Mienbros:

lara falfan noel de jesus

martinez ramirez fernando

Rosas Hernandez christian Guillermo

Proyecto en Equipo “Simulador de NAT” Sistemas Operativos

2023

Contenido

[Introducción 2](#_Toc153473406)

[Desarrollo 2](#_Toc153473407)

[Proceso A (Código 1) 2](#_Toc153473408)

[Proceso B (Servidor - Código 2) 4](#_Toc153473409)

[Proceso C (Manipulador de Datos - Código 3) 9](#_Toc153473410)

[Conclusiones 12](#_Toc153473411)

# Introducción

Este proyecto busca simular un sistema de comunicación basado en la estructura de un NAT (Network Address Translation), con el propósito de ofrecer un entendimiento práctico de los conceptos fundamentales en Redes de Computadoras. En esta configuración, se presentan tres procesos distintos: A, B y C, los cuales emulan la interacción entre dispositivos y servidores en una red.

# Desarrollo

## Proceso A (Código 1)

El código correspondiente al Proceso A (cliente) inicia estableciendo un socket para la comunicación con el servidor (Proceso B). Utiliza funciones como socket(), connect(), send() y recv() para la creación del socket, conexión al servidor, envío de mensajes y recepción de respuestas, respectivamente. Esta sección simula la interacción de un cliente que envía información al servidor mediante la consola, representando la fase inicial de la transmisión de datos.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

void manejarConexion(int client\_socket) {

char buffer[1024];

while (1) {

// Enviar mensaje al servidor

printf("Ingrese un mensaje para el servidor: ");

fgets(buffer, sizeof(buffer), stdin);

send(client\_socket, buffer, sizeof(buffer), 0);

// Recibir respuesta del servidor

recv(client\_socket, buffer, sizeof(buffer), 0);

printf("Respuesta del servidor: %s\n", buffer);

}

}

int main() {

int client\_socket;

struct sockaddr\_in server\_addr;

// Crear el socket del cliente

if ((client\_socket = socket(AF\_INET, SOCK\_STREAM, 0)) == -1) {

perror("Error al crear el socket");

exit(EXIT\_FAILURE);

}

// Configurar la estructura sockaddr\_in

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(PORT);

server\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

// Conectar al servidor

if (connect(client\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1) {

perror("Error al conectar al servidor");

exit(EXIT\_FAILURE);

}

printf("Conectado al servidor.\n");

// Llamar a la función para manejar la conexión

manejarConexion(client\_socket);

// Cerrar el socket del cliente (si es necesario cerrar la conexión al finalizar)

close(client\_socket);

return 0;

}

## Proceso B (Servidor - Código 2)

El código del Proceso B actúa como un servidor que acepta múltiples conexiones entrantes. Emplea socket(), bind(), listen() y accept() para crear un socket, vincularlo a una dirección y puerto específicos, escuchar conexiones entrantes y aceptarlas, respectivamente. Maneja conexiones entrantes de los clientes (Proceso A) y del Proceso C. Cuando recibe datos del Proceso A, los reenvía al Proceso C utilizando sockets adicionales, espera la respuesta modificada y la devuelve al cliente inicial. Este proceso simula la función de un servidor central que coordina la comunicación entre dispositivos en una red.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <pthread.h>

#define PORT 8080

#define MAX\_CONNECTIONS 5

typedef struct {

int socket;

struct sockaddr\_in client\_addr;

} ConnectionInfo;

void \*manejarConexion(void \*arg)

{

ConnectionInfo \*info = (ConnectionInfo \*)arg;

char buffer[1024];

printf("Conexión aceptada desde %s:%d\n", inet\_ntoa(info->client\_addr.sin\_addr), ntohs(info->client\_addr.sin\_port));

while (1)

{

ssize\_t bytes\_recv = recv(info->socket, buffer, sizeof(buffer), 0);

if (bytes\_recv <= 0)

{

perror("Error al recibir datos o conexión cerrada");

break;

}

// Enviar mensaje al proceso C

printf("Enviando mensaje a C: %s\n", buffer);

// Crear un socket para la comunicación con el proceso C

int socket\_to\_C = socket(AF\_INET, SOCK\_STREAM, 0);

if (socket\_to\_C == -1)

{

perror("Error al crear el socket para C");

close(info->socket);

free(info);

pthread\_exit(NULL);

}

struct sockaddr\_in c\_addr;

c\_addr.sin\_family = AF\_INET;

c\_addr.sin\_port = htons(8081); // Puerto para la comunicación con C

c\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1"); // Dirección IP de localhost

// Conectar con el proceso C

if (connect(socket\_to\_C, (struct sockaddr \*)&c\_addr, sizeof(c\_addr)) == -1)

{

perror("Error al conectar con C");

close(socket\_to\_C);

continue;

}

// Enviar mensaje a C

send(socket\_to\_C, buffer, bytes\_recv, 0);

// Recibir respuesta de C (ya en mayúsculas)

recv(socket\_to\_C, buffer, sizeof(buffer), 0);

printf("Respuesta de C: %s\n", buffer);

// Enviar respuesta de C al cliente A

printf("Enviando respuesta al cliente A... \n");

send(info->socket, buffer, bytes\_recv, 0);

// Cerrar el socket de comunicación con C

close(socket\_to\_C);

}

close(info->socket);

free(info);

pthread\_exit(NULL);

}

int main()

{

int server\_socket, new\_socket;

struct sockaddr\_in server\_addr;

socklen\_t addr\_size = sizeof(struct sockaddr\_in);

pthread\_t thread\_id;

ConnectionInfo \*info;

// Crear el socket del servidor

if ((server\_socket = socket(AF\_INET, SOCK\_STREAM, 0)) == -1)

{

perror("Error al crear el socket");

exit(EXIT\_FAILURE);

}

// Configurar la estructura sockaddr\_in

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(PORT);

server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

// Enlazar el socket a la dirección y puerto

if (bind(server\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1)

{

perror("Error al enlazar el socket");

exit(EXIT\_FAILURE);

}

// Escuchar por conexiones entrantes

if (listen(server\_socket, MAX\_CONNECTIONS) == -1)

{

perror("Error al intentar escuchar");

exit(EXIT\_FAILURE);

}

printf("Esperando conexiones...\n");

while (1)

{

// Aceptar la conexión entrante

info = (ConnectionInfo \*)malloc(sizeof(ConnectionInfo));

if ((new\_socket = accept(server\_socket, (struct sockaddr \*)&(info->client\_addr), &addr\_size)) == -1)

{

perror("Error al aceptar la conexión");

free(info);

exit(EXIT\_FAILURE);

}

info->socket = new\_socket;

// Crear un hilo para manejar la conexión

if (pthread\_create(&thread\_id, NULL, manejarConexion, (void \*)info) != 0)

{

perror("Error al crear el hilo");

free(info);

exit(EXIT\_FAILURE);

}

}

close(server\_socket);

return 0;

}

## Proceso C (Manipulador de Datos - Código 3)

El código del Proceso C recibe los datos enviados por el Proceso B mediante sockets. Posteriormente, lleva a cabo operaciones específicas sobre estos datos, como transformaciones, análisis o cualquier procesamiento requerido. Usa funciones como socket(), bind(), listen(), accept(), recv() y send() para recibir datos, procesarlos y enviar la respuesta modificada al Proceso B. Esta sección representa la manipulación de datos dentro del sistema, modificando la información antes de ser devuelta al servidor para su posterior transmisión al cliente original (Proceso A).

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <ctype.h>

#define PORT 8081 // Puerto para la comunicación con B

int main()

{

int server\_socket, new\_socket;

struct sockaddr\_in server\_addr, client\_addr;

socklen\_t addr\_size = sizeof(struct sockaddr\_in);

char buffer[1024];

// Crear el socket del servidor

if ((server\_socket = socket(AF\_INET, SOCK\_STREAM, 0)) == -1)

{

perror("Error al crear el socket");

exit(EXIT\_FAILURE);

}

// Configurar la estructura sockaddr\_in

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(PORT);

server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

// Enlazar el socket a la dirección y puerto

if (bind(server\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1)

{

perror("Error al enlazar el socket");

exit(EXIT\_FAILURE);

}

// Escuchar por conexiones entrantes

if (listen(server\_socket, 5) == -1)

{

perror("Error al intentar escuchar");

exit(EXIT\_FAILURE);

}

printf("Esperando conexiones...\n");

while (1)

{

// Aceptar la conexión entrante de B

if ((new\_socket = accept(server\_socket, (struct sockaddr \*)&client\_addr, &addr\_size)) == -1)

{

perror("Error al aceptar la conexión de B");

continue; // Continuar esperando conexiones

}

printf("Conexión aceptada desde B: %s:%d\n", inet\_ntoa(client\_addr.sin\_addr), ntohs(client\_addr.sin\_port));

// Ciclo para manejar múltiples solicitudes de B

while (1)

{

// Recibir mensaje de B

ssize\_t bytes\_recv = recv(new\_socket, buffer, sizeof(buffer), 0);

if (bytes\_recv <= 0)

{

if (bytes\_recv == 0)

{

// Conexión cerrada por B de manera ordenada

printf("Conexión cerrada por B\n");

}

else

{

perror("Error al recibir datos o conexión cerrada por B");

}

break; // Salir del ciclo interno si la conexión se cierra

}

printf("Mensaje de B: %s\n", buffer);

// Convertir a mayúsculas

int i = 0;

while (buffer[i] != '\0')

{

buffer[i] = toupper(buffer[i]);

i++;

}

// Enviar respuesta a B

printf("Enviando respuesta a B: %s\n", buffer);

send(new\_socket, buffer, bytes\_recv, 0);

}

// Cerrar socket de conexión con B

close(new\_socket);

}

close(server\_socket);

return 0;

}

# Conclusiones

El modelo de comunicación entre estos procesos representa un ejercicio que refleja los conceptos fundamentales en Redes de Computadoras. La interacción entre el cliente (Proceso A), el servidor (Proceso B) y la capa de manipulación de datos (Proceso C) destaca la relevancia y la interdependencia de cada componente en el procesamiento de información en entornos de red.

Es esencial destacar que la realización de este proyecto implicó una exhaustiva investigación por parte de nuestro equipo. Durante este proceso, encontramos un recurso valioso: un video explicativo en YouTube, el cual sirvió como fuente de inspiración y orientación para abordar este desafío y desarrollar esta simulación de un entorno de comunicación basado en un NAT.

El aprendizaje y la práctica en el uso de sockets durante este proyecto han demostrado ser una experiencia invaluable. Esta habilidad técnica adquirida será de gran utilidad en futuros proyectos, proporcionándonos las bases necesarias para abordar desafíos más avanzados en el ámbito de las redes de computadoras.