GEBZE
TECHNICAL UNIVERSITY

CSE344 – System Programming

Midterm – Report

Ömer Faruk Bitikçioğlu

161044010

Client

Client-side implementation of the project. It allows users to connect to a server, send commands to the server, and receive responses from the server. The client and server communicate with each other using FIFOs (named pipes).

The biboClient program takes two arguments - the first argument specifies the type of connection (either "Connect" or "tryConnect"), and the second argument is the process ID (PID) of the server process.

The main function checks the number and validity of the command-line arguments and sets up a signal handler for SIGINT (the signal sent by the CTRL+C command). It then extracts the server PID from the second argument and generates the server FIFO name using a template.

The connectServer function sends a connection request to the server by creating a request structure containing the client's PID and the requested command type ("Connect" or "tryConnect") and writes it to the server FIFO.

If the connection is successful, the connectionEstablished function is called. This function runs in a loop, reads the user's input command from stdin, sends it to the server using the client's FIFO for requests, reads the server's response from the client's FIFO for responses, prints the response to stdout, and checks if the user has decided to quit or kill the connection. If the user chooses to quit or kill, the loop is exited, and the client FIFOs are unlinked.

The signint_handler function is the signal handler for CTRL+C and is responsible for unlinking the client FIFOs and killing the client process if the connection is terminated abruptly.

The sendRequest function writes a request structure to a FIFO, and the getResponse function reads a response structure from a FIFO.

Server

The server process listens for client connection requests and serves their commands. It supports multiple concurrent clients using a FIFO (named pipe) communication mechanism. Each client is assigned a separate child process to handle its commands.

The server interacts with the file system to perform various file operations requested by clients. It maintains a queue of clients waiting to connect when the server reaches the maximum allowed connections. Once connected, each client communicates with the server using its own FIFOs for request and response messages.

Design Decisions

Inter-Process Communication: The server application utilizes FIFOs (named pipes) for communication between the main process and child processes, as well as between child processes and clients. This choice allows for reliable and synchronous communication, ensuring that commands and responses are properly transmitted between the components.

Concurrency and Scalability: By creating a separate child process for each client connection, the server application achieves concurrency and scalability. Multiple clients can be served simultaneously, and the server can handle a high number of connections without blocking or slowing down.

Client Queue: To manage client connections when the maximum number of connections is reached, a client queue data structure is used. This allows for orderly handling of incoming connections, ensuring fairness and preventing resource exhaustion. When a child process becomes available, it dequeues the next client from the queue and starts serving it.

Locking Mechanism: When reading from or writing to a file, a shared lock is acquired to ensure data integrity and prevent conflicts when multiple processes attempt to access the same file simultaneously. This locking mechanism enables safe file operations and avoids race conditions.

Signal Handling: The server application includes a signal handler for the SIGINT signal, which is generated when the user presses Ctrl+C. When this signal is received, the server unlinks the server FIFO, terminates all child processes, and gracefully shuts down. This ensures that resources are properly released and prevents any lingering processes or orphaned FIFOs.

Implementation Details

signint_handler(): Signal handler for the SIGINT signal (generated by pressing Ctrl+C). It unlinks the server FIFO and terminates the server and all child processes.

sendResponseToClient(): Sends the given response structure to the client through the client's response FIFO.

processRequestList(): Processes the "list" command by reading the files in the specified directory and concatenating their names into a response message.

processReadFile(): Reads the contents of a file specified by the given filename. It acquires a shared lock on the file, reads either a specific line or the entire content, and releases the lock.

processWriteToFile(): Writes the given string to a specific line or the end of a file specified by the filename. It acquires a shared lock on the file, reads the existing content, modifies it based on the line number, and writes the updated content back to the file.

processRequestKillServer(): Handles the "killServer" command from the client. It sends a response to the client, unlinks the server FIFO, and terminates the server and all child processes.

serveClient(): Handles the commands requested by a specific client. It reads commands from the client's request FIFO, executes the corresponding functions, prepares a response, and sends it back to the client.

main(): The main function of the server application. It initializes signal handling, sets up the server FIFO, creates the client queue, and manages client connections. It listens for connection requests, creates child processes to handle connections, and enqueues clients if the maximum number of connections is reached.

Client Queue (client_queue.h)

“client_queue.h” defines a client queue data structure and provides functions for managing the queue. Here's an overview of the code:

The client_node_t struct represents a node in the client queue and contains a pointer to a client object (client_t) and a pointer to the next node in the queue.

The client_queue_t struct represents the client queue and contains pointers to the head and tail nodes of the queue.

The newNode function is a utility function that creates a new client node and initializes its client object (client_t). It allocates memory for the client object, sets its pid field, and dynamically allocates memory for the client's read and write FIFO names.

The createQueue function creates an empty client queue by allocating memory for the client_queue_t struct and setting its head and tail pointers to NULL.

The enqueueClient function adds a client to the tail of the queue. It creates a new client node, sets its client object, and adds it to the tail of the queue by updating the tail pointer accordingly.

The dequeueClient function removes and returns the first client from the queue. It checks if the queue is empty, removes the first node from the queue, updates the head pointer, and returns the client object from the removed node. The function also frees the memory allocated for the removed node.

The isClientInQueue function checks if a client with a given cl_pid exists in the queue. It iterates over the nodes in the queue, compares the pid field of each client, and returns 1 if a match is found (indicating that the client is in the queue) or 0 if no match is found.

Client.h

The client_t struct in “client.h” represents a client and contains the following fields:

- id: An integer value representing the client's ID.
- pid: A process ID (pid_t) representing the client's process ID.
- fifo_req_name: A pointer to a character array (char*) representing the name of the client's request FIFO (First-In-First-Out) pipe.
- fifo_res_name: A pointer to a character array (char*) representing the name of the client's response FIFO pipe.

Bibo_cl_sv.h

The code in "bibo_cl_sv.h" header file contains various constants and structure definitions related to the client-server communication in the system. Here's an overview of the code:

The header file defines several constants for naming conventions and lengths of FIFO (First-In-First-Out) pipe names and log file names.

It also defines the maximum length of a response message (MAX_RESPONSE_LEN), which is set to 1024 characters (1 KB).

The request_t struct represents a client request and contains the following fields:

- cmd: An array of characters (char) representing the command string of the request.
- cl_pid: A process ID (pid_t) representing the client's process ID.

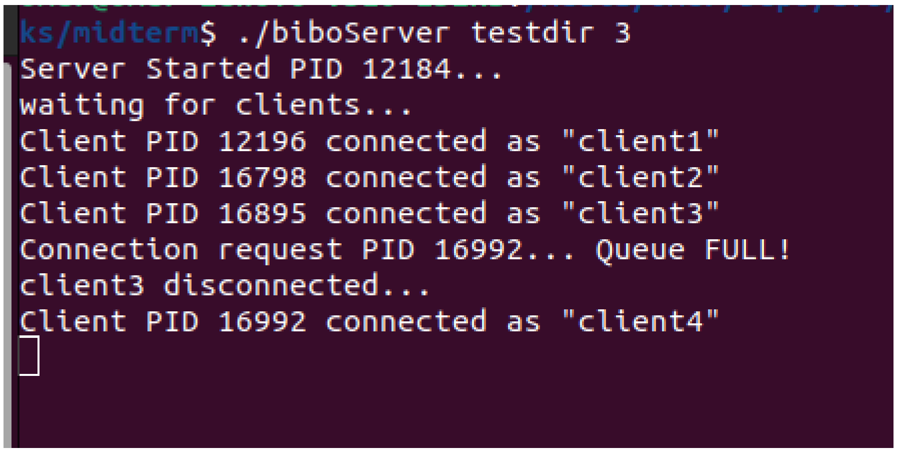
The response_code_e enumeration defines three possible response codes: OK, WAIT, and FAIL.

The response_t struct represents a server response and contains the following fields:

- msg: An array of characters (char) representing the response message.
- code: A member of the response_code_e enumeration representing the response code.

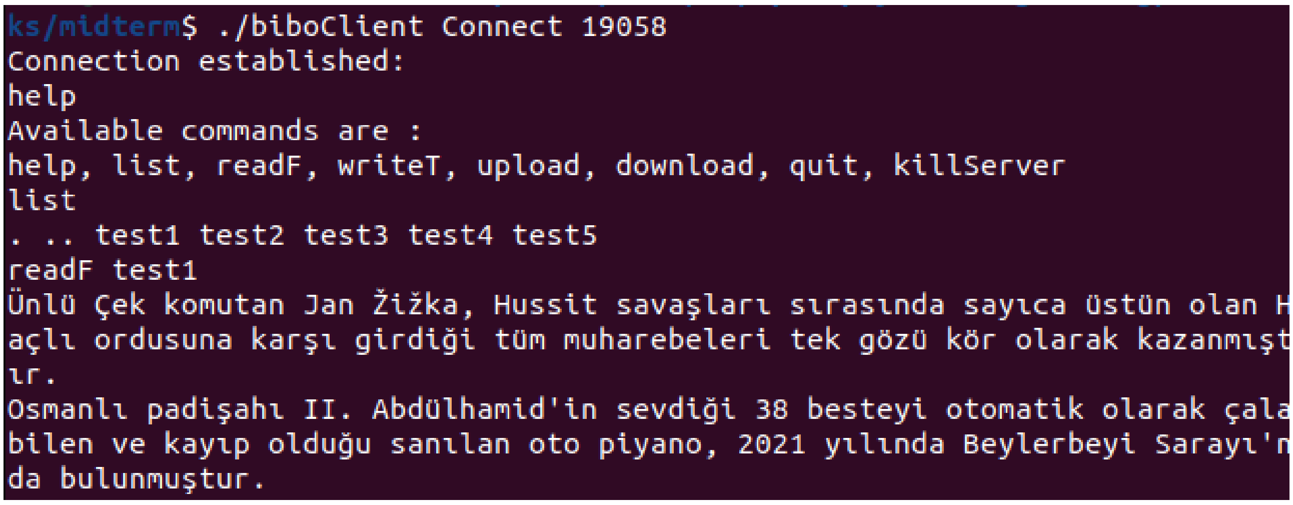
Test & Results

Server execution flow:



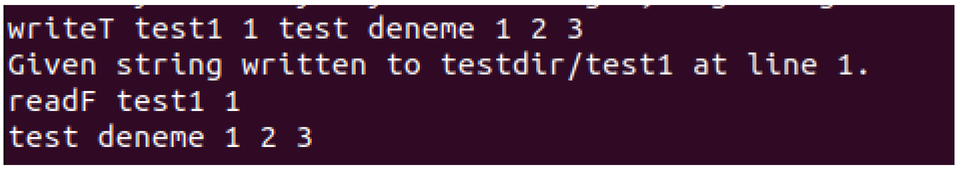
```
ks/midterm$ ./biboServer testdir 3
Server Started PID 12184...
waiting for clients...
Client PID 12196 connected as "client1"
Client PID 16798 connected as "client2"
Client PID 16895 connected as "client3"
Connection request PID 16992... Queue FULL!
client3 disconnected...
Client PID 16992 connected as "client4"
```

Server waits for client connections. If the connection limit is reached, new connection requests are queued. Whenever a connected client quits, the queued clients connect to the server.



```
ks/midterm$ ./biboClient Connect 19058
Connection established:
help
Available commands are :
help, list, readF, writeT, upload, download, quit, killServer
list
. .. test1 test2 test3 test4 test5
readF test1
Ünlü Çek komutan Jan Žižka, Hussit savaşları sırasında sayıca üstün olan H
açlı ordusuna karşı girdiği tüm muharebeleri tek gözü kör olarak kazanmışt
ır.
Osmanlı padişahı II. Abdülhamid'in sevdiği 38 besteyi otomatik olarak çala
bilen ve kayıp olduğu sanılan oto piyano, 2021 yılında Beylerbeyi Sarayı'n
da bulunmuştur.
```

Client can use help, list, readF, writeT, quit and killServer options. Upload and download is not implemented in this project.



```
writeT test1 1 test deneme 1 2 3
Given string written to testdir/test1 at line 1.
readF test1 1
test deneme 1 2 3
```

Quit option kills the client process. KillServer option kills both the client and server processes.

KillServer also kills all the child server processes, but do not kill the clients other than the client sent the killServer request.

If client process is terminated using ctrl+c, server is not decreasing connection count.