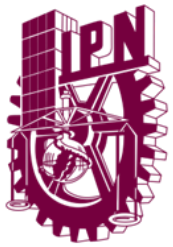


Instituto Politécnico Nacional

IPN

Escuela Superior de Cómputo

ESCOM

Academia de Redes

Aplicaciones para comunicaciones en red

6CV2

Práctica 2

“Sockets Orientados a Conexiones Bloqueantes en C”

Alumna:

Navarrete Becerril Sharon Anette

Fecha de entrega: “Lunes, 14-Octubre-2024”

Profesor: Ojeda Santillan Rodrigo

OBJETIVO

El objetivo de esta práctica es desarrollar y demostrar el uso efectivo de sockets no bloqueantes en aplicaciones de red utilizando el lenguaje de programación C. La práctica se centra en la construcción de servidores capaces de manejar múltiples conexiones de clientes de manera simultánea, sin interrumpir el funcionamiento del servidor ni comprometer la respuesta del sistema. Esta habilidad es esencial en aplicaciones de tiempo real, como servidores de juegos en línea, plataformas de chat y sistemas de transacciones financieras. El estudiante aprenderá a configurar sockets no bloqueantes y gestionará múltiples conexiones de manera concurrente, lo que es clave para el desarrollo de aplicaciones de alto rendimiento.

INTRODUCCIÓN

En aplicaciones de red, especialmente aquellas que requieren alta concurrencia y respuesta rápida, como servidores web o juegos en línea, es crucial evitar que el servidor se detenga mientras espera a que una operación de red, como la lectura o escritura, se complete. Los sockets bloqueantes pueden causar que el hilo principal de ejecución quede "congelado" mientras espera, afectando la latencia y la eficiencia del sistema.

Para evitar este problema, los sockets no bloqueantes permiten que el programa continúe su ejecución mientras las operaciones de red están en proceso. En lugar de detenerse, el socket retorna inmediatamente con un error que indica que la operación debe reintentarse más tarde. Esto permite que un servidor gestione múltiples clientes de manera simultánea y sin interrupciones.

En esta práctica se utiliza el modelo cliente-servidor para demostrar cómo los sockets no bloqueantes pueden mejorar significativamente la capacidad de un servidor para manejar múltiples clientes de manera eficiente. Se utilizan funciones como select() para monitorear múltiples conexiones simultáneamente y determinar si están listas para ser leídas o escritas.

DESARROLLO

La implementación de la práctica consistió en el desarrollo de dos programas en lenguaje C: uno para el cliente y otro para el servidor. A continuación se describe la funcionalidad de cada uno.

Servidor

El servidor se configura como un socket no bloqueante y utiliza la función `select()` para monitorear múltiples conexiones de clientes simultáneamente. Esto permite al servidor seguir aceptando nuevas conexiones mientras gestiona las solicitudes de clientes conectados sin detener su ejecución.

El socket del servidor se crea con la función `socket()`, se marca como no bloqueante utilizando `fcntl()`, y luego se une a un puerto con `bind()`. Finalmente, el servidor comienza a escuchar conexiones entrantes utilizando `listen()`.

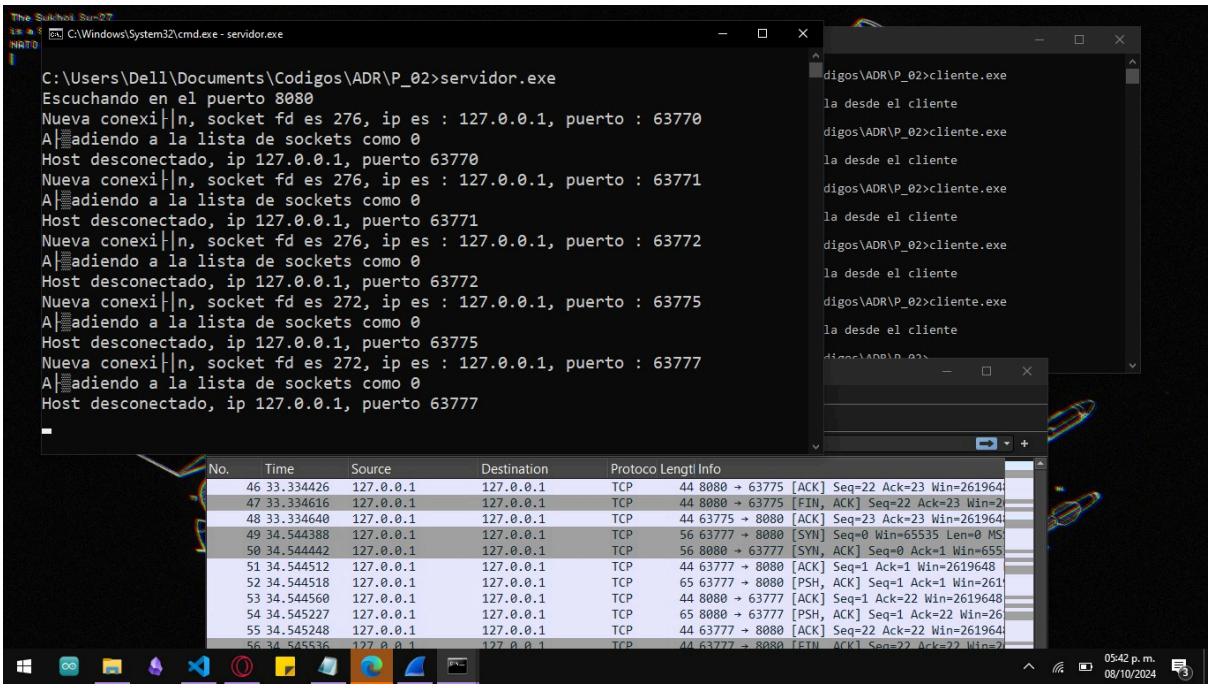
El bucle principal del servidor utiliza `select()` para verificar si hay actividad en el socket del servidor o en los sockets de los clientes conectados. Si se detecta una nueva conexión, el servidor la acepta, y si se reciben datos de un cliente, los procesa y responde.

Código del servidor:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <fcntl.h>
#include <sys/select.h>

#define PORT 8080
#define BUFFER_SIZE 1024

int main() {
    int server_fd, new_socket, max_sd, sd, activity, valread;
    int client_socket[30] = {0};
    int max_clients = 30;
    struct sockaddr_in address;
    char buffer[BUFFER_SIZE];
    fd_set readfds;

    // Crear socket del servidor
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }

    // Configurar el socket del servidor como no bloqueante
    fcntl(server_fd, F_SETFL, O_NONBLOCK);

    // Configurar el tipo de socket
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    // Adjuntar el socket al puerto 8080
    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
        perror("bind failed");
        close(server_fd);
        exit(EXIT_FAILURE);
    }
```

```
    // Escuchar en el socket
    if (listen(server_fd, 3) < 0) {
        perror("listen failed");
        close(server_fd);
        exit(EXIT_FAILURE);
    }

    printf("Escuchando en el puerto %d \n", PORT);

    // Bucle principal para aceptar y manejar conexiones
    while (1) {
        // Limpiar el conjunto de descriptores de socket
        FD_ZERO(&readfds);

        // AÃ±adir el socket del servidor al conjunto de descriptores
        FD_SET(server_fd, &readfds);
        max_sd = server_fd;

        // AÃ±adir los sockets de cliente al conjunto de descriptores
        for (int i = 0; i < max_clients; i++) {
            sd = client_socket[i];
            if (sd > 0) {
                FD_SET(sd, &readfds);
            }
            if (sd > max_sd) {
                max_sd = sd;
            }
        }

        // Esperar a que ocurra alguna actividad en uno de los sockets
        activity = select(max_sd + 1, &readfds, NULL, NULL, NULL);
        if (activity < 0) {
            perror("select error");
        }

        // Si hay una actividad en el socket del servidor, es una nueva conexiÃ³n
        if (FD_ISSET(server_fd, &readfds)) {
            int addrlen = sizeof(address);
            if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t*)&addrlen)) <
0) {
                perror("accept failed");
                exit(EXIT_FAILURE);
            }

            printf("Nueva conexiÃ³n, socket fd es %d, ip es : %s, puerto : %d\n",
                  new_socket, inet_ntoa(address.sin_addr), ntohs(address.sin_port));

            // AÃ±adir el nuevo socket al array de sockets de cliente
            for (int i = 0; i < max_clients; i++) {
                if (client_socket[i] == 0) {
                    client_socket[i] = new_socket;
                    printf("AÃ±adiendo a la lista de sockets como %d\n", i);
                    break;
```

```
            }
        }
    }

    // Manejar IO en otros sockets
    for (int i = 0; i < max_clients; i++) {
        sd = client_socket[i];
        int addrlen = sizeof(address);
        if (FD_ISSET(sd, &readfds)) {
            // Revisar si fue por cierre y leer el mensaje
            if ((valread = read(sd, buffer, BUFFER_SIZE)) == 0) {
                // Alguien se desconectÃ³, obtener detalles e imprimir
                getpeername(sd, (struct sockaddr*)&address, (socklen_t*)&addrlen);
                printf("Host desconectado, ip %s, puerto %d\n",
                      inet_ntoa(address.sin_addr), ntohs(address.sin_port));
                // Cerrar el socket y marcarlo como 0 en la lista
                close(sd);
                client_socket[i] = 0;
            } else {
                // Poner terminador de cadena en el buffer y enviar mensaje de vuelta al cliente
                buffer[valread] = '\0';
                send(sd, buffer, strlen(buffer), 0);
            }
        }
    }
  }
  return 0;
}
```



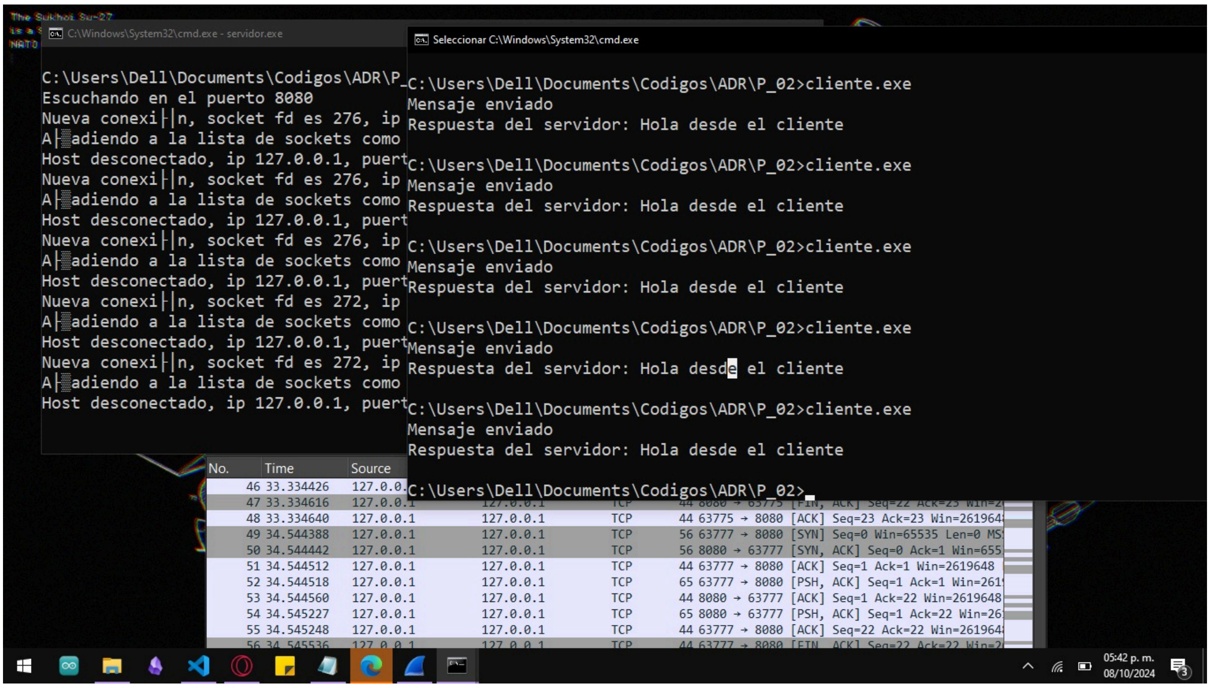
Cliente

El cliente se conecta al servidor no bloqueante y envía un mensaje. Una vez que el servidor responde, el cliente lee la respuesta y luego cierra la conexión.

El socket del cliente se crea con `socket()`, y luego se conecta al servidor utilizando `connect()`. El cliente envía un mensaje con `send()` y recibe la respuesta con `read()`.

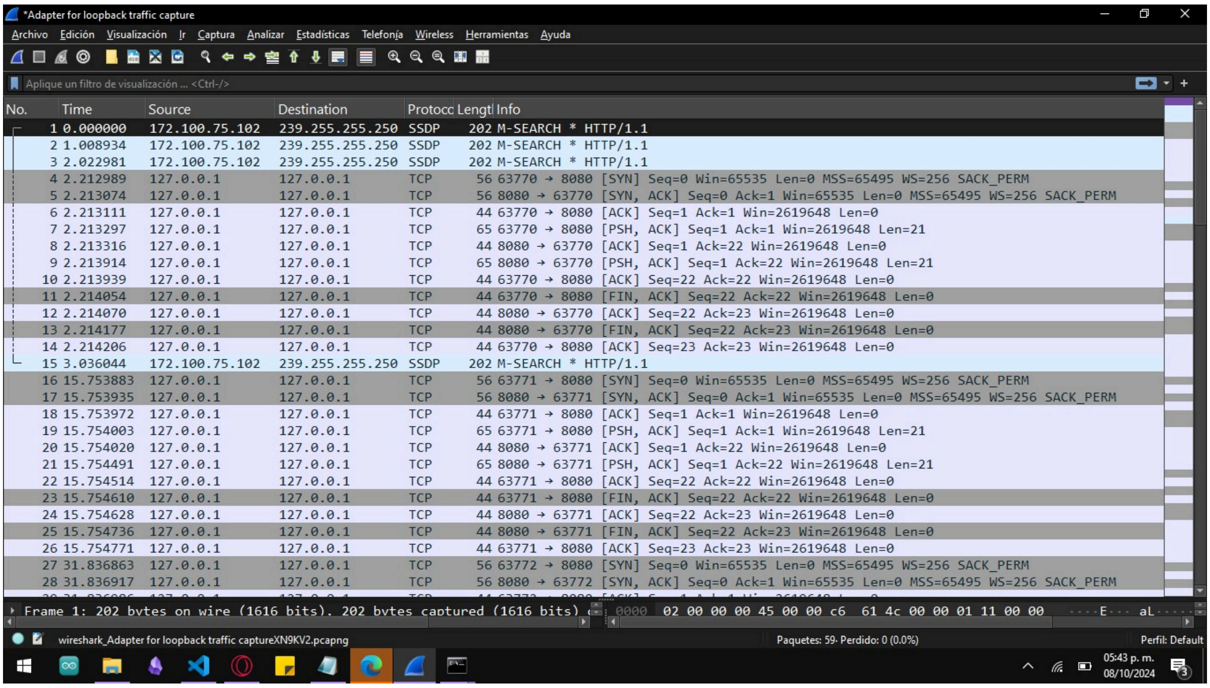
Código del cliente:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <sys/socket.h>
#define PORT 8080
#define BUFFER_SIZE 1024
int main() {
int sock = 0, valread;
struct sockaddr_in serv_addr;
char *hello = "Hola desde el cliente";
char buffer[BUFFER_SIZE] = {0};
// Crear socket
if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
perror("Socket creation error");
return -1;
}
serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(PORT);
// Convertir direcciones IPv4 e IPv6 de texto a binario
if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
perror("Invalid address/ Address not supported");
return -1;
}
// Conectarse al servidor
if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
perror("Connection Failed");
return -1;
}
// Enviar mensaje al servidor
send(sock, hello, strlen(hello), 0);
printf("Mensaje enviado\n");
// Leer respuesta del servidor
valread = read(sock, buffer, BUFFER_SIZE);
printf("Respuesta del servidor: %s\n", buffer);
// Cerrar el socket
close(sock);
}
```



Análisis de tráfico con Wireshark

En el análisis se pudieron identificar claramente las IPs del cliente y el servidor durante la transmisión de datos.Se confirmó que la conexión seguía el protocolo TCP, garantizando una transmisión fiable y secuencial de los datos. Se observó la secuencia de paquetes generados cuando el cliente enviaba un mensaje y el servidor respondía. Esta visualización permitió verificar que el proceso de envío y recepción se realizaba correctamente.



CONCLUSIONES

Navarrete Becerril Sharon Anette:

La implementación práctica de los conceptos de sockets no bloqueantes permitió entender de manera clara su importancia en el desarrollo de aplicaciones de red modernas. Además de ofrecer una solución eficiente para el manejo de múltiples usuarios, estos sockets contribuyen directamente a mejorar la estabilidad y la capacidad de escalamiento en sistemas que requieren interacción en tiempo real.

Bibliografía

Stevens, W. Richard. *Programación en red con Unix: La API de Sockets.* Addison-Wesley Professional, 2003.

Forouzan, Behrouz A. *Comunicaciones de Datos y Redes.* McGraw-Hill, 2012.

Tanenbaum, Andrew S., y David J. Wetherall. *Redes de Computadoras.* Pearson, 2010.

Comer, Douglas E. *Internetworking con TCP/IP Volumen Uno.* Prentice Hall, 2006.

Kurose, James F., y Keith W. Ross. *Redes de Computadoras: Un Enfoque Descendente.* Pearson, 2017.

Beazley, David, y Brian K. Jones. *Python Cookbook: Recetas para Dominar Python 3.* O'Reilly Media, 2013.

Begg, A. *Sockets TCP/IP en C: Guía Práctica para Programadores.* Morgan Kaufmann, 2000.