GEBZE TECHNICAL UNIVERSITY
COMPUTER ENGINEERING FACULTY

CSE 344 SYSTEM PROGRAMMING

FINAL REPORT

ENES AYSU
1901042671

Program Description

The "Pide Shop Simulation" project aims to develop a comprehensive food production and delivery system for a pide house. Key objectives include simulating multiple threads for cooks and delivery personnel, efficiently managing customer orders, and integrating a specialized pide oven into the workflow. Cooks handle orders assigned by the manager, involving complex matrix calculations and utilizing special tools for quicker food preparation. Meals are then oven-cooked and managed within its capacity limits before being passed to delivery personnel for distribution. Delivery times are computed based on customer addresses within a defined town area, with delivery efficiency tracked and rewarded. The system requires a multithreaded internet server, proper signal handling, and logging capabilities for operational activities. Testing will involve varying client loads and assessing system responsiveness to order modifications and reconnecting clients. Ultimately, the project aims to emulate real-world complexities in food service operations while emphasizing efficiency and resource management in system programming.

Function and Processes

Structures

order_t:

- **Description:** Represents an order placed by a client.
- **Fields:**
    - int client_socket; - Socket descriptor for the client connection.
    - int order_id; - Unique identifier for the order.
    - int client_x; - X-coordinate of the client's location.
    - int client_y; - Y-coordinate of the client's location.
    - int town_size_x; - Width of the town.
    - int town_size_y; - Height of the town.
    - int is_prepared; - Flag indicating if the order is prepared.
    - int is_cooked; - Flag indicating if the order is cooked.
    - int is_delivered; - Flag indicating if the order is delivered.
    - int canceled; - Flag indicating if the order is canceled.

Global Variables

- **orders**: Array of order_t to store all orders.
- **cooked_orders_list**: Array to store the IDs of cooked orders.
- **cooked_orders_count**: Counter for the number of cooked orders.
- **total_orders**: Counter for the total number of orders.
- **oven_queue**: Queue to store orders waiting to be cooked.
- **oven_queue_size**: Size of the oven queue.
- **oven_apparatus**: Counter for the number of available oven apparatus.
- **oven_door:** Counter for the number of available oven doors.
- **cooked_orders:** Counter for the number of orders currently cooked.
- **num_delivery_personnel:** Number of delivery personnel.
- **delivery_speed:** Speed of delivery.
- **total_delivered_orders:** Counter for the total number of delivered orders.
- **cook_order_count:** Array to track the number of orders each cook handles.
- **delivery_order_count:** Array to track the number of deliveries each delivery person handles.
- **log_file:** File pointer for the log file.

Functions (Server)

log_event:

- **Description:** Logs an event with a timestamp to the log file.
- **Parameters:**
  - const char *event - Event message to log.
- **Returns:** Nothing.
- **Operation:**
  - Gets the current time.
  - Writes the event message with a timestamp to the log file.

check_all_orders_done:

- **Description:** Checks if all orders have been delivered and identifies the most hardworking cook and delivery person.
- **Parameters:**
  - void *arg - Unused argument.
- **Returns:** NULL.
- **Operation:**
  - Waits until all orders are delivered.
  - Identifies the cook and delivery person who handled the most orders.
  - Prints a thank you message to the most hardworking cook and delivery person.

cook_thread:

- **Description:** Function executed by cook threads to prepare orders.
- **Parameters:**
  - void *arg - Pointer to the cook ID.
- **Returns:** NULL.
- **Operation:**
  - Continuously checks for new orders to prepare.
  - Prepares the order in random time (I couldn’t implement inverse matrix, instead of I assigned random sleep time between 2 – 7 second.)
  - If oven door and oven apparatus statuses are okay places the order in the oven.
  - Tracks the number of orders each cook handles.

oven_thread:

- **Description:** Function executed by the oven thread to cook orders.
- **Parameters:**
    - void *arg - Unused argument.
- **Returns:** NULL.
- **Operation:**
    - Continuously checks for orders in the oven queue.
    - Simulates cooking the orders in 5 seconds.
    - Marks orders as cooked and signals the delivery thread.

calculate_manhattan_distance:

- **Description:** Calculates the Manhattan distance between two points.
- **Parameters:**
    - int x1, int y1 - Coordinates of the first point.
    - int x2, int y2 - Coordinates of the second point.
- **Returns:** The Manhattan distance.
- **Operation:**
    - Calculates the sum of the absolute differences of the coordinates.

delivery_thread:

- **Description:** Function executed by delivery threads to deliver orders.
- **Parameters:**
    - void *arg - Pointer to the delivery person ID.
- **Returns:** NULL.
- **Operation:**
    - Continuously checks for cooked orders to deliver.
    - Delivers the orders according to their locations and updates the delivery status.
    - Tracks the number of deliveries each delivery person handles.

handle_client_order:

- **Description:** Handles an incoming client order.
- **Parameters:**
    - void *arg - Pointer to the client socket.
- **Returns:** NULL.
- **Operation:**
    - Reads the order details from the client.
    - Creates a new order and adds it to the orders array.
    - Sends a response to the client indicating the order is being processed.

handle_client:

- **Description:** Handles incoming client connections.
- **Parameters:**
    - void *arg - Pointer to the server socket.
- **Returns:** NULL.
- **Operation:**
    - Continuously accepts new client connections.
    - Creates a new thread to handle each client order.

handle_signal:

- **Description:** Handles cancellation signals (e.g., SIGINT) to shut down the server.
- **Parameters:**
    - int sig - Signal number.
- **Returns:** Nothing.
- **Operation:**
    - Sets the canceled flag for all orders.
    - Broadcasts condition variables to unblock threads.
    - Closes the log file and exits the program.

Main:

- **Description:** Entry point of the program. Initializes the server and starts the threads.
- **Parameters:**
  - int argc - Argument count.
  - char *argv[] - Argument vector.
- **Returns:** Exit status.
- **Operation:**
  - Validates command-line arguments.
  - Initializes server socket and binds it to the specified port.
  - Sets up signal handling for graceful shutdown.
  - Creates and detaches threads for client handling, cooking, oven, and delivery.
  - Enters an infinite loop to keep the main thread running.

Example Execution Flow for Server

- **Server Initialization:**
  - The main function initializes the server and starts the necessary threads for client handling, cooking, oven management, and delivery.
- **Order Handling:**
  - A client connects to the server and sends an order.
  - The handle_client function accepts the connection and creates a new thread to handle the order.
  - The handle_client_order function reads the order details, creates a new order_t entry, and sends a response to the client.
- **Cooking:**
  - The cook_thread function continuously checks for new orders to prepare.
  - Once an order is prepared, it is placed in the oven queue.

- **Oven Management:**
    - The oven_thread function continuously checks the oven queue and simulates cooking the orders.
    - Cooked orders are marked as ready for delivery.
- **Delivery:**
    - The delivery_thread function continuously checks for cooked orders to deliver.
    - The delivery person delivers the orders and updates the delivery status.
- **Completion Check:**
    - The check_all_orders_done function checks if all orders are delivered and identifies the most hardworking cook and delivery person.
- **Graceful Shutdown:**
    - If a cancellation signal (e.g., SIGINT) is received, the handle_signal function sets the canceled flag for all orders and unblocks all threads, allowing them to terminate gracefully.

Functions (Client)

client_thread:

- **Description:** Handles the client's connection to the server, sends initial messages, and processes server responses.
- **Parameters:**
  - void* arg - Pointer to the client ID.
- **Returns:** NULL
- **Operation:**
  - Creates a client socket using socket(AF_INET, SOCK_STREAM, 0).
  - Sets up the server address structure server_addr with AF_INET, the specified port, and the server IP address.
  - Connects to the server using connect(client_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)).
  - Prints a message indicating the client connection.
  - Generates random coordinates within the town (client_x and client_y).
  - Sends a message containing the client ID and coordinates to the server.
  - Reads and prints the initial server response.
  - Reads and prints the delivery message from the server.
  - Closes the client socket.

sigint_handler:

- **Description:** Handles SIGINT signals (e.g., Ctrl+C) to perform a graceful shutdown.
- **Parameters:**
  - int signum - Signal number.
- **Returns:** Nothing.
- **Operation:**
  - Prints messages indicating order cancellations and that the server is being notified.
  - Exits the program with exit(EXIT_SUCCESS).

Main:

- **Description:** Initializes the client program, sets up signal handling, and creates client threads.
- **Parameters:**
  - int argc - Number of command-line arguments.
  - char *argv[] - Array of command-line arguments.
- **Returns:** int - Exit status.
- **Operation:**
  - Checks if the correct number of arguments are provided. If not, it prints the usage message and exits.
  - Parses the command-line arguments to set port, number_of_clients, town_size_x, and town_size_y.
  - Sets the server IP to 127.0.0.1.
  - Sets up signal handling for SIGINT using sigaction.
  - Creates an array of pthread_t for client threads and an array of client IDs.
  - Iterates over the number of clients, creating a thread for each client using pthread_create.
  - Waits for all client threads to finish using pthread_join.
  - Prints a message indicating all clients have been served.

Tests

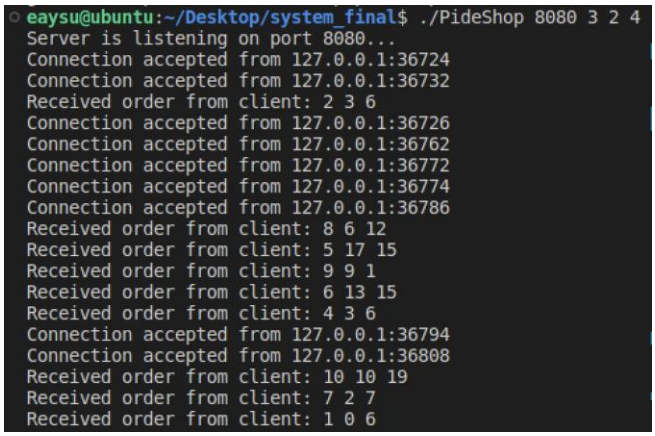
For Compilation

- $ make
- $ ./PideShop [portnumber] [CookthreadPoolSize] [DeliveryPoolSize] [k]
- $ ./HungryVeryMuch [portnumber] [numberOfClients] [p] [q]

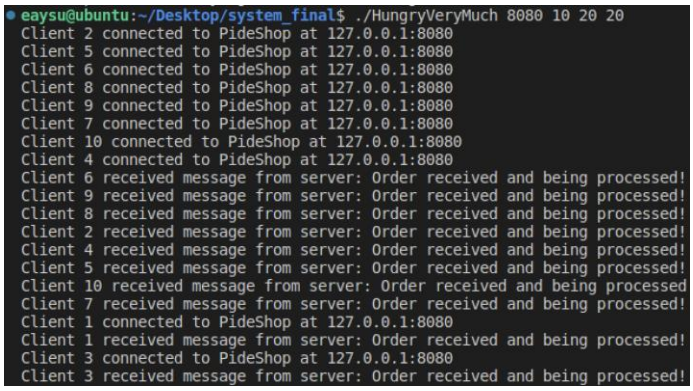
Note: For less waiting durations, keep p and q values small (10 – 30), make k value bigger (5 – 10). My implementations works better with less cook and less delivery (2 – 4). Additionaly port number is stable and its 127.0.0.1. Just give same port number both server and client.

Test 1

Server Side



Client Side



```
Cook 1 placed order 2 in oven...
Cook 1 preparing order 4...
Cook 2 placed order 1 in oven...
Cook 2 preparing order 5...
Cook 3 placed order 3 in oven...
Cook 3 preparing order 6...
Cook 1 placed order 4 in oven...
Cook 1 preparing order 7...
Cook 3 placed order 6 in oven...
Cook 3 preparing order 8...
Cook 2 placed order 5 in oven...
Cook 2 preparing order 9...
Order 2 cooked
Order 1 cooked
Order 3 cooked
Order 4 cooked
Order 6 cooked
Order 5 cooked
Delivery person 1 is delivering 3 orders... order ids: 2 1 3
Delivery person 1 is delivering 3 orders... order ids: 4 6 5
Cook 1 placed order 7 in oven...
Cook 1 preparing order 10...
Cook 2 placed order 9 in oven...
Cook 3 placed order 8 in oven...
Cook 1 placed order 10 in oven...
Order 7 cooked
Order 9 cooked
Order 8 cooked
Order 10 cooked
Delivery person 2 is delivering 3 orders... order ids: 7 9 8
Delivery person 2 is delivering 1 orders... order ids: 10
Done serving client
Waiting for another orders...
```

```
Client 8 received delivery message from server: Order 8
delivered in 14 seconds
Client 2 received delivery message from server: Order 2
delivered in 14 seconds
Client 5 received delivery message from server: Order 5
delivered in 14 seconds
Client 9 received delivery message from server: Order 9
delivered in 14 seconds
Client 4 received delivery message from server: Order 4
delivered in 14 seconds
Client 6 received delivery message from server: Order 6
delivered in 14 seconds
Client 10 received delivery message from server: Order 1
0 delivered in 14 seconds
Client 1 received delivery message from server: Order 1
delivered in 14 seconds
Client 7 received delivery message from server: Order 7
delivered in 14 seconds
Client 3 received delivery message from server: Order 3
delivered in 2 seconds
All clients served.
```

Test 2

After completing one client's orders server still alive and can accept new new client's order.

```
Done serving client
Waiting for another orders...
Connection accepted from 127.0.0.1:57558
Connection accepted from 127.0.0.1:57554
Connection accepted from 127.0.0.1:57556
```

Test 3

When CTRL^C comes, server can able to close itself.

```
Done serving client
Waiting for another orders...
^CCancellation signal received, shutting down server...
eaysu@ubuntu:~/Desktop/system_final$
```

Test 4

Testing the system with different parameters.

```
eaysu@ubuntu:~/Desktop/system_final$ ./PideShop 8088 3 3 5
```

```
eaysu@ubuntu:~/Desktop/system_final$ ./HungryVeryMuch 8088 30 30 20
```

```
Server is listening on port 8088...
Connection accepted from 127.0.0.1:51542
Connection accepted from 127.0.0.1:51532
Connection accepted from 127.0.0.1:51548
Connection accepted from 127.0.0.1:51560
Connection accepted from 127.0.0.1:51562
Connection accepted from 127.0.0.1:51576
Connection accepted from 127.0.0.1:51584
Connection accepted from 127.0.0.1:51592
Connection accepted from 127.0.0.1:51596
Received order from client: 5 7 15
Received order from client: 10 23 15
Received order from client: 1 11 8
Received order from client: 4 27 15
Received order from client: 2 13 6
Received order from client: 7 20 19
Received order from client: 8 2 10
Connection accepted from 127.0.0.1:51604
Connection accepted from 127.0.0.1:51622
Connection accepted from 127.0.0.1:51620
Connection accepted from 127.0.0.1:51646
Connection accepted from 127.0.0.1:51650
Received order from client: 12 9 1
Connection accepted from 127.0.0.1:51742
Connection accepted from 127.0.0.1:51744
Received order from client: 15 2 3
Received order from client: 3 23 6
Received order from client: 19 0 6
Received order from client: 11 16 12
Received order from client: 13 2 7
Received order from client: 14 22 16
```

```
Received order from client: 29 29 2
Received order from client: 28 12 18
Connection accepted from 127.0.0.1:51590
Received order from client: 6 27 9
Cook 3 preparing order 1...
Cook 1 preparing order 2...
Cook 2 preparing order 3...
Connection accepted from 127.0.0.1:51636
Connection accepted from 127.0.0.1:51730
Connection accepted from 127.0.0.1:51726
Connection accepted from 127.0.0.1:51716
Connection accepted from 127.0.0.1:51712
Received order from client: 16 29 7
Connection accepted from 127.0.0.1:51704
Connection accepted from 127.0.0.1:51690
Connection accepted from 127.0.0.1:51706
Connection accepted from 127.0.0.1:51688
Received order from client: 27 1 2
Received order from client: 20 9 13
Connection accepted from 127.0.0.1:51752
Received order from client: 25 21 19
Connection accepted from 127.0.0.1:51678
Received order from client: 24 8 4
Received order from client: 26 1 0
Connection accepted from 127.0.0.1:51682
Received order from client: 9 13 16
Received order from client: 21 5 10
Received order from client: 22 14 17
Received order from client: 18 2 10
Received order from client: 30 26 13
```

```
Connection accepted from 127.0.0.1:51664
Received order from client: 23 23 6
Received order from client: 17 16 1
Cook 2 placed order 3 in oven...
Cook 2 preparing order 4...
Cook 3 placed order 1 in oven...
Cook 3 preparing order 5...
Cook 1 placed order 2 in oven...
Cook 1 preparing order 6...
Cook 2 placed order 4 in oven...
Cook 2 preparing order 7...
Cook 1 placed order 6 in oven...
Cook 1 preparing order 8...
Order 3 cooked
Order 1 cooked
Order 2 cooked
Order 4 cooked
Order 6 cooked
Delivery person 1 is delivering 3 orders... order ids: 3 1 2
Delivery person 1 is delivering 2 orders... order ids: 4 6
Cook 3 placed order 5 in oven...
Cook 2 placed order 7 in oven...
Cook 2 preparing order 9...
Cook 3 preparing order 10...
Cook 1 placed order 8 in oven...
Cook 1 preparing order 11...
Order 5 cooked
Order 7 cooked
Order 8 cooked
Delivery person 2 is delivering 3 orders... order ids: 5 7 8
```

```
Cook 2 placed order 9 in oven...
Cook 2 preparing order 12...
Cook 3 placed order 10 in oven...
Cook 3 preparing order 13...
Cook 1 placed order 11 in oven...
Cook 1 preparing order 14...
Cook 2 placed order 12 in oven...
Cook 2 preparing order 15...
Cook 1 placed order 14 in oven...
Cook 1 preparing order 16...
Cook 3 placed order 13 in oven...
Cook 3 preparing order 17...
Order 9 cooked
Order 10 cooked
Order 11 cooked
Order 12 cooked
Order 14 cooked
Order 13 cooked
Delivery person 3 is delivering 3 orders... order ids: 9 10 11
Delivery person 3 is delivering 3 orders... order ids: 12 14 13
Cook 1 placed order 16 in oven...
Cook 1 preparing order 18...
Cook 3 placed order 17 in oven...
Cook 3 preparing order 19...
Order 16 cooked
Order 15 cooked
Order 17 cooked
Cook 2 placed order 15 in oven...
Cook 2 preparing order 20...
Delivery person 1 is delivering 3 orders... order ids: 16 15 17
```

```
Cook 1 placed order 18 in oven...
Cook 1 preparing order 21...
Cook 3 placed order 19 in oven...
Cook 3 preparing order 22...
Cook 2 placed order 20 in oven...
Cook 2 preparing order 23...
Cook 2 placed order 23 in oven...
Cook 2 preparing order 24...
Cook 1 placed order 21 in oven...
Cook 1 preparing order 25...
Cook 3 placed order 22 in oven...
Cook 3 preparing order 26...
There is no space for new order in oven
Order 18 cooked
Order 19 cooked
Order 20 cooked
Order 23 cooked
Order 21 cooked
Order 22 cooked
Delivery person 2 is delivering 3 orders... order ids: 18 19 20
Delivery person 2 is delivering 3 orders... order ids: 23 21 22
Cook 1 placed order 25 in oven...
Cook 2 placed order 24 in oven...
Cook 1 preparing order 27...
Cook 3 placed order 26 in oven...
Cook 2 preparing order 28...
Order 25 cooked
Order 24 cooked
Order 26 cooked
Cook 3 preparing order 29...
```

```
Delivery person 3 is delivering 3 orders... order ids: 25 24 26
Cook 2 placed order 28 in oven...
Cook 2 preparing order 30...
Cook 3 placed order 29 in oven...
Cook 1 placed order 27 in oven...
Cook 2 placed order 30 in oven...
Order 28 cooked
Order 29 cooked
Order 27 cooked
Order 30 cooked
Delivery person 1 is delivering 3 orders... order ids: 28 29 27
Delivery person 1 is delivering 1 orders... order ids: 30
Done serving client
Waiting for another orders...
```

```
Client 7 received delivery message from server: Order 7 delivered in 14 seconds
Client 5 received delivery message from server: Order 5 delivered in 14 seconds
Client 2 received delivery message from server: Order 2 delivered in 14 seconds
Client 1 received delivery message from server: Order 1 delivered in 7 seconds
Client 10 received delivery message from server: Order 10 delivered in 7 seconds
Client 4 received delivery message from server: Order 4 delivered in 14 seconds
Client 8 received delivery message from server: Order 8 delivered in 14 seconds
Client 12 received delivery message from server: Order 12 delivered in 14 seconds
Client 15 received delivery message from server: Order 15 delivered in 14 seconds
Client 3 received delivery message from server: Order 3 delivered in 14 seconds
Client 19 received delivery message from server: Order 19 delivered in 14 seconds
Client 11 received delivery message from server: Order 11 delivered in 14 seconds
Client 14 received delivery message from server: Order 14 delivered in 14 seconds
Client 13 received delivery message from server: Order 13 delivered in 14 seconds
Client 28 received delivery message from server: Order 28 delivered in 14 seconds
Client 29 received delivery message from server: Order 29 delivered in 14 seconds
Client 6 received delivery message from server: Order 6 delivered in 14 seconds
Client 16 received delivery message from server: Order 16 delivered in 14 seconds
Client 27 received delivery message from server: Order 27 delivered in 14 seconds
Client 20 received delivery message from server: Order 20 delivered in 14 seconds
Client 9 received delivery message from server: Order 9 delivered in 14 seconds
Client 25 received delivery message from server: Order 25 delivered in 14 seconds
Client 24 received delivery message from server: Order 24 delivered in 14 seconds
Client 22 received delivery message from server: Order 22 delivered in 14 seconds
Client 21 received delivery message from server: Order 21 delivered in 14 seconds
Client 18 received delivery message from server: Order 18 delivered in 14 seconds
Client 26 received delivery message from server: Order 26 delivered in 14 seconds
Client 23 received delivery message from server: Order 23 delivered in 14 seconds
Client 30 received delivery message from server: Order 30 delivered in 14 seconds
Client 17 received delivery message from server: Order 17 delivered in 4 seconds
All clients served.
```

Summary

The client-server program demonstrates a practical implementation of concurrent client-server communication using socket programming and multithreading in C. The server sets up a listening socket on a specified port, waiting for incoming client connections. Each client connection is handled in a separate thread, allowing the server to process multiple client requests concurrently. The server reads client messages, simulates order processing, and sends appropriate responses back to the clients. The program also incorporates signal handling to manage graceful shutdown when a SIGINT signal is received, ensuring that the server can clean up resources and exit properly.

On the client side, each client establishes a connection to the server, sends its client ID and coordinates, and waits for responses from the server. The client receives an initial acknowledgment and a subsequent delivery message from the server, indicating the status of its order. Each client runs in a separate thread to simulate multiple clients interacting with the server simultaneously.

By integrating both the client and server functionalities, this program showcases a robust client-server architecture capable of handling concurrent connections efficiently. It provides a foundational example of how to build scalable and responsive networked applications, emphasizing the importance of proper synchronization, resource management, and graceful handling of shutdown signals.