Gebze Technical University

Computer Engineering Faculty

Midterm Report

System Programming

Student: Burak Ersoy

No: 1901042703

BiboServer

1. Problem Description

The assignment was to implement a biboServer program that can handle multiple client requests using shared memory and semaphores. The program should be able to perform various operations on files such as reading, writing, uploading, and downloading. The program should also be able to list the files in the server directory, provide help messages and terminate gracefully.

2. Discussion

My solution is a server application that uses a shared memory segment and a semaphore to communicate with the clients. The shared memory segment contains a queue of client PIDs, an array of file entries, and a semaphore. The file entries store the file name, size, descriptor, and number of readers and writers for each file in the server directory. The semaphore is used to synchronize access to the shared memory segment.

The server process creates a child process for each client request that is dequeued from the queue. The child process handles the client request by reading and writing to a FIFO that is named after the client PID. The child process performs different operations on files depending on the client request. For example, if the client requests to read a line from a file, the child process reads the line from the file descriptor stored in the shared memory segment and writes it to the FIFO. If the client requests to upload a file, the child process receives the file content from the FIFO and writes it to a new file in the server directory.

The server process also handles signals from clients or children. If a client sends a SIGUSR1 signal, the server process enqueues its PID to the queue. If a child sends a SIGCHLD signal, the server process reaps it when it terminates. If the server receives a SIGINT signal (Ctrl-C), it cleans up the shared memory segment and semaphore, kills all child processes, and exits gracefully.

BiboClient

1. Problem Description

The assignment was to implement a client program that can connect to a server queue and perform various file operations such as reading, writing, uploading, and downloading. The program should also be able to list the files in the server directory, provide help messages and terminate gracefully.

2. Discussion

My solution is a client application that uses signals and FIFOs to communicate with the server. The client can choose to connect or tryConnect to the server queue by sending a SIGUSR1 signal to the server PID. The server responds by sending a SIGUSR1 signal to confirm the connection or a SIGUSR2 signal to reject the connection. The client sets a flag to indicate if it is connected or not.

The client can enter different commands to request file operations from the server. The commands are: help, list, readF, writeT, upload, download, quit, killServer. The client sends the command to the server by writing it to a FIFO that is named after the client PID. The server reads the command from the FIFO and performs the corresponding operation.

The client receives the response from the server by reading from the same FIFO. The response can be a message, a file content, or an error. The client writes the response to standard output. If the response is a file content, the client can save it to a local file.

The client can terminate by entering quit or killServer commands. The quit command tells the server to close the FIFO and remove the client PID from the queue. The killServer command tells the server to clean up the shared memory segment and semaphore, kill all child processes and exit.

The client also handles signals from user or server. If Ctrl-C signal is received, the client prints a message and exits gracefully. If SIGUSR1 signal is received from server, the client sets connected flag to true and prints a message. If SIGUSR2 signal is received from server, the client sets connected flag to false and prints a message.

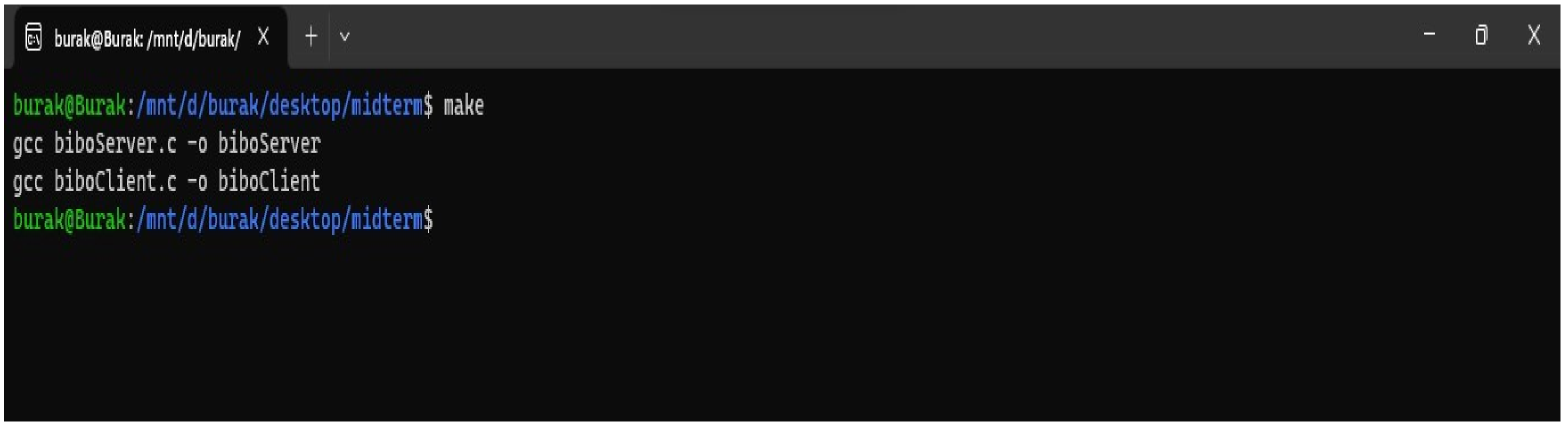
Lessons Learned

From this assignment, I learned:

- How to use shared memory and semaphores for interprocess communication.
- How to use FIFOs for bidirectional communication between processes.
- How to handle signals from processes using signal handlers.
- How to implement basic file operations such as reading, writing, uploading, and downloading.

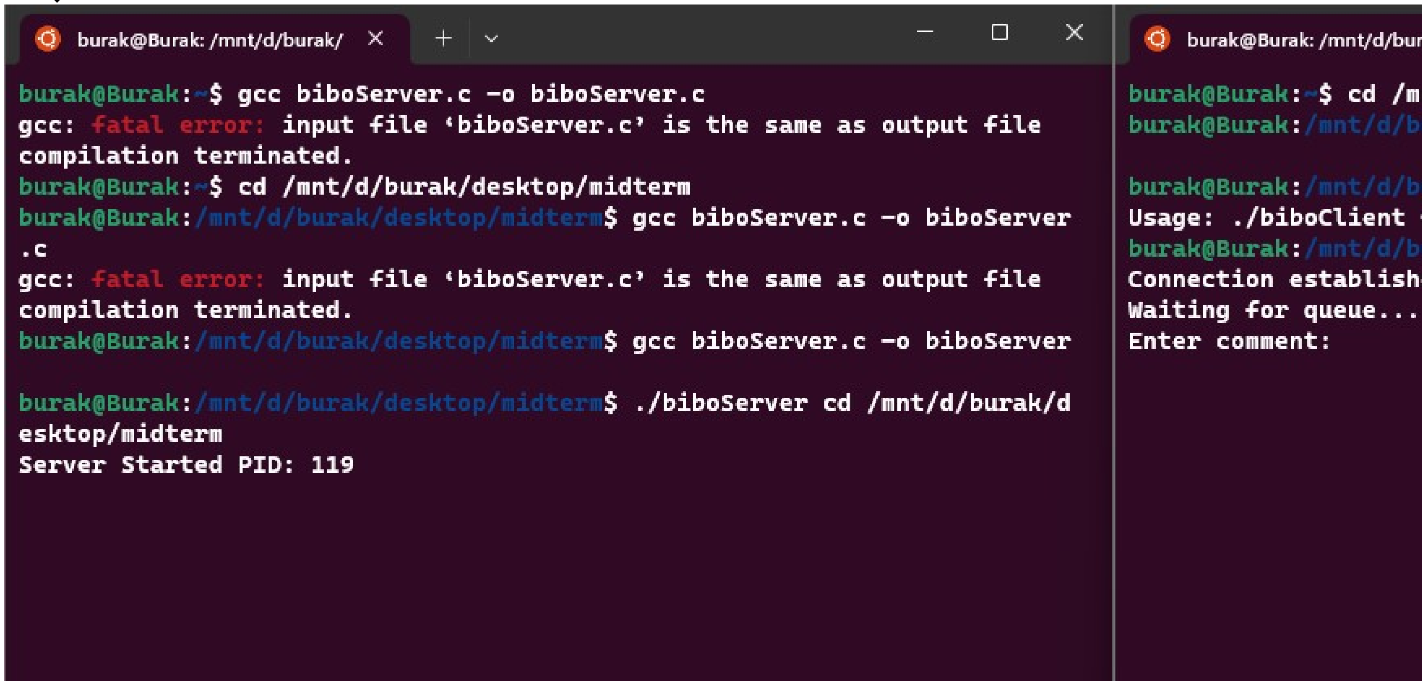
Test Cases:

❖Makefile



```
burak@Burak:/mnt/d/burak/desktop/midterm$ make
gcc biboServer.c -o biboServer
gcc biboClient.c -o biboClient
burak@Burak:/mnt/d/burak/desktop/midterm$
```

❖



```
burak@Burak:~$ gcc biboServer.c -o biboServer.c
gcc: fatal error: input file 'biboServer.c' is the same as output file
compilation terminated.
burak@Burak:~$ cd /mnt/d/burak/desktop/midterm
burak@Burak:/mnt/d/burak/desktop/midterm$ gcc biboServer.c -o biboServer
.c
gcc: fatal error: input file 'biboServer.c' is the same as output file
compilation terminated.
burak@Burak:/mnt/d/burak/desktop/midterm$ gcc biboServer.c -o biboServer

burak@Burak:/mnt/d/burak/desktop/midterm$ ./biboServer cd /mnt/d/burak/d
esktop/midterm
Server Started PID: 119
```

```
burak@Burak:~$ cd /m
burak@Burak:/mnt/d/b

burak@Burak:/mnt/d/b
Usage: ./biboClient
burak@Burak:/mnt/d/b
Connection establish
Waiting for queue...
Enter comment:
```