or a child process is terminated, they post the server semaphore to wake server up. Server then proceeds to accept connections or connect a client on the queue to a child process.

After child process is assigned to a client, it writes “connected” message to the client and posts client response semaphore. Then, proceeds to wait on client request semaphore. The client, waiting on the client response semaphore after being put in queue, gets a command from the user, and writes the message to the child process and posts the request semaphore to wake child up. This process continues, each process is wake each other up after writing their message and be blocked while waiting for a response. By this way, the CPU time is used as efficient as possible.

Worker Management

In order to manage the availibility of the child processes, **shared memory** is used by server. One integer value “count” is shared to keep the number of active workers, and one array, “worker list” is shared to keep track of the PID’s of the workers.

When deciding whether to fork a new child or put the client in the queue, the counter variable is used. It is incremented whenever a new child is created and decremented when a child is done with the client. The active child’s PID’s are also saved and deleted to/from the worker list in a parallel manner.

Signal Handling

The server handles two kinds of signals, SIGINT and SIGUSR1. SIGUSR1 is only received from clients to terminate the server, while SIGINT can be received from command line.

When the client wants to terminate the server, it sends SIGUSR1 to the server. This way, server can know what is the source of the termination. Then, using worker list, sends SIGINT to its active children, and sends a SIGINT to itself.

When server receives SIGINT, it pops the clients on the queue and sends them SIGINT, to make them not wait on a terminated server. If the child receives SIGINT, it sends SIGINT to the client it has been currently handling to inform about the termination of the server.

After sending and receiving all necessary signals to the from/to clients, all processes free their resources and exit.

File Handling

The file handling functions are implemented in the "utils.h" file. When the child receives a request from the server, it first parses and checks the validity of the request, and then call the functions defined on utils.h file to handle the request. The race conditions to write the same file between processes are avoided by using flock().

Requirements

As far as the program is tested, all requirements related to general workflow of such client-server model is working fine. The communication, synchronization and services are satisfied. The resources used efficiently and freed when terminated. No junk files are left behind.

Test Plan

The test plan are made and run under the following titles:

- ➔ IPC:
  - Testing the FIFO and shared memory communication between the three processes.
  - Testing the client/queue management.

First testings are done with busy waiting, before implementing synchronization module.

- ➔ Synchronization:
  - Testing the presence of busy waits.
  - Testing the use of waits of posts, blocking situations.

- ➔ Signals
  - Testing SIGINT signals from command line.
  - Testing killServer signal from a client.

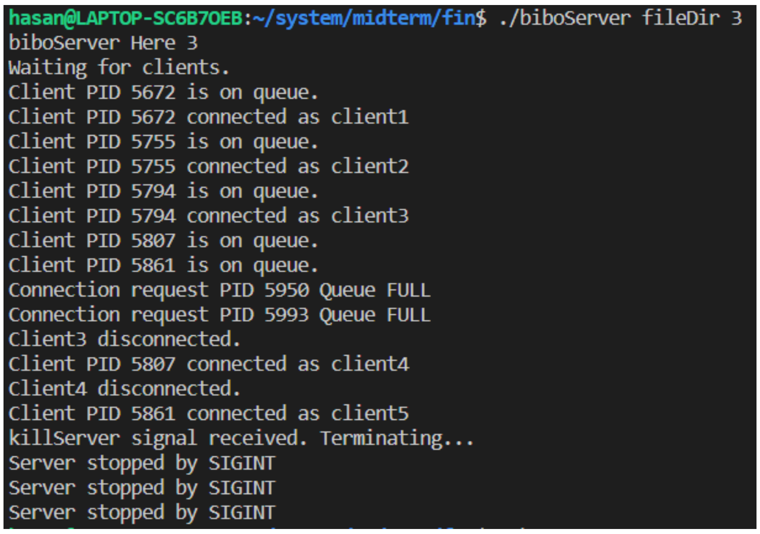
- ➔ Cleanup
  - Testing if there is any junk files left from the fifos.

- ➔ File Operations
  - Testing the validity of operations.
  - Testing exception situations.
  - Testing opening different directories.
  - Testing race conditions, writing to same file at the same time.

Test Results

The program is tested with several number of childs. Here is an example of using 3 childs.

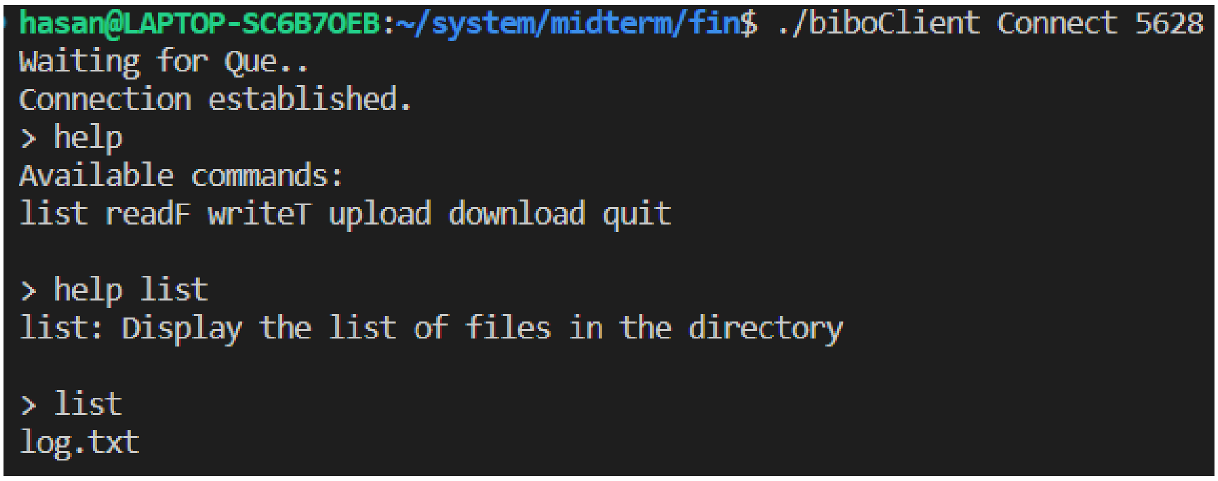
Server Output



```
hasan@LAPTOP-SC6B7OEB:~/system/midterm/fin$ ./biboServer fileDir 3
biboServer Here 3
Waiting for clients.
Client PID 5672 is on queue.
Client PID 5672 connected as client1
Client PID 5755 is on queue.
Client PID 5755 connected as client2
Client PID 5794 is on queue.
Client PID 5794 connected as client3
Client PID 5807 is on queue.
Client PID 5861 is on queue.
Connection request PID 5950 Queue FULL
Connection request PID 5993 Queue FULL
Client3 disconnected.
Client PID 5807 connected as client4
Client4 disconnected.
Client PID 5861 connected as client5
killServer signal received. Terminating...
Server stopped by SIGINT
Server stopped by SIGINT
Server stopped by SIGINT
```

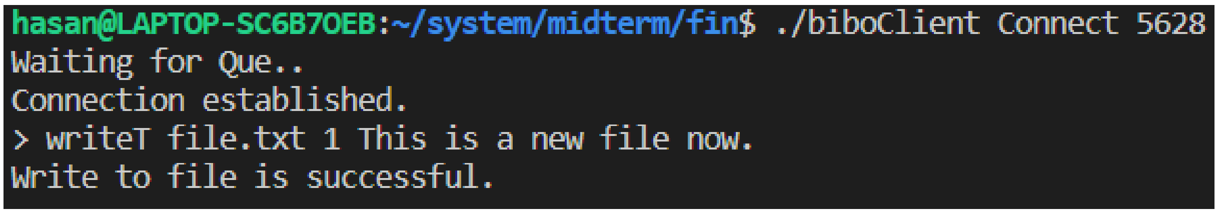
2 clients are put on queue and 2 clients are returned after a call in a full queue with tryConnect.

Client Outputs



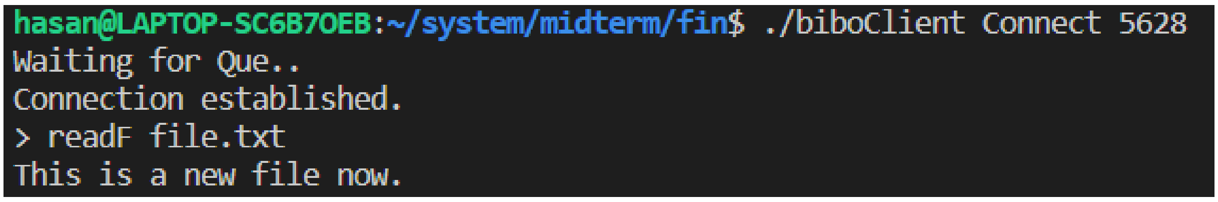
```
hasan@LAPTOP-SC6B7OEB:~/system/midterm/fin$ ./biboClient Connect 5628
Waiting for Que..
Connection established.
> help
Available commands:
list readF writeT upload download quit

> help list
list: Display the list of files in the directory

> list
log.txt
```

Help, help <cmd> and list commands are working.



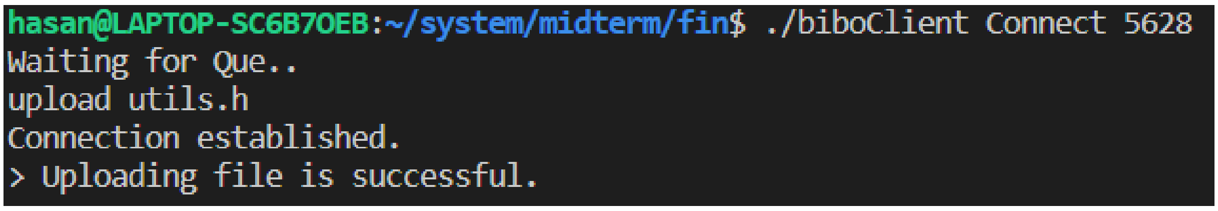
```
hasan@LAPTOP-SC6B7OEB:~/system/midterm/fin$ ./biboClient Connect 5628
Waiting for Que..
Connection established.
> writeT file.txt 1 This is a new file now.
Write to file is successful.
```

writeT, creating a new file.



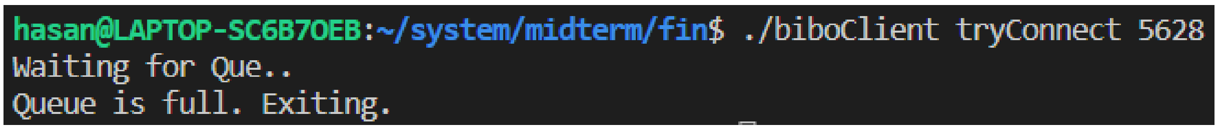
```
hasan@LAPTOP-SC6B7OEB:~/system/midterm/fin$ ./biboClient Connect 5628
Waiting for Que..
Connection established.
> readF file.txt
This is a new file now.
```

Reading from file.



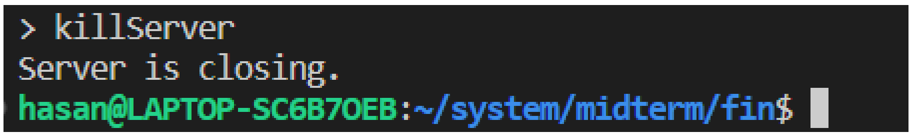
```
hasan@LAPTOP-SC6B7OEB:~/system/midterm/fin$ ./biboClient Connect 5628
Waiting for Que..
upload utils.h
Connection established.
> Uploading file is successful.
```

Uploading file.



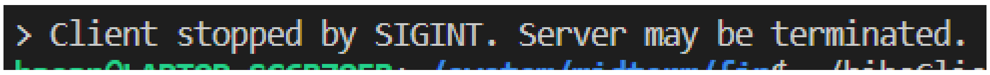
```
hasan@LAPTOP-SC6B7OEB:~/system/midterm/fin$ ./biboClient tryConnect 5628
Waiting for Que..
Queue is full. Exiting.
```

Exiting when queue is full.



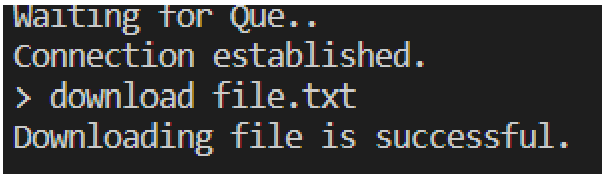
```
> killServer
Server is closing.
hasan@LAPTOP-SC6B7OEB:~/system/midterm/fin$
```

killServer



```
> Client stopped by SIGINT. Server may be terminated.
```

After killServer request is made, all active clients displayed this message and terminated.



```
Waiting for Que..
Connection established.
> download file.txt
Downloading file is successful.
```

Download file. (This is from another test session, isn't present in log file)

Example Log File Output

```
[Wed May 17 04:38:28 2023] SERVER: Directory fileDir is opened.

[Wed May 17 04:38:40 2023] SERVER: Waiting for clients.

[Wed May 17 04:38:40 2023] SERVER: Client PID5672 is on queue.

[Wed May 17 04:38:40 2023] WORKER5673: Serving Client1 with PID5672

[Wed May 17 04:38:50 2023] SERVER: Client PID5755 is on queue.

[Wed May 17 04:38:50 2023] WORKER5756: Serving Client2 with PID5755

[Wed May 17 04:38:53 2023] SERVER: Client PID5794 is on queue.

[Wed May 17 04:38:53 2023] WORKER5795: Serving Client3 with PID5794

[Wed May 17 04:38:54 2023] SERVER: Client PID5807 is on queue.

[Wed May 17 04:38:57 2023] SERVER: Client PID5861 is on queue.

[Wed May 17 04:39:01 2023] SERVER: Client PID5950 tried connect but queue is full.

[Wed May 17 04:39:06 2023] SERVER: Client PID5993 tried connect but queue is full.

[Wed May 17 04:39:09 2023] WORKER5673: Client1 requested: help

[Wed May 17 04:39:09 2023] WORKER5673: Client1 request is handled.

[Wed May 17 04:39:11 2023] WORKER5673: Client1 requested: help list

[Wed May 17 04:39:11 2023] WORKER5673: Client1 request is handled.

[Wed May 17 04:39:12 2023] WORKER5673: Client1 requested: list

[Wed May 17 04:39:12 2023] WORKER5673: Client1 request is handled.

[Wed May 17 04:39:37 2023] WORKER5756: Client2 requested: writeT file.txt 1 This is a new
file now.

[Wed May 17 04:39:37 2023] WORKER5756: Client2 request is handled.

[Wed May 17 04:39:50 2023] WORKER5795: Client3 requested: readF file.txt

[Wed May 17 04:39:50 2023] WORKER5795: Client3 request is handled.

[Wed May 17 04:40:05 2023] WORKER5795: Client3 requested: quit
```

```
[Wed May 17 04:40:05 2023] WORKER5795: Client3 is quiting.

[Wed May 17 04:40:05 2023] WORKER5795: Client3 exited. Terminating..

[Wed May 17 04:40:05 2023] WORKER6232: Serving Client4 with PID5807

[Wed May 17 04:40:05 2023] WORKER6232: Client4 requested: upload utils.h

[Wed May 17 04:40:05 2023] WORKER6232: Client4 request is handled.

[Wed May 17 04:40:16 2023] WORKER6232: Client4 requested: quit

[Wed May 17 04:40:16 2023] WORKER6232: Client4 is quiting.

[Wed May 17 04:40:16 2023] WORKER6232: Client4 exited. Terminating..

[Wed May 17 04:40:16 2023] WORKER6342: Serving Client5 with PID5861

[Wed May 17 04:40:20 2023] SERVER: killServer signal received from a client. Terminating...

[Wed May 17 04:40:20 2023] WORKER5673: SIGINT received. SIGINT sent to client1

[Wed May 17 04:40:20 2023] WORKER5756: SIGINT received. SIGINT sent to client2
```