BLG312E Operating Systems Assignment 3

Abdullah Shamout
School of Computer Engineering
Istanbul Technical University
150200919
Email: shamout21@itu.edu.tr

***Abstract*—This report presents the implementation details of a multi-threaded web server written in C. The server is designed to handle multiple client requests concurrently using threads.**

I. Introduction

The objective is to create a multi-threaded web server that can handle multiple client requests simultaneously. The server should:

- Accept a specified number of threads to handle incoming connections.
- Use a buffer to store incoming connection file descriptors.
- Use semaphores to synchronize access to shared resources.
- Handle server shutdown to ensure no memory leaks or unfinished tasks.

I only needed to edit the server.c file and add a multi-threaded request example to my client.c to test it

II. Implementation Details

A. Server Implementation (server.c)

The server implementation includes the following key functions and data structures:

```
struct THREADS_TYPE {
    pthread_t *threads;
    int number_of_threads;
};
```

Listing 1. Thread Data Structure

The THREADS_TYPE structure holds information about the threads used in the server. It contains a pointer to an array of pthread_t representing the threads and an integer number_of_threads indicating the total number of threads.

```
STRUCTURE THREADS_TYPE
DECLARE threads_instance AS THREADS_TYPE
DECLARE connection_file_descriptors
AS POINTER TO INTEGER
SET connection_file_descriptors TO NULL
DECLARE semaphore num_of_threads
DECLARE semaphore connection_top
DECLARE semaphore buffer_mutex
DECLARE INTEGER buffer_size_G
```

Listing 2. Variables

Here, I am declaring the necessary variables for the server implementation. connection_file_descriptors is a pointer to an array of integers to store connection file descriptors. The semaphore variables (num_of_threads, connection_top, and buffer_mutex) are used for synchronization. buffer_size_G is an integer to keep track of the buffer size.

The server starts by initializing semaphores and creating the specified number of worker threads.

```
PROCEDURE semaphore_initialize(
buffer_mutex, connection_top,
num_of_threads, no_threads)
    CALL sem_init(buffer_mutex, 0, 1)
    CALL sem_init(connection_top, 0, 0)
    CALL sem_init(num_of_threads, 0,
        no_threads)
END PROCEDURE
```

Listing 3. Semaphore Initialization

The semaphore_initialize procedure initializes the semaphores with appropriate initial values. buffer_mutex is initialized to 1 to allow mutual exclusion, connection_top is initialized to 0 to start with no connections, and num_of_threads is initialized to the number of threads provided as input.

Each worker thread waits for a connection to be available, processes the request, and then closes the connection.

```
PROCEDURE worker()
WHILE TRUE DO
    CALL sem_wait(connection_top)
    CALL sem_wait(buffer_mutex)
    DECLARE INTEGER connection_file_descriptor
    = connection_file_descriptors[0]
    FOR INTEGER i = 0 TO buffer_size_G - 1
        SET connection_file_descriptors[i]
        = connection_file_descriptors[i + 1]
    END FOR
    SET buffer_size_G = buffer_size_G - 1
    CALL sem_post(buffer_mutex)
    CALL requestHandle(
        connection_file_descriptor)
    CALL Close(connection_file_descriptor)
END WHILE
RETURN NULL
END PROCEDURE
```

Listing 4. Worker Function

The main function sets up the server, creates threads, and handles incoming connections.

The worker procedure represents the functionality of individual worker threads. It continuously waits for a connection to become available, then processes the request and closes the connection. This function ensures that each thread handles

one connection at a time, maintaining thread safety using semaphores.

```
PROCEDURE main(argc, argv)
IF argc != 4 THEN
    PRINT ERROR
    EXIT
END IF

DECLARE INTEGER port, no_threads, buffers
CALL getargs(port, no_threads,
buffers, argv)

ALLOCATE MEMORY connection_file_descriptors
CALL semaphore_initialize(buffer_mutex,
connection_top, num_of_threads, no_threads)

ALLOCATE MEMORY threads_instance.threads
SET threads_instance.number_of_threads
TO no_threads
SET buffer_size_G TO 0

CALL handle_exiting(empty)
CALL create_threads(no_threads)

LISTEN TO INPUT
END PROCEDURE
```

Listing 5. Main Server Function

The main procedure is the entry point of the server program. It starts by parsing command-line arguments to get the port number, number of threads, and buffer size. Then, it initializes necessary data structures, sets up synchronization mechanisms, and creates worker threads. Finally, it enters a loop to accept incoming connections and manage them with worker threads.

B. Client Implementation (client.c)

The client implementation is responsible for sending requests to the server and printing the responses.

```
PROCEDURE request(args)
    DECLARE STRING host
    SET host TO CAST(STRING, args[0])
    DECLARE INTEGER port
    SET port TO atoi(CAST(STRING, args[1]))
    DECLARE STRING filename
    SET filename TO CAST(STRING, args[2])
    DECLARE INTEGER clientfd

    SET clientfd TO Open_clientfd(host, port)
    CALL clientSend(clientfd, filename)
    CALL clientPrint(clientfd)
    CALL Close(clientfd)

    RETURN NULL
END PROCEDURE
```

Listing 6. Client Request Function

The request procedure represents the functionality of individual client requests. It takes args as input, which contains the host, port, and filename. It then opens a client file descriptor using the provided host and port, sends a request to the server using clientSend, prints the response received from the server using clientPrint, and finally closes the client file descriptor.

```
PROCEDURE main(argc, argv)
DECLARE STRING host, filename, port
DECLARE INTEGER no_threads

IF argc != 4 THEN
    PRINT ERROR
    EXIT
END IF

SET host TO argv[1]
SET port TO argv[2]
SET filename TO argv[3]

SET no_threads TO sysconf(_SC_NPROCESSORS_ONLN
    )
IF no_threads < 1 THEN
    SET no_threads TO 1
    PRINT ERROR
END IF

DECLARE ARRAY OF pthread_t threads[no_threads]
DECLARE ARRAY OF STRING args[3]
SET args[0] TO host
SET args[1] TO port
SET args[2] TO filename

FOR INTEGER i FROM 0 TO no_threads - 1 DO
    PRINT CREATING THREAD i
    CREATE THREAD I with args
END FOR

FOR INTEGER i FROM 0 TO no_threads - 1 DO
    PRINT JOINING THREADS
    JOING THREADS
END FOR

EXIT
END PROCEDURE
```

Listing 7. Client Main Function

The main procedure serves as the entry point for the client program. It starts by parsing command-line arguments to get the host, port, and filename. It then determines the number of threads to use based on the number of processors available using sysconf(_SC_NPROCESSORS_ONLN). If the number of threads is less than 1, it sets it to 1 to avoid errors.

Next, it creates an array of pthread threads and an array of arguments to pass to each thread. It then iterates through each thread, creating and joining them. Each thread is responsible for sending a request to the server.

III. Results

```
abdullahlinux@Abdullah-Laptop:~/HW3$ make
mkdir -p public
cp output.cgi favicon.ico home.html public
abdullahlinux@Abdullah-Laptop:~/HW3$ ./server 5003 8 16
GET /output.cgi?5 HTTP/1.1
GET /output.cgi?5 HTTP/1.1
GET /output.cgi?5 HTTP/1.1
GET /output.cgi?5 HTTP/1.1
GET /output.cgi?5 HTTP/1.1
GET /output.cgi?5 HTTP/1.1
GET /output.cgi?5 HTTP/1.1
GET /output.cgi?5 HTTP/1.1
```

Fig. 1. running server

```
abdullahlinux@Abdullah-Laptop:~/HW3$ ./client localhost 5003 /output.cgi?5
Creating thread 0
Creating thread 1
Creating thread 2
Creating thread 3
Creating thread 4
Creating thread 5
Creating thread 6
Creating thread 7
Joining thread 0
Header: HTTP/1.0 200 OK
Header: Server: blg312e Web Server
Header: HTTP/1.0 200 OK
Header: Server: blg312e Web Server
Header: HTTP/1.0 200 OK
Header: Server: blg312e Web Server
Header: HTTP/1.0 200 OK
Header: Server: blg312e Web Server
Header: HTTP/1.0 200 OK
Header: Server: blg312e Web Server
Header: HTTP/1.0 200 OK
Header: Server: blg312e Web Server
Header: HTTP/1.0 200 OK
Header: Server: blg312e Web Server
Header: HTTP/1.0 200 OK
Header: Server: blg312e Web Server
Header: Content-length: 123
Header: Content-type: text/html
<p>Welcome to the CGI program</p>
<p>My only purpose is to waste time on the server!</p>
<p>I spun for 5.00 seconds</p>
Header: Content-length: 123
Header: Content-type: text/html
<p>Welcome to the CGI program</p>
Header: Content-length: 123
<p>My only purpose is to waste time on the server!</p>
Header: Content-length: 123
<p>I spun for 5.00 seconds</p>
Header: Content-type: text/html
<p>Welcome to the CGI program</p>
<p>My only purpose is to waste time on the server!</p>
<p>I spun for 5.00 seconds</p>
Header: Content-length: 123
Header: Content-type: text/html
Header: Content-length: 123
```

Fig. 2. client requesting multiple times in a multithreaded way

```
Header: Content-type: text/html
<p>Welcome to the CGI program</p>
Header: Content-length: 123
Header: Content-type: text/html
Header: Content-type: text/html
<p>Welcome to the CGI program</p>
<p>Welcome to the CGI program</p>
<p>My only purpose is to waste time on the server!</p>
<p>I spun for 5.00 seconds</p>
Header: Content-length: 123
Header: Content-type: text/html
<p>My only purpose is to waste time on the server!</p>
<p>I spun for 5.00 seconds</p>
Joining thread 1
Joining thread 2
Joining thread 3
<p>Welcome to the CGI program</p>
<p>My only purpose is to waste time on the server!</p>
<p>I spun for 5.00 seconds</p>
Joining thread 4
Joining thread 5
Joining thread 6
<p>Welcome to the CGI program</p>
<p>My only purpose is to waste time on the server!</p>
<p>I spun for 5.00 seconds</p>
<p>My only purpose is to waste time on the server!</p>
<p>I spun for 5.00 seconds</p>
Joining thread 7
abdullahlinux@Abdullah-Laptop:~/HW3$
```

Fig. 3. client terminating after joining threads

```
abdullahlinux@Abdullah-Laptop:~/HW3$ ./server 5003 8 16
GET /output.cgi?5 HTTP/1.1
GET /output.cgi?5 HTTP/1.1
GET /output.cgi?5 HTTP/1.1
GET /output.cgi?5 HTTP/1.1
GET /output.cgi?5 HTTP/1.1
GET /output.cgi?5 HTTP/1.1
GET /output.cgi?5 HTTP/1.1
GET /output.cgi?5 HTTP/1.1
^C
shutting down....
abdullahlinux@Abdullah-Laptop:~/HW3$
```

Fig. 4. server shutting down properly

Answers to Questions

Acknowledgment

I would like to express my gratitude to my mom and dad.

References

[1] My brain and fingers