Escuela de Ingeniería en Computación

Principios de Sistemas Operativos

Proyecto #3

R-find

Integrantes:

Francisco Villanueva Quirós - 2021043887

Jarod Cervantes Gutiérrez - 2019243821

Semestre II

2024

Fecha de Entrega:

25/11/2024

Introducción

Para el tercer proyecto del curso de sistemas operativos, se solicitó hacer una implementación del comando rfind de Unix, mediante el uso de sockets. Para ello, es necesario realizar una aplicación que simule dicho comportamiento, sobre un servidor remoto que se va a crear. Similar al comando rfind, este proyecto se va a ejecutar desde línea de comandos, siguiendo la siguiente estructura:

- rfind . -name SEARCH_NAME -ip 127.0.0.1

Por lo que, es necesario establecer el servidor con sus respectivas funciones de búsqueda de archivos y manejo de hilos. Además, es importante el manejo de múltiples clientes que van a realizar peticiones de búsqueda al servidor.

Descripción del problema (enunciado)

Como se mencionó anteriormente, la idea principal del proyecto consta en la creación de dos programas, el programa del cliente, que deberá poder ejecutarse en cualquier computador y el programa rfind del servidor que se ejecutara en una máquina en particular. Para su funcionamiento, el servidor debe ser el primero en ejecutarse en un directorio mediante un socket. Con esto, el cliente debe ser capaz de ejecutar el programa mediante la línea de comando e indicar cual o cuales archivos buscar en el servidor. Por lo que, el flujo del sistema consta que el cliente le solicita al servidor hacer una búsqueda de archivos y este de responder enviando las coincidencias encontradas

El programa rfind en el servidor recibe una expresión regular que corresponde a los archivos que se van a buscar. Es importante que el comando contenga la expresión regular y el número de ip en el cual se encuentra la red del servidor. El resultado del comando será un listado, mostrado en la máquina cliente, de cada uno de los archivos en el servidor que cumplen con dicha condición de búsqueda.

Uno de los aspectos más importantes del proyecto es el manejo de expresiones regulares. Es necesario utilizar la librería regex.h, librería de C que cuenta con funciones capaces de realizar un análisis de las expresiones regulares. La función regcomp se debe ejecutar inicialmente y recibe la expresión regular ingresada por comando, luego la función regexec se debe ejecutar sobre cada una de los nombres de los archivos en el servidor

Definición de estructuras de datos

Para la creación del programa se realizaron diferentes estructuras de datos. Cada una de estas son necesarias para cumplir con los requerimientos mencionados anteriormente.

- FindPaths:
    - Contiene campos:
        - int numFiles
        - char list[MAX_STRINGS][PATH_SIZE]
        - regex_t reg
- Client_addr:
    - sockaddr_in
- Funciones del server:
    - void searchFiles(const char *dir, FindPaths *findPaths)
    - void send_file(int client_socket, char *filename)
    - char *extract_path_parameter(const char *uri, char *parameterName)
    - void paths(int client_socket, const char *uri)
    - void files(int client_socket, const char *uri)
    - void *handle_client(void *arg)
    - void send_files_list(int client_socket, FindPaths *findPaths)
- Funciones del cliente:
    - void receive_file(int client_socket)
    - void receive_path(int client_socket)
    - void getPaths(const char *dir, const char *regex)
    - void getFiles(const char *dir, const char *regex)
    - int *create_socket_request(const char *method, const char *uri)

    - void receive_header(int client_socket)
- Serv_addr:
    - sockaddr_in

Descripción detallada y explicación de componentes principales

A. Programa Servidor

Para este componente, desarrollo una aplicación en C para crear un servidor que implemente un servicio de búsqueda de archivos y devolución de resultados. Para ello, el servidor está programa para recibir peticiones del cliente HTTP, donde mediante el uso de la librería regex de C, se puede hacer una evaluación de las expresiones regulares que envía el cliente. Una vez recibidas, el servidor devuelve las coincidencias si este encuentra algunas.

Primeramente, se inicia con la creación del socket y se le establece el puerto 8080 para escuchar las peticiones del cliente. Posteriormente este utiliza la función bind para la dirección local y se pone en modo para escuchar las peticiones de los clientes conectados. A los clientes que se conectan se les crea un hilo y se les asocia para realizar los servicios de búsqueda.

Mediante los hilos del cliente se realizan las solicitudes al servidor, el cual utiliza dos funciones principales, “paths” y “files”. En la función de paths, se recibe por parámetro el socket del cliente y una URI, luego mediante la libreria de regex, se logra analizar las expresiones regulares que el cliente busca en el servidor que son las direcciones de los archivos. Estos parámetros contienen la dirección de los archivos en un directorio. Sin embargo, se llama a la función searchFiles, con los parámetros de dir y findPaths, que los datos ya procesados del directorio y la expresión regular. SearchFiles se encarga de recorrer recursivamente el sistema de archivos y utiliza regexec para evaluar las coincidencias. Dicha función se encarga de validar si lo que hay es un subdirectorio y si es así se llama de forma recursiva hasta completar la ruta enviada por el cliente si es que esta existe y devuelve una

lista con los archivos que coinciden con la expresión regular de búsqueda. Una vez se completa el llamado a la función, la función paths envía la lista a la nueva función “send_file_list”, la cual se encarga de armar la respuesta que se va a retornar al cliente mediante un JSON. Esta funcion tambien valida si al agregar los archivos de la lista, se excede el tamaño del buffer, hace que se envíen en otro paquete.

Por otro lado, la función files tiene un comportamiento similar a paths, esta se encarga de igual forma llamar a searchFiles con los parámetros mencionados anteriormente y obtener la lista de coincidencias retorna por searchFiles. Sin embargo, esta función de files, tira sobre esta lista un archivo a la vez y se envían a la función sendFile. Esta recibe de igual forma el socket del cliente y el nombre del archivo que hace match. Luego de igual forma va creando el encabezado de la respuesta que se le va a enviar al cliente en formato JSON. El contenido se obtiene del bucle que lee todo el archivo hasta que ya se haya completado y se envía al cliente mediante: “send(client_socket, body_content, bytes_read, 0);”.

B. Programa Cliente

En este componente, se desarrolla un programa en C, que permite realizar solicitudes HTTP al servidor mencionado anteriormente, para realizar la búsqueda de archivos mediante una expresión regular la cual es enviada. El cliente se conecta con el servidor mediante el uso de sockets y el sistema se inicia desde la línea de comandos donde se deben establecer el directorio y la expresión regular que se va a buscar, el regex.

Lo primero que realiza el programa, es que el comando contenga la estructura correcta, debe ser la siguiente: “rfind . -name SEARCH_NAME”, donde -name es el nombre del directorio y SEARCH_NAME es el regex que se va a indicar al servidor que debe buscar. Por lo que, el cliente se encarga de validar que el comando contenga los componentes adecuados. Posteriormente, se hace una distinción, ya que, si el comando tiene tres parámetros se llama a la función getPaths, mientras si contiene los cuatro parámetros se llama a la función getFiles.

Para el caso de contar con tres parámetros, se llama a la función getPaths. Esta recibe por parámetros el directorio y el regex que se va a utilizar para la búsqueda. La función construye un URI que se utiliza para realizar la solicitud HTTP al servidor. Este URI tiene el formato "/paths?dir=<dir>&regex=<regex>". Una vez se construye la solicitud, se llama a la función create_socket_request, con el método GET y la uri como parámetros. Esta es la función que se encarga de crear la conexión con el servidor mediante sockets como se mencionó anteriormente. Esta va a retornar un identificador que se utiliza para gestionar la conexión con el servidor.

Si este identificar es true se llama a la función receive_paths cuando el identificador como parámetro. Esta función se encarga de recibir la lista de rutas de archivos enviadas por el servidor y de imprimirlas en la consola. Esto para finalizar cerrando el identificador del cliente.

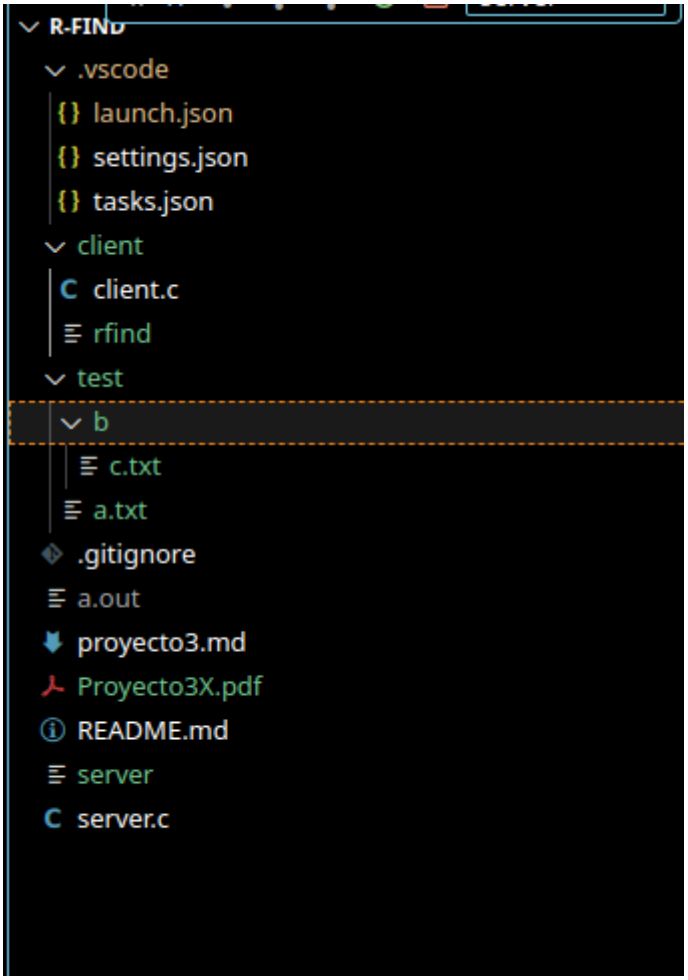
Por otra parte, como se mencionaba anteriormente, si el comando cuenta con cuatro parámetros, se hace el llamado a la función getFiles. Esta tiene un comportamiento muy similar a getPaths, recibe por parámetro, el directorio y la expresión regex. Posteriormente, se hace la creación del URI, de la misma forma que se mencionaba anteriormente. Una vez, la creación del uri es correcta se llama a la función create_socket_request con GET y el uri como parámetros. De igual forma, esta retorna un identificador que funciona para gestionar la conexión con el servidor, donde si esta es exitosa, llama a la función receive_file con el identificar como parámetro. Esta función mediante un chicle, lee los datos del archivo que recibe del servidor. Se utiliza el siguiente código para completar con la lectura de la respuesta HTTP enviada del servidor: “(bytes_read = read(client_socket, response, HTTP_RESPONSE_SIZE)) > 0”. Mientras se esté en el ciclo, se va a imprimir la respuesta recibida del servidor.

Análisis de resultados de pruebas

Búsqueda de archivos

Para la búsqueda de archivos se logró comprobar que el proyecto cumple con las expectativas de extraer los archivos que cumplan el patrón regex. Este listado de archivos se visualiza en la terminal del cliente de dónde él mismo puede ver las opciones que tiene para la recuperación de archivos.

Directorio de pruebas



Comando para búsqueda y resultado

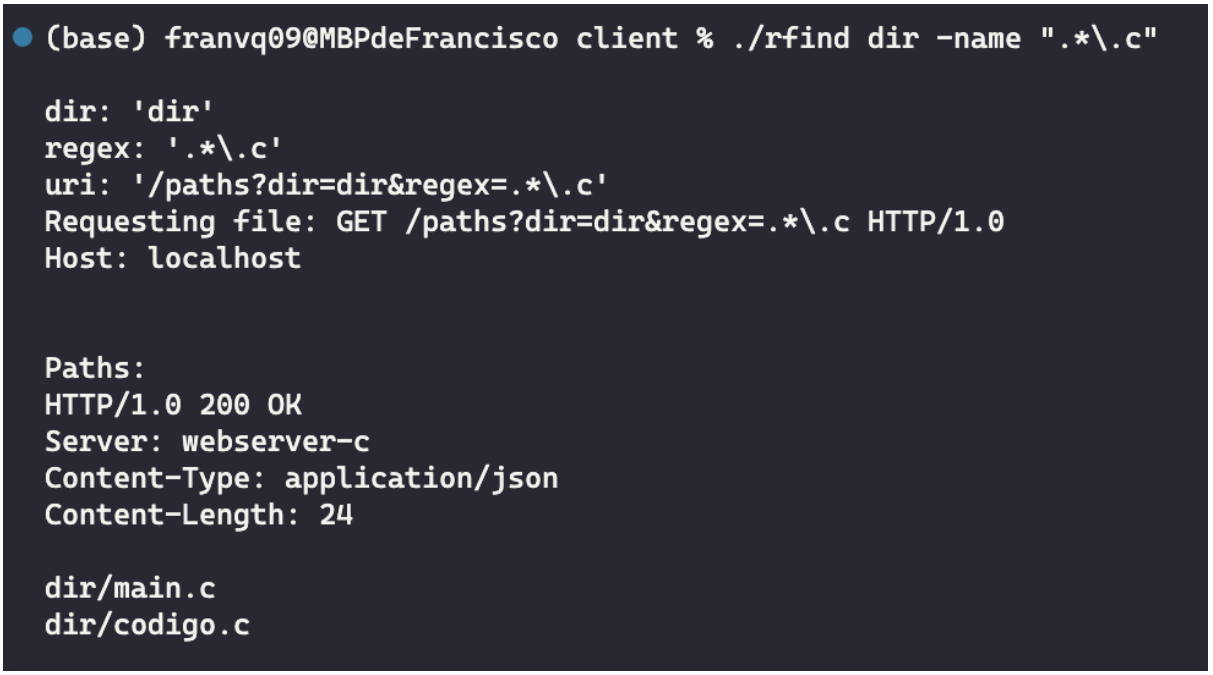
```
rodxja@rodxja-Inspiron-5570:~/Documents/os/r-find/client$ ./rfind . -name .*\.txt
dir: '.'
regex: '.*.txt'
uri: '/paths?dir=.&regex=.*.txt'
Requesting file: GET /paths?dir=.&regex=.*.txt HTTP/1.0
Host: localhost


Paths:
HTTP/1.0 200 OK
Server: webserver-c
Content-Type: application/json
Content-Length: 28

./test/a.txt
./test/b/c.txt
```

Como se muestra en la imagen los únicos dos archivos que cumplen con el patrón son los mostrados en la terminal

Prueba 2

Directorio de pruebas:

Comando de búsqueda y respuesta:

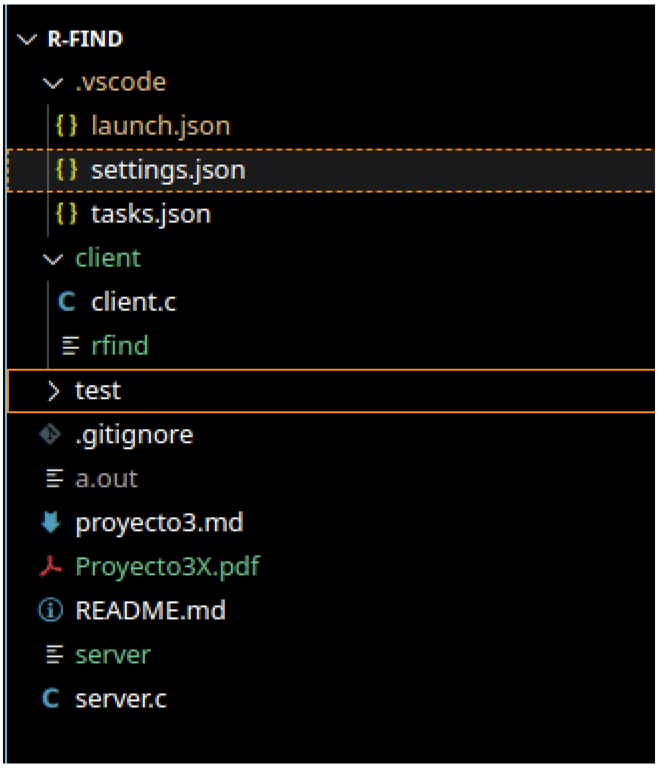


Como se puede ver en la imagen, la respuesta HTTP del servidor fue exitosa. Las archivos que se retornan son los que tienen la extensión .c.

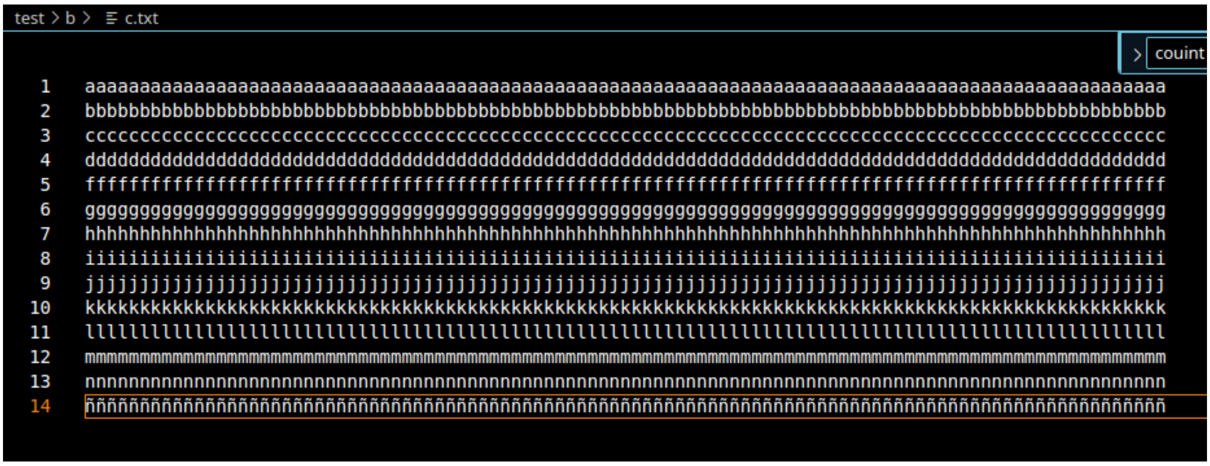
Recuperación de archivos

El servidor logra enviar al cliente y crear los archivos que cumplan con el patrón de búsqueda. Debido a que los archivos devueltos contienen los paths relativos, estos mismos se duplican en el directorio desde donde el cliente ejecuta su comando.
Dentro de la carpeta cliente, que es de donde se ejecuta el comando, se podrá ver cómo se crea un archivo c.txt con la información que tiene

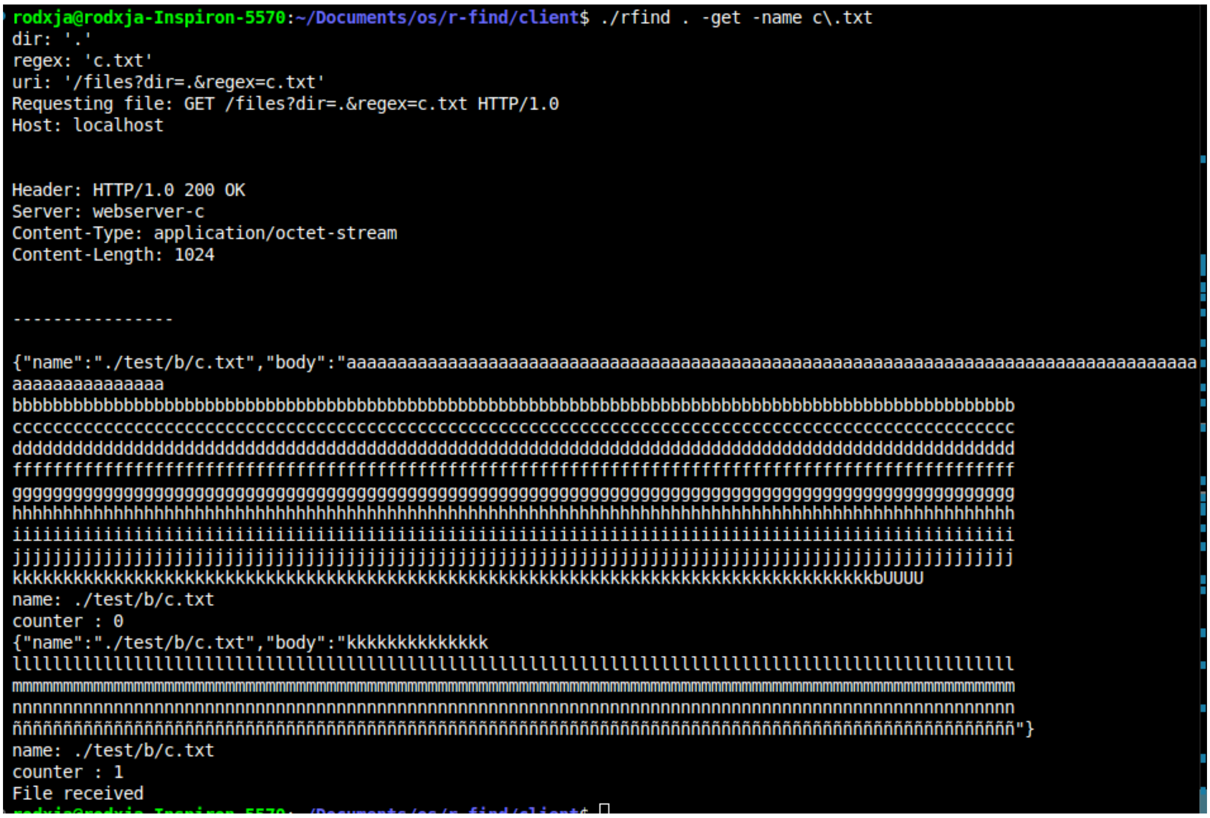
Directorio



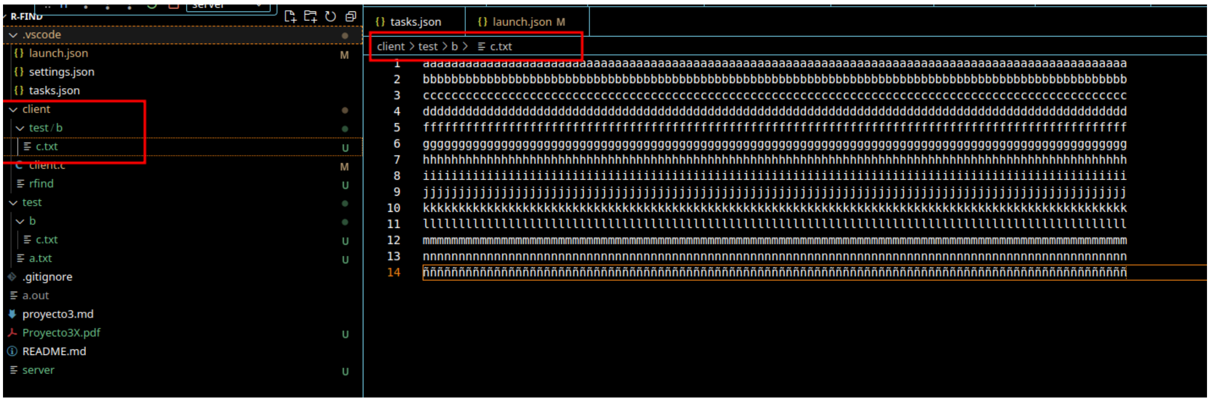
Archivo c.txt



Ejecución del comando



Creación del archivo en el directorio cliente



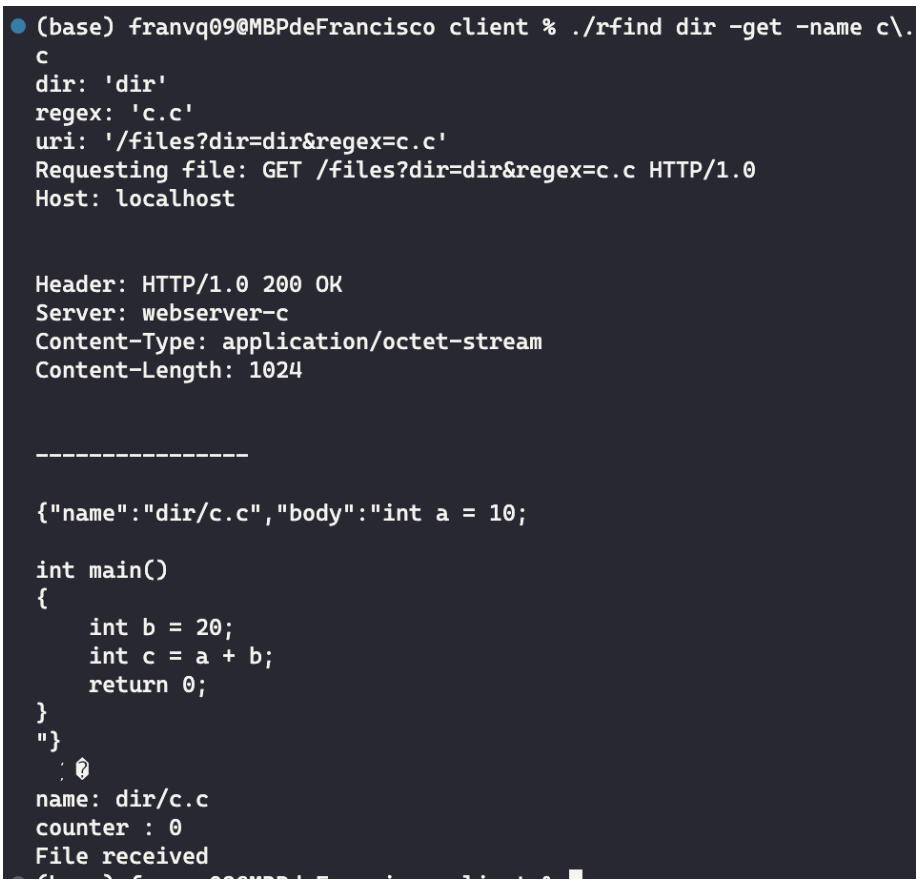
Prueba 2

Directorio de pruebas:

Archivo main.c

```c
int a = 10;

int main()
{
    int b = 20;
    int c = a + b;
    return 0;
}
```

Respuesta del servidor hacia el cliente:



Como se puede observar, el servidor retorna el archivo que está buscando de forma correcta.

Conclusiones

- El uso de sockets es una herramienta vital para la comunicación entre distintos computadores que no compartan la misma ubicación física.
- Se logró implementar la búsqueda de archivos en un servidor, mientras cumplan con el patrón de regex.
- La recuperación de archivos es posible mediante el uso de sockets, este se logra por medio del envío de bloques de tamaño n, y el cliente se encarga de construir el archivo con los datos necesarios.