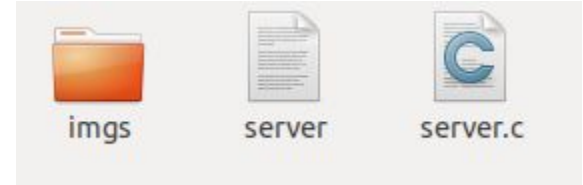

Laboratorio III

— Presentación final —
Zanotti Eduardo - 92071

Contenido del trabajo a presentar

- Archivo de código fuente **server.c**.
- Carpeta **imgs** con las correspondientes imágenes a devolver por el server.
- Binario para ejecutar **server**.



Funcion socket()

```
if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
{
    perror("socket failed");
    exit(EXIT_FAILURE);
}
```

Crea un endpoint para establecer la comunicación, retornando un **file descriptor** que hace referencia a dicho endpoint.

- **AF_INET**: Establece como protocolo de comunicación IPv4.
- **SOCK_STREAM**: Canal de comunicación TCP/IP.

Funcion setsockopt()

```
if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt, sizeof(opt)))
{
    perror("setsockopt");
    exit(EXIT_FAILURE);
}
```

Utilizada para setear opciones de configuración al socket.

- **Server_fd**: Nuestro socket.
- **SOL_SOCKET**: Establece el nivel al que se va a manipular dicho socket.
- **SO_REUSEADDR / SO_REUSEPORT**: Establece que es válido reusar el puerto y la ip asignada al socket.
- **&OPT**: Utilizado para pasar información utilizada por un comando en particular.

Estructura sockaddr_in y funcion bind()

- **Sockaddr_in** es una estructura utilizada para configurar la dirección del socket.

```
struct sockaddr_in address;  
address.sin_family = AF_INET;  
address.sin_addr.s_addr = INADDR_ANY;  
address.sin_port = htons(PORT);  
  
if (bind(server_fd, (struct sockaddr *)&address, sizeof(address))<0)  
{  
    perror("bind failed");  
    exit(EXIT_FAILURE);  
}
```

Cuando un socket es creado mediante la función **socket()** aún no tiene asignada una dirección de conexión, por lo que la función **bind()**, mediante la estructura definida previamente, le asignamos al socket una dirección específica.

Funcion listen()

La función **listen()** marca a nuestro socket en estado pasivo, eso significa que va a estar a la escucha de nuevas conexiones.

```
if (listen(server_fd, 3) < 0)
{
    perror("listen");
    exit(EXIT_FAILURE);
}
```

Creamos un poll de threads

```
fds[0].fd = server_fd;

while(1==1)
{
    int ret = poll(fds, 1, 5000);

    if (ret == -1) {
        fprintf(stderr, "Error al intentar agregar al poll: %s\n", strerror(errno));
        return 1;
    }

    if (fds[0].revents & POLLIN) {
        if ((new_socket = accept(server_fd, (struct sockaddr *)&address,
                                (socklen_t *)&addrlen)) < 0)
        {
            perror("accept");
            exit(EXIT_FAILURE);
        }
        pthread_create(&thread, NULL, SocketThread, (void*) &new_socket);
    }
}
```

Creamos un poll de threads

- La funcion **poll()** espera a que algun file descriptor (socket) reciba un evento para realizar alguna operación de I/O.
- POLLIN significa que hay datos para leer.
- La funcion **accept()** acepta la conexión del socket. **accept** a connection on a socket. Extrae la primer conexión, crea un nuevo socket conectado y retorna un nuevo file descriptor que apunta al nuevo socket.
- La funcion **pthread_create()** comienza un nuevo hilo de ejecución.

Funcion SocketThread()

- Función encargada de elegir al azar una imagen y comenzar el envío de datos al cliente.

```
void *SocketThread(void* socket)
{
    const char *a[MAX_IMGS];
    a[0] = "imgs/car1.jpg";
    a[1] = "imgs/car2.jpg";
    a[2] = "imgs/nice.jpg";
    a[3] = "imgs/supercar.jpg";

    srand(time(0));
    int r = rand() % 4;

    sendFile(*(int*)socket,a[r],(char*)malloc(sizeof(a[r])));
    printf("response enviado\n");
}
```

Funcion sendFile()

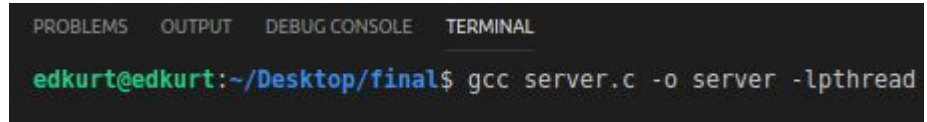
```
int sendFile(int socket, const char* path, long fileSize) {  
  
    char* string;  
    asprintf(&string, HEADERS, getFileSize(path));  
    send(socket, string , strlen(string) , 0 );  
  
    long bytesSent = 0, result = 0, offset = 0;  
    int file = open(path, O_RDONLY);  
  
    if(fileSize == -1 || file == -1) {  
        return -1;  
    }  
  
    do {  
        result = sendfile(socket, file, &offset, fileSize - bytesSent);  
        if(result < 0) {  
            printf("Socket: No se pudo encontrar el file '%s' en el socket %d\n", path, socket);  
            break;  
        }  
        bytesSent += result;  
    } while(bytesSent < fileSize || result < 0);  
  
    free(string);  
    close(file);  
  
    return result;  
}
```

Funcion `sendFile()`

Función encargada de leer del disco la imagen correspondiente, setear los headers del response y enviar mediante stream de bytes la imagen.

Compilar y ejecutar

1. Compilar el código fuente mediante el siguiente comando



A screenshot of a terminal window with a dark background. At the top, there are four tabs: 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL', with 'TERMINAL' being the active tab. The terminal shows a prompt 'edkurt@edkurt:~/Desktop/final\$' followed by the command 'gcc server.c -o server -lpthread'.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
edkurt@edkurt:~/Desktop/final$ gcc server.c -o server -lpthread
```

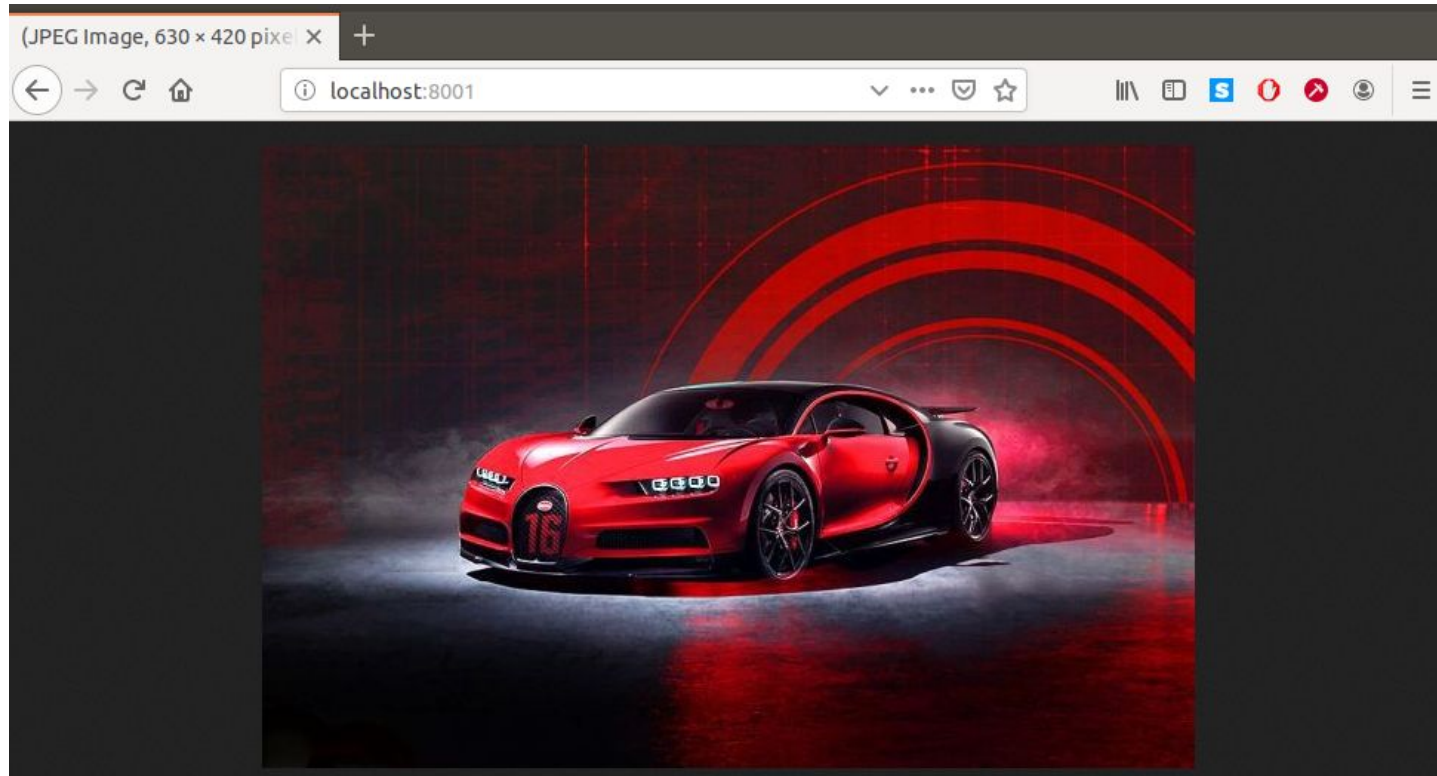
2. Ejecutar el servidor



A screenshot of a terminal window with a dark background. At the top, there are four tabs: 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL', with 'TERMINAL' being the active tab. The terminal shows a prompt 'edkurt@edkurt:~/Desktop/final\$' followed by the command './server'.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
edkurt@edkurt:~/Desktop/final$ ./server
```

Ejemplo



Gracias!