Instituto Politécnico Nacional

Escuela Superior de Cómputo

“Tienda en línea”

Asignatura:

Aplicaciones para comunicaciones en red

Alumnos:

León Flores Daniela Alejandra

Ocaña Castro Héctor Rigel

Profesor:

Moreno Cervantes Axel

Grupo: 6CM2

Introducción

Cuando dos computadoras se comunican dentro de una red, necesitan formas seguras y confiables para intercambiar información entre los programas que corren en cada una. Una de las formas más comunes de hacerlo es usando el modelo Cliente–Servidor: básicamente, un programa (el servidor) ofrece servicios o recursos, y otro (el cliente) los pide y los usa.

Para que esta comunicación funcione, se usan sockets, que son como canales por donde viajan los datos de ida y vuelta. Estos sockets pueden trabajar con diferentes protocolos, pero el más popular es TCP (Transmission Control Protocol), porque se asegura de que los datos lleguen completos, en el orden correcto y sin errores.

La práctica consiste en crear una aplicación de Tienda en Línea que se comunica usando sockets TCP. En este sistema, el cliente puede hacer compras virtuales, mientras que el servidor se encarga de manejar el inventario, simular el carrito de compras y calcular el total.

Para que todo esto funcione, se desarrollaron tres partes clave:

1. Servidor: administra los productos disponibles, lleva el control de los carritos de cada cliente y responde a búsquedas, listados, agregados y compras.
2. Cliente: es una interfaz en consola que permite al usuario buscar productos, añadirlos al carrito, revisar lo que lleva y cerrar la compra.
3. Modelo de datos (modelos.h): define cómo se estructuran los mensajes que se mandan entre cliente y servidor, usando una estructura llamada MensajeTienda que se puede convertir en datos binarios.

Este sistema sirve como ejemplo práctico de cómo funciona la comunicación entre cliente y servidor, cómo se envían datos binarios de forma ordenada, cómo se manejan varias solicitudes al mismo tiempo y cómo se sincronizan las operaciones en red, todo dentro de una versión sencilla de una tienda en línea.

Desarrollo

El sistema empieza definiendo un modelo de datos que comparten el cliente y el servidor. Este modelo está en el archivo modelos.h y contiene la estructura MensajeTienda, que es clave para que ambos lados puedan intercambiar información sin enredos. Esta estructura junta todos los datos necesarios en un solo mensaje binario: qué operación se quiere hacer, el ID y la cantidad de un producto, los textos para buscar, y los detalles del artículo como nombre, marca, tipo, precio y cuántos hay disponibles.

Al tener esta base común, tanto el cliente como el servidor pueden entender los mensajes de la misma forma, sin importar quién los haya mandado. En ese mismo archivo también están las enumeraciones que definen los comandos disponibles (como buscar, listar, agregar al carrito, ver el carrito o cerrar la compra) y las respuestas que puede dar el servidor (como enviar un producto, terminar la lista, confirmar pedido, mostrar errores, mandar el ticket o mostrar el contenido del carrito). Cada mensaje lleva todo lo necesario para saber sobre qué se está pidiendo.

```h
#ifndef MODELOS_H
#define MODELOS_H
#include <stdint.h>

#define TXT_MAX 64

enum TipoMsg {
    CMD_BUSCAR = 1,
    CMD_LISTAR = 2,
    CMD_ADD    = 3,
    CMD_CART   = 4,
    CMD_CHECK  = 5, // checkout
    sendItem   = 100,
    sendEnd    = 101, // fin de lista
    sendOK     = 102,
    sendError  = 103, // error con texto
    sendTicket = 104, // folio + total
    sendCartItem = 105
};

typedef struct {
    uint32_t id;
    char     nombre[40];
    char     marca[24];
    char     tipo[16];
    uint32_t precio_cent;
    uint16_t stock;
} Articulo;

typedef struct {
    int32_t  tipo;
    int32_t  id_articulo;         // para ADD/CHECK
    int32_t  cantidad;
    char     datos[TXT_MAX];      // texto de búsqueda o tipo
    Articulo articulo
    uint32_t folio;
    uint32_t total_articulo;      // total por artículo en carrito
    uint32_t total_cent;

#endif
```

Modelos.h

Servidor

El servidor es el componente central del sistema, ya que se encarga de almacenar los productos disponibles y de procesar todas las solicitudes que llegan desde los clientes. Al principio del código se definen algunas estructuras auxiliares y funciones clave para garantizar una comunicación confiable entre ambos extremos. Entre ellas están sendAll() y recvAll(), que aseguran que los datos enviados o recibidos por el socket lleguen completos, incluso si la red no los transmite de una sola vez.

Luego, se inicializa un pequeño catálogo de productos (como una laptop, un mouse y un libro) usando la función inicializar_datos(). Cada artículo se representa con la estructura Articulo, que incluye su ID, nombre, marca, tipo, precio en centavos y cantidad en inventario.

Para responder a las acciones del cliente, el servidor usa funciones como sendItemMsg, sendOKMsg y sendErrorMsg, que construyen mensajes tipo MensajeTienda y los mandan por el socket. El flujo principal está en mensajeAtender(), que decide qué hacer según el comando recibido.

```
#define WIN32_LEAN_AND_MEAN
#define _WIN32_WINNT 0x0600

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <winsock2.h>
#include <ws2tcpip.h>
#include "modelos.h"

#pragma comment(lib, "ws2_32.lib")

#define MAX_ARTICULOS 100

static Articulo articulos[MAX_ARTICULOS];
static int cant_articulos = 0;

static void error(const char *msg){ fprintf(stderr, "Error: %s (WSA:%d)\n", msg,
WSAGetLastError()); exit(1);}

static int sendAll(SOCKET s, const char* buf, int len){
    int sent = 0;
    while (sent < len){
        int n = send(s, buf + sent, len - sent, 0);
        if(n <= 0) return -1;
        sent += n;
    }
    return 0;
}
```

```c
static int recvAll(SOCKET s, char* buf, int len){
    int recvd = 0;
    while (recvd < len){
        int n = recv(s, buf + recvd, len - recvd, 0);
        if(n <= 0) return -1;
        recvd += n;
    }
    return 0;
}

/* ---------------- Datos de ejemplo ---------------- */
static void inicializar_datos(void){
    Articulo a1 = (Articulo){1, "Laptop Gamer", "Dell", "Electronica", 1500000,
5};   // $15000.00
    Articulo a2 = (Articulo){2, "Mouse Inalambrico", "Logitech", "Electronica",
30000, 10};
    Articulo a3 = (Articulo){3, "El Principito", "Penguin", "Libros", 25000, 8};
    articulos[0] = a1; articulos[1] = a2; articulos[2] = a3;
    cant_articulos = 3;
}

/* ---------------- Respuestas (renombradas) ---------------- */
static void sendItemMsg(SOCKET s, const Articulo* a){
    MensajeTienda r = (MensajeTienda){0};
    r.tipo = sendItem;
    r.articulo = *a;
    sendAll(s, (const char*)&r, sizeof r);
}

static void sendEndMsg(SOCKET s){
    MensajeTienda r = (MensajeTienda){0};
    r.tipo = sendEnd;
    sendAll(s, (const char*)&r, sizeof r);
}

static void sendOKMsg(SOCKET s, const char* texto){
    MensajeTienda r = (MensajeTienda){0};
    r.tipo = sendOK;
    strncpy(r.datos, texto, sizeof r.datos - 1);
    sendAll(s, (const char*)&r, sizeof r);
}

static void sendErrorMsg(SOCKET s, const char* texto){
    MensajeTienda r = (MensajeTienda){0};
    r.tipo = sendError;
```

```
    strncpy(r.datos, texto, sizeof r.datos - 1);
    sendAll(s, (const char*)&r, sizeof r);
}

static void sendTicketMsg(SOCKET s, uint32_t folio, uint32_t total_cent){
    MensajeTienda r = (MensajeTienda){0};
    r.tipo = sendTicket;
    r.folio = folio; r.total_cent = total_cent;
    sendAll(s, (const char*)&r, sizeof r);
}

/* ---------------- Lógica ---------------- */
static void buscar_articulos(SOCKET cli, const char *texto){
    printf("Buscando: %s\n", texto);
    for(int i=0; i < cant_articulos; i++){
        if(strstr(articulos[i].nombre, texto) || strstr(articulos[i].marca,
texto)){
            sendItemMsg(cli, &articulos[i]);
        }
    }
    sendEndMsg(cli);
}

static void listarTipo(SOCKET cli, const char *tipo){
    printf("Listando tipo: %s\n", tipo);
    for(int i=0; i < cant_articulos; i++){
        if(strcmp(articulos[i].tipo, tipo) == 0){
            sendItemMsg(cli, &articulos[i]);
        }
    }
    sendEndMsg(cli);
}

// Carrito simple en servidor por conexión
typedef struct{uint32_t id; uint16_t cantidad;} Item;
typedef struct{Item it[64]; int n;} Carrito;

static Articulo* buscarArt(uint32_t id){
    for(int i=0;i<cant_articulos;i++) if(articulos[i].id == id) return
&articulos[i];
    return NULL;
}

static int carritoEdit(Carrito* c, uint32_t id, uint16_t cantidad){
    Articulo* a = buscarArt(id);
```

```
    if(!a) return 1;
    if(cantidad > a->stock) return 2;

    for(int i=0;i<c->n;i++){
        if(c->it[i].id==id){
            if(cantidad==0){
                for(int j=i;j<c->n-1;j++) c->it[j]=c->it[j+1];
                c->n--;
            } else c->it[i].cantidad = cantidad;
            return 0;
        }
    }
    if(cantidad==0) return 0;
    c->it[c->n++] = (Item){id, cantidad};
    return 0;
}

static int checkout(Carrito* c, uint32_t* total_cent){
    uint32_t total = 0;
    // Revalidar stock
    for(int i=0;i<c->n;i++){
        Articulo* a = buscarArt(c->it[i].id);
        if(!a) return 1;
        if(c->it[i].cantidad > a->stock) return 2;
    }
    // Cobrar y descontar
    for(int i=0;i<c->n;i++){
        Articulo* a = buscarArt(c->it[i].id);
        total += a->precio_cent * c->it[i].cantidad;
        a->stock -= c->it[i].cantidad;
    }
    c->n = 0;
    *total_cent = total;
    return 0;
}

static void mostrarCarrito(SOCKET cli, Carrito* c) {
    uint32_t total_general = 0;

    printf("Mostrando carrito (%d items)\n", c->n);

    for(int i = 0; i < c->n; i++) {
        Articulo* a = buscarArt(c->it[i].id);
        if(a) {
            uint32_t total_articulo = a->precio_cent * c->it[i].cantidad;
```

```
            total_general += total_articulo;

            // Enviar item del carrito
            MensajeTienda r = (MensajeTienda){0};
            r.tipo = sendCartItem;
            r.articulo = *a;
            r.cantidad = c->it[i].cantidad;
            r.total_articulo = total_articulo; // Total para este artículo
            sendAll(cli, (const char*)&r, sizeof r);

            printf("  ID:%d %s x%d = $%.2f\n",
                   a->id, a->nombre, c->it[i].cantidad, total_articulo / 100.0);
        }
    }

    // Enviar total general
    MensajeTienda r = (MensajeTienda){0};
    r.tipo = sendTicket; // Reutilizamos este tipo para el total
    r.total_cent = total_general;
    strncpy(r.datos, "Total carrito", sizeof r.datos - 1);
    sendAll(cli, (const char*)&r, sizeof r);

    sendEndMsg(cli);
}

/* ---------------- Dispatcher ---------------- */
static void mensajeAtender(SOCKET cli, MensajeTienda *msg, Carrito* cart){
    switch(msg->tipo){
        case CMD_BUSCAR:
            buscar_articulos(cli, msg->datos);
            break;
        case CMD_LISTAR:
            listarTipo(cli, msg->datos);
            break;
        case CMD_ADD: {
            printf("ADD/EDIT ID:%d Q:%d\n", msg->id_articulo, msg->cantidad);
            int rc = carritoEdit(cart, (uint32_t)msg->id_articulo,
(uint16_t)msg->cantidad);
            if(rc==0) sendOKMsg(cli,"Carrito actualizado");
            else if(rc==1) sendErrorMsg(cli,"Producto no existe");
            else sendErrorMsg(cli,"Stock insuficiente");
        } break;
        case CMD_CART: {
            printf("Solicitud de ver carrito\n");
            mostrarCarrito(cli, cart);
```

```
        } break;
        case CMD_CHECK: {
            uint32_t total=0; int rc = checkout(cart, &total);
            if(rc==0){ sendOKMsg(cli,"Compra exitosa"); sendTicketMsg(cli,
12345, total);}
            else if(rc==1) sendErrorMsg(cli,"Producto no existe");
            else sendErrorMsg(cli,"Stock insuficiente");
        } break;
        default:
            sendErrorMsg(cli, "Tipo de mensaje desconocido");
    }
}

/* ---------------- Main ---------------- */
int main(int argc, char *argv[]){
    if(argc < 2) error("Falta puerto");

    WSADATA wsa; if(WSAStartup(MAKEWORD(2,2), &wsa) != 0) error("WSAStartup");

    inicializar_datos();

    SOCKET s = socket(AF_INET, SOCK_STREAM, 0);
    if(s == INVALID_SOCKET) error("socket");

    BOOL yes=1; setsockopt(s, SOL_SOCKET, SO_REUSEADDR, (const char*)&yes,
sizeof(yes));

    struct sockaddr_in srv; memset(&srv,0,sizeof srv);
    srv.sin_family = AF_INET;
    srv.sin_addr.s_addr = INADDR_ANY;
    srv.sin_port = htons((u_short)atoi(argv[1]));

    if(bind(s, (struct sockaddr *)&srv, sizeof(srv)) == SOCKET_ERROR)
error("bind");
    if(listen(s, 5) == SOCKET_ERROR) error("listen");

    printf("Servidor Tienda en %s:%s\n", "0.0.0.0", argv[1]);

    for(;;){
        struct sockaddr_in cli; int clen = sizeof(cli);
        SOCKET c = accept(s, (struct sockaddr *)&cli, &clen);
        if(c == INVALID_SOCKET){ fprintf(stderr,"accept falló\n"); continue;}

        printf("Cliente conectado\n");
        Carrito cart = (Carrito){0};
```

```
        MensajeTienda msg;
        while (1){
            if(recvAll(c, (char*)&msg, sizeof msg) <0) break;
            mensajeAtender(c, &msg, &cart);
        }
        closesocket(c);
        printf("Cliente desconectado\n");
    }

    closesocket(s);
    WSACleanup();
    return 0;
}
```

Servidor.c

Cliente

El cliente es el componente que permite al usuario interactuar con la tienda desde la consola. Al iniciar el programa, se configura la biblioteca Winsock con WSAStartup(), se crea un socket TCP y se conecta al servidor usando connect(). Una vez establecida la conexión, se muestra un menú con opciones como buscar productos, listar por tipo, agregar al carrito, ver el carrito o finalizar la compra.

Cada opción activa una función que construye un mensaje estructurado, lo envía al servidor y espera la respuesta. Por ejemplo, buscar() envía un mensaje CMD_BUSCAR con el texto ingresado por el usuario y recibe varios mensajes sendItem con los productos encontrados, que se muestran en consola. De forma similar, listar() filtra por categoría, y agregar() permite añadir o eliminar productos del carrito según la cantidad enviada.

La función verCarrito() muestra los artículos agregados y permite modificarlos antes de cerrar la compra. El servidor responde con mensajes sendCartItem y sendTicket, que el cliente interpreta para mostrar subtotales y el total general. Finalmente, finalizar() envía el comando CMD_CHECK para procesar el pago y mostrar el comprobante.

```
#define WIN32_LEAN_AND_MEAN
#define _WIN32_WINNT 0x0600

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <winsock2.h>
#include <ws2tcpip.h>
#include "modelos.h"

#pragma comment(lib, "ws2_32.lib")
```

```c
static void error(const char *msg){
    fprintf(stderr, "Error: %s (WSA:%d)\n", msg, WSAGetLastError());
    exit(1);
}

static int sendAll(SOCKET s, const char* buf, int len){
    int sent = 0;
    while (sent < len){
        int n = send(s, buf + sent, len - sent, 0);
        if(n <= 0) return -1;
        sent += n;
    }
    return 0;
}

static int recvAll(SOCKET s, char* buf, int len){
    int recvd = 0;
    while (recvd < len){
        int n = recv(s, buf + recvd, len - recvd, 0);
        if(n <= 0) return -1;
        recvd += n;
    }
    return 0;
}

void buscar(SOCKET s, const char *texto){
    MensajeTienda msg = (MensajeTienda){0};
    msg.tipo = CMD_BUSCAR;
    strncpy(msg.datos, texto, sizeof(msg.datos) - 1);
    sendAll(s, (const char*)&msg, sizeof msg);

    MensajeTienda resp;
    while (1){
        if(recvAll(s, (char*)&resp, sizeof resp) <0) break;
        if(resp.tipo == sendEnd) break;
        if(resp.tipo == sendItem)
            printf("Producto: %s (%s) $%.2f Stock:%d\n",
                   resp.articulo.nombre, resp.articulo.marca,
                   resp.articulo.precio_cent / 100.0, resp.articulo.stock);
    }
}

void listar(SOCKET s, const char *tipo){
    MensajeTienda msg = (MensajeTienda){0};
    msg.tipo = CMD_LISTAR;
```

```
    strncpy(msg.datos, tipo, sizeof(msg.datos) - 1);
    sendAll(s, (const char*)&msg, sizeof msg);

    MensajeTienda resp;
    while (1){
        if(recvAll(s, (char*)&resp, sizeof resp) <0) break;
        if(resp.tipo == sendEnd) break;
        if(resp.tipo == sendItem)
            printf("Producto: %s (%s) $%.2f Stock:%d\n",
                   resp.articulo.nombre, resp.articulo.marca,
                   resp.articulo.precio_cent / 100.0, resp.articulo.stock);
    }
}

void agregar(SOCKET s, int id, int cantidad) {
    MensajeTienda msg = (MensajeTienda){0};
    msg.tipo = CMD_ADD;
    msg.id_articulo = id;
    msg.cantidad = cantidad;
    sendAll(s, (const char*)&msg, sizeof msg);

    MensajeTienda resp;
    if(recvAll(s, (char*)&resp, sizeof resp) <0) return;

    // Mensajes más descriptivos
    if (cantidad == 0) {
        if (resp.tipo == sendOK) {
            printf("✅ Artículo eliminado del carrito\n");
        } else {
            printf("Error: %s\n", resp.datos);
        }
    } else {
        printf("%s\n", resp.datos);
    }
}

void verCarrito(SOCKET s) {
    MensajeTienda msg = (MensajeTienda){0};
    msg.tipo = CMD_CART;
    sendAll(s, (const char*)&msg, sizeof msg);

    MensajeTienda resp;
    uint32_t total_general = 0;
    int tiene_items = 0;
    int items[64] = {0}; // Para almacenar IDs mostrados
```

```
    int item_count = 0;

    printf("\n=== MI CARRITO ===\n");

    while (1) {
        if(recvAll(s, (char*)&resp, sizeof resp) < 0) break;
        if(resp.tipo == sendEnd) break;

        if(resp.tipo == sendCartItem) {
            tiene_items = 1;
            items[item_count] = resp.articulo.id;
            printf("%d. ID:%d | %s (%s) | $%.2f x %d = $%.2f\n",
                   item_count + 1,  // Número para eliminar
                   resp.articulo.id, resp.articulo.nombre, resp.articulo.marca,
                   resp.articulo.precio_cent / 100.0, resp.cantidad,
                   resp.total_articulo / 100.0);
            item_count++;
        }
        else if(resp.tipo == sendTicket) {
            total_general = resp.total_cent;
            printf("-----------------------------------\n");
            printf("Total: $%.2f\n", total_general / 100.0);
        }
    }

    if (!tiene_items) {
        printf("Carrito vacio\n");
        printf("==================\n\n");
        return;
    }

    printf("==================\n\n");

    // Menú de opciones para el carrito
    int opcion_carrito;
    do {
        printf("Opciones del carrito:\n");
        printf("1. Eliminar articulo\n");
        printf("2. Modificar cantidad\n");
        printf("3. Volver al menu principal\n");
        printf("Seleccione una opcion: ");
        scanf("%d", &opcion_carrito);
        scanf("%*c"); // Limpiar buffer

        switch(opcion_carrito) {
```

```
            case 1: {
                // Eliminar artículo
                if (item_count == 0) {
                    printf("Carrito vacio\n");
                    break;
                }

                int num_eliminar;
                printf("Ingrese el numero del articulo a eliminar (1-%d): ",
item_count);
                scanf("%d", &num_eliminar);
                scanf("%*c");

                if (num_eliminar >= 1 && num_eliminar <= item_count) {
                    int id_eliminar = items[num_eliminar - 1];
                    agregar(s, id_eliminar, 0);
                    printf("Articulo eliminado del carrito.\n");
                    return;
                } else {
                    printf("Numero invalido.\n");
                }
                break;
            }
            case 2: {
                // Modificar cantidad
                if (item_count == 0) {
                    printf("Carrito vacio\n");
                    break;
                }

                int num_modificar, nueva_cantidad;
                printf("Ingrese el numero del articulo a modificar (1-%d): ",
item_count);
                scanf("%d", &num_modificar);
                printf("Ingrese la nueva cantidad (0 para eliminar): ");
                scanf("%d", &nueva_cantidad);
                scanf("%*c");

                if (num_modificar >= 1 && num_modificar <= item_count) {
                    int id_modificar = items[num_modificar - 1];
                    if (nueva_cantidad >= 0) {
                        agregar(s, id_modificar, nueva_cantidad);
                        if (nueva_cantidad == 0) {
                            printf("Articulo eliminado del carrito.\n");
                        } else {
```

```
                            printf("Cantidad actualizada.\n");
                        }
                        return; // Volver a cargar el carrito
                    } else {
                        printf("Cantidad invalida.\n");
                    }
                } else {
                    printf("Numero invalido.\n");
                }
                break;
            }
            case 3:
                printf("Volviendo al menu principal...\n");
                break;
            default:
                printf("Opcion no valida.\n");
                break;
        }
    } while (opcion_carrito != 3);
}

void finalizar(SOCKET s){
    MensajeTienda msg = (MensajeTienda){0};
    msg.tipo = CMD_CHECK;
    sendAll(s, (const char*)&msg, sizeof msg);

    MensajeTienda resp;
    while (1){
        if(recvAll(s, (char*)&resp, sizeof resp) <0) break;
        if(resp.tipo == sendOK)
            printf("%s\n", resp.datos);
        else if(resp.tipo == sendTicket)
            printf("Folio: %u | Total: $%.2f\n", resp.folio, resp.total_cent /
100.0);
        else break;
    }
}

int main(int argc, char *argv[]){
    int Opcion;
    char texto[64];
    if(argc < 3){
        printf("Uso: %s <IP> <PUERTO>\n", argv[0]);
        return 1;
    }
```

```
    WSADATA wsa; if(WSAStartup(MAKEWORD(2,2), &wsa) != 0) error("WSAStartup");

    SOCKET s = socket(AF_INET, SOCK_STREAM, 0);
    if(s == INVALID_SOCKET) error("socket");

    struct sockaddr_in srv; memset(&srv, 0, sizeof srv);
    srv.sin_family = AF_INET;
    srv.sin_port = htons((u_short)atoi(argv[2]));
    if(inet_pton(AF_INET, argv[1], &srv.sin_addr) != 1) error("inet_pton");

    if(connect(s, (struct sockaddr*)&srv, sizeof(srv)) == SOCKET_ERROR)
error("connect");

    printf("Conectado al servidor.\n");
    while (1) {
        printf("Bienvenido a la tienda en linea.\n");
        printf("Ingrese lo que desea hacer:\n");
        printf("1. Buscar productos (ejemplo: Laptop)\n");
        printf("2. Listar productos por tipo (ejemplo: Electronica)\n");
        printf("3. Agregar producto al carrito (ejemplo: 1 2)\n");
        printf("4. Ver carrito\n");
        printf("5. Finalizar compra\n");
        scanf("%d", &Opcion);
        switch (Opcion){
        case 1:
            scanf("%*c");
            printf("Ingrese el texto a buscar:\n");
            fgets(texto, sizeof(texto), stdin);
            texto[strcspn(texto, "\n")] = 0;
            break;
        case 2:
            scanf("%*c");
            printf("Ingrese el tipo de producto a listar:\n");
            fgets(texto, sizeof(texto), stdin);
            texto[strcspn(texto, "\n")] = 0;
            listar(s, texto);
            break;
        case 3:
            scanf("%*c");
            int id, cantidad;
            printf("Ingrese el ID del producto y la cantidad a agregar:\n");
            scanf("%d %d", &id, &cantidad);
            agregar(s, id, cantidad);
            break;
```

```
        case 4:
            scanf("%*c");
            verCarrito(s);
            break;
        case 5:
            scanf("%*c");
            finalizar(s);
            break;
        default:
            break;
        }
    }
    closesocket(s);
    WSACleanup();
    return 0;
}
```

Client.c

Conclusión

La práctica “Tienda en Línea” nos permitió mejorar los conceptos de comunicación entre procesos usando sockets TCP. Más allá de escribir código, sirvió para entender cómo se construye un sistema distribuido que funcione de manera confiable, estructurada y cercana a una aplicación real de comercio electrónico.

Uno de los aprendizajes clave fue la importancia de definir un lenguaje común entre cliente y servidor. Al establecer estructuras de datos compartidas y mensajes bien definidos, se logró una comunicación clara y sin ambigüedades. Además, el uso de funciones como sendAll() y recvAll() ayudó a mantener la integridad de los datos, incluso ante posibles fallos en la red.

El servidor mostró cómo manejar múltiples solicitudes de forma ordenada, respondiendo a cada comando con precisión. El cliente, por su parte, ofreció una interfaz sencilla pero efectiva, que permitió al usuario interactuar con el sistema de forma intuitiva. Juntos, ambos componentes demostraron cómo se puede simular el flujo completo de una compra en línea, desde la búsqueda de productos hasta el cierre de la transacción.

En términos técnicos, esta práctica reforzó habilidades en comunicación orientada a la conexión, sincronización de procesos, serialización de estructuras en C y diseño modular con manejo de errores. Pero también dejó claro que detrás de cada sistema funcional hay decisiones de diseño que deben pensarse con cuidado.

En resumen, este proyecto no solo consolidó conocimientos teóricos, sino que mostró cómo aplicarlos en un entorno práctico, dando forma a una aplicación distribuida que refleja los principios fundamentales del modelo Cliente–Servidor y el potencial de los sockets TCP en el desarrollo de sistemas modernos.

Referencias

González, J. D. M. (2021, 20 junio). Sockets (Cliente-Servidor).
Sockets en Java: Un sistema cliente-servidor con sockets
Universidad de Cantabria. (s.f.). *Práctica 4: Manual de Sockets en C* [PDF]. OCW Universidad de Cantabria. Recuperado el 23 de octubre de 2025, de https://ocw.unican.es/pluginfile.php/2343/course/section/2300/PIR-Practica4_ManualSocketsC.pdf
Panzarasa, N. (2021). *Socket Library in C* [Repositorio GitHub] https://github.com/nahuelpanzarasa/socket-library-c