

**GEBZE TECHNICAL UNIVERSITY**

CSE 344  
SYSTEM PROGRAMMING

Final Report

**PideShop Server-Client Application**

**BARIŞ YURDAKUL**  
1801042620

June 15, 2024

# Contents

<b>1</b>	<b>Problem Definition</b>	<b>3</b>
1.1	Challenges . . . . .	3
1.2	Requirements . . . . .	3
<b>2</b>	<b>Implementation</b>	<b>3</b>
2.1	Order Queue Management . . . . .	3
2.2	Cooking and Delivery Thread Management . . . . .	4
2.3	Client Handling . . . . .	5
2.4	Logging and Reporting . . . . .	6
<b>3</b>	<b>Synchronization Mechanisms</b>	<b>6</b>
3.1	Mutexes . . . . .	6
3.2	Condition Variables . . . . .	7
3.3	Example: Cook Thread . . . . .	7
3.4	Example: Delivery Thread . . . . .	8
<b>4</b>	<b>Result and Test Cases</b>	<b>8</b>
4.1	Results . . . . .	8
4.2	Test Cases . . . . .	8
4.3	Screenshots . . . . .	9

# 1 Problem Definition

In the context of a modern food delivery system, particularly a PideShop, managing order processing, cooking, and delivery poses several challenges. The primary objective of this project is to design and implement a server-client application capable of efficiently handling these tasks.

## 1.1 Challenges

- **Order Management:** Handle orders arriving asynchronously from multiple clients, ensuring each order is queued and processed in sequence.
- **Cooking Management:** Assign orders to cooks, manage cooking time and oven capacity, and ensure optimal use of resources.
- **Delivery Management:** Manage a queue of cooked orders, assign couriers, and ensure timely and efficient delivery.
- **Concurrency:** Handle multiple clients and workers concurrently, using robust synchronization mechanisms to avoid race conditions.
- **Logging and Reporting:** Maintain detailed logs and generate summary reports upon shutdown, recognizing top-performing workers.

## 1.2 Requirements

- **Scalability:** Handle varying numbers of clients, orders, cooks, and couriers without performance degradation.
- **Fault Tolerance:** Handle unexpected conditions without crashing.
- **Efficiency:** Optimize resource use to minimize wait times and ensure timely deliveries.
- **User Feedback:** Provide feedback to clients about their order status and maintain logs for administrative purposes.

By addressing these challenges, this project aims to develop a comprehensive server-client application for a PideShop, ensuring efficient order processing, cooking, and delivery, while maintaining logs and generating reports.

# 2 Implementation

## 2.1 Order Queue Management

The server maintains two queues: one for orders waiting to be cooked and another for orders ready for delivery.

```

typedef struct Order {
    int client_socket;
    int order_id;
    int x, y;
    pid_t client_pid;
    struct Order* next;
} Order;

typedef struct {
    Order* front;
    Order* rear;
    int size;
    pthread_mutex_t mutex;
} OrderQueue;

OrderQueue order_queue;
OrderQueue delivery_queue;

```

- **Order:** Represents a client's order, including its ID, coordinates, client's PID, and socket information.
- **OrderQueue:** A linked list structure for managing orders, protected by a mutex for thread safety.

#### Functions:

- `void enqueue(OrderQueue* queue, Order* order):` Adds an order to the queue.
- `Order* dequeue(OrderQueue* queue):` Removes an order from the queue.

## 2.2 Cooking and Delivery Thread Management

Cooks and couriers are represented by worker threads, each processing orders in their respective pools.

```

typedef struct {
    int id;
    bool available;
    pthread_cond_t cond;
    int orders_processed;
} Worker;

Worker* cooks;
Worker* couriers;
int cook_thread_pool_size;
int delivery_thread_pool_size;

```

- **Worker:** Represents a cook or a courier, including ID, availability status, condition variable for synchronization, and the number of processed orders.

#### **Thread Functions:**

- `void* cook_thread(void* arg):` Manages cooking process.
- `void* delivery_thread(void* arg):` Manages delivery process.

#### **Cook Thread Logic:**

1. Wait for an order in the queue.
2. Process the order (simulate cooking).
3. Move the order to the delivery queue.

#### **Delivery Thread Logic:**

1. Wait for cooked orders in the delivery queue.
2. Deliver the orders to the client's location.
3. Update the delivery count and free resources.

## **2.3 Client Handling**

Clients connect to the server, send their orders, and wait for the delivery. The server handles incoming connections and assigns orders to the appropriate queues.

```
typedef struct {
    pid_t pid;
    int numberOfClients;
    bool announced;
    int orders_to_serve;
} ClientInfo;

ClientInfo clients[MAX_CLIENTS];
int client_count = 0;
```

#### **Client Connection Handling:**

1. Accept incoming client connections.
2. Receive order details from clients.
3. Enqueue orders into the order queue.
4. Maintain client information for logging and tracking.

## 2.4 Logging and Reporting

The server maintains a log file to record activities and provides a summary report upon shutdown.

```
FILE* log_file;
```

```
void handle_sigint(int sig);
```

```
void thank_most_orders(Worker* workers, int size, const char* role);
```

- **Log File:** Records each significant event, such as new orders, cooking start and end, deliveries, and shutdown summary.
- **Signal Handling:** Captures SIGINT for graceful shutdown, ensuring all threads are terminated and resources are cleaned up.
- **Thank You Messages:** Displays and logs thank you messages for the most productive workers.

**Functions:**

- `void thank most orders(Worker* workers, int size, const char* role):` Identifies and thanks the workers with the highest order counts.

## 3 Synchronization Mechanisms

Synchronization is crucial for managing multiple clients and workers concurrently. We use mutexes and condition variables to ensure thread safety and coordination.

### 3.1 Mutexes

Mutexes are used to protect shared data structures, such as order queues.

```
pthread_mutex_t mutex;  
pthread_mutex_init(&mutex, NULL);
```

```
// Locking  
pthread_mutex_lock(&mutex);  
// Critical section  
pthread_mutex_unlock(&mutex);
```

### 3.2 Condition Variables

Condition variables are used to block threads until a particular condition is met, such as waiting for new orders in the queue.

```
pthread_cond_t cond;
pthread_cond_init(&cond, NULL);

// Waiting for a condition
pthread_mutex_lock(&mutex);
while (condition_not_met) {
    pthread_cond_wait(&cond, &mutex);
}
// Critical section
pthread_mutex_unlock(&mutex);

// Signaling a condition
pthread_mutex_lock(&mutex);
condition_met = 1;
pthread_cond_signal(&cond);
pthread_mutex_unlock(&mutex);
```

### 3.3 Example: Cook Thread

The cook thread waits for an order to be available, processes it, and then moves it to the delivery queue.

```
void* cook_thread(void* arg) {
    while (1) {
        pthread_mutex_lock(&order_queue.mutex);
        while (order_queue.size == 0) {
            pthread_cond_wait(&order_queue.cond, &order_queue.mutex);
        }
        Order* order = dequeue(&order_queue);
        pthread_mutex_unlock(&order_queue.mutex);

        // Simulate cooking
        sleep(cooking_time);

        pthread_mutex_lock(&delivery_queue.mutex);
        enqueue(&delivery_queue, order);
        pthread_cond_signal(&delivery_queue.cond);
        pthread_mutex_unlock(&delivery_queue.mutex);
    }
}
```

### 3.4 Example: Delivery Thread

The delivery thread waits for cooked orders and delivers them to clients.

```
void* delivery_thread(void* arg) {
    while (1) {
        pthread_mutex_lock(&delivery_queue.mutex);
        while (delivery_queue.size == 0) {
            pthread_cond_wait(&delivery_queue.cond, &delivery_queue.mutex);
        }
        Order* order = dequeue(&delivery_queue);
        pthread_mutex_unlock(&delivery_queue.mutex);

        // Simulate delivery
        sleep(delivery_time);

        // Update delivery count
        pthread_mutex_lock(&stats_mutex);
        deliveries++;
        pthread_mutex_unlock(&stats_mutex);
    }
}
```

## 4 Result and Test Cases

### 4.1 Results

The implementation successfully handles multiple client connections, processes orders efficiently using cook and courier threads, and maintains a log of all activities. Upon shutdown, a summary report of total orders and deliveries, along with a thank you message to the most productive workers, is generated.

### 4.2 Test Cases

- **Single Client Order:** Connect a single client to the server, place an order, verify processing and delivery, and check log entries.
- **Multiple Client Orders:** Connect multiple clients, place orders, verify processing and delivery for all, and check log entries.
- **High Load:** Simulate a high number of clients and orders, ensure stability, and verify logs.
- **Graceful Shutdown:** Send SIGINT to the server, ensure all threads terminate gracefully, and check the summary report and logs.



### 4.3 Screenshots

```
Thanks Moto 5 (Orders: 22)
baris@baris-Ubuntu:~/Desktop/system_final$ ./PideShop 8081 4 6 1
PideShop active waiting for connections...
100 new customers... Serving
Order 1 placed from location (7, 13) by client PID 84426
Cook 3 is cooking order 1...
Cook 3 completed cooking for order 1. Delivery of the order
```

Figure 1: Server initialization and handling multiple client orders.

```
Order placed from location (7, 13)
baris@baris-Ubuntu:~/Desktop/system_final$ ./HungryVeryMuch 127.0.0.1 8081 100 10 20
HungryVeryMuch client PID: 84426
Order placed from location (7, 13)
Order placed from location (3, 4)
Order placed from location (2, 4)
Order placed from location (1, 14)
Order placed from location (7, 14)
Order placed from location (7, 2)
```

Figure 2: Client connections and order placements from various locations.

```
Order 90 delivered by Moto 4.
Delivering order 99 to location (2, 5)...
Order 90 delivered by Moto 2.
Delivering order 91 to location (3, 3)...
Order 99 delivered by Moto 4.
Order 91 delivered by Moto 2.
Order 94 delivered by Moto 1.
Order 97 delivered by Moto 5.
^C
Shutting down PideShop...
Total orders: 100, Delivered: 98
Thanks Cook 4 (Orders: 34)
Thanks Moto 5 (Orders: 22)
```

Figure 3: Server processing orders and delivering them, with shutdown summary.

```

baris@baris-Ubuntu:~/Desktop/system_final$ ./HungryVeryMuch 127.0.0.1 8081 5 10 20
HungryVeryMuch client PID: 83370
Order placed from location (7, 10)
Order placed from location (3, 16)
Order placed from location (1, 7)
Order placed from location (6, 11)
Order placed from location (5, 17)
baris@baris-Ubuntu:~/Desktop/system_final$

baris@baris-Ubuntu:~/Desktop/system_final$ ./HungryVeryMuch 127.0.0.1 8081 3 5 10
HungryVeryMuch client PID: 84145
Order placed from location (2, 0)
Order placed from location (0, 4)
Order placed from location (2, 3)
baris@baris-Ubuntu:~/Desktop/system_final$

```

Figure 4: Multiple clients placing orders simultaneously.

```

baris@baris-Ubuntu:~/Desktop/system_final$ ./HungryVeryMuch 127.0.0.1 8081 3 10 20
HungryVeryMuch client PID: 84133
Order placed from location (7, 9)
Order placed from location (5, 10)
Order placed from location (7, 18)
baris@baris-Ubuntu:~/Desktop/system_final$

```

Figure 5: Client placing a few orders for testing.

```

Moto 1 is on the way with 1 orders...
Order placed from location (0, 2)
Moto 2 is on the way with 2 orders...
Order placed from location (0, 2)
Moto 5 is on the way with 3 orders...

```

Figure 6: Moto could carry 1, 2 or 3 orders at the same time.

```

Delivering order 1 to location (4, 14)...
Moto 2 is waiting for orders...
^C
Shutting down PideShop...
Total orders: 3, Delivered: 0
Thanks Cook 3 (Orders: 2)
Thanks Moto 1 (Orders: 0)
Thanks Moto 2 (Orders: 0)
Thanks Moto 3 (Orders: 0)
Thanks Moto 4 (Orders: 0)
Thanks Moto 5 (Orders: 0)
Thanks Moto 6 (Orders: 0)

```

Figure 7: Graceful shutdown with summary report and thank you messages for top performers.

```

Order 3 placed from location (1, 7) by client PID 83370
Cook 2 is cooking order 3...
Cook 2 completed cooking for order 3, taken out of the oven...
Moto 1 is on the way with 1 orders...
Delivering order 1 to location (7, 10)...
Moto 2 is waiting for orders...
Order 3 is ready for delivery.
Order 4 placed from location (6, 11) by client PID 83370
Cook 1 is cooking order 4...
Cook 1 completed cooking for order 4, taken out of the oven...
Order 4 is ready for delivery.
Order 5 placed from location (5, 17) by client PID 83370
Cook 3 is cooking order 5...
Cook 3 completed cooking for order 5, taken out of the oven...
Moto 2 is on the way with 1 orders...
Delivering order 2 to location (3, 16)...
Moto 3 is waiting for orders...
Order 5 is ready for delivery.
Moto 3 is on the way with 1 orders...
Moto 4 is waiting for orders...
Delivering order 3 to location (1, 7)...
Moto 4 is on the way with 1 orders...
Delivering order 4 to location (6, 11)...
Moto 5 is waiting for orders...
Moto 5 is on the way with 1 orders...
Delivering order 5 to location (5, 17)...
Order 3 delivered by Moto 3.
Order 1 delivered by Moto 1.
Order 2 delivered by Moto 2.
Order 4 delivered by Moto 4.
Order 5 delivered by Moto 5.
Done serving client PID 83370

```

Figure 8: Order placement and processing by different cooks and couriers.

```

Order 10 placed from location (0, 4) by client PID 84145
Cook 2 is cooking order 10...
Cook 2 completed cooking for order 10, taken out of the oven...
Order 10 is ready for delivery.
Order 11 placed from location (2, 3) by client PID 84145
Cook 3 is cooking order 11...
Cook 3 completed cooking for order 11, taken out of the oven...
Moto 2 is on the way with 1 orders...
Delivering order 9 to location (2, 0)...
Moto 4 is waiting for orders...
Order 11 is ready for delivery.
Order 9 delivered by Moto 2.
Moto 4 is on the way with 1 orders...
Delivering order 10 to location (0, 4)...
Moto 5 is waiting for orders...
Moto 5 is on the way with 1 orders...
Delivering order 11 to location (2, 3)...
Order 10 delivered by Moto 4.
Order 6 delivered by Moto 6.
Order 11 delivered by Moto 5.
Done serving client PID 84145
Order 8 delivered by Moto 3.
Delivering order 7 to location (5, 10)...
Order 7 delivered by Moto 3.
Done serving client PID 84133
^C
Shutting down PideShop...
Total orders: 11, Delivered: 11
Thanks Cook 1 (Orders: 4)
Thanks Cook 2 (Orders: 4)
Thanks Moto 3 (Orders: 3)
haris@baris-Ubuntu:~/Desktop/system_final$

```

Figure 9: Server handling orders, deliveries, and shutdown summary.

```
Order 2 is ready for delivery.
Order 3 placed from location (4, 3) by client PID 85284
Cook 2 is cooking order 3...
Cook 2 completed cooking for order 3, taken out of the oven...
Order 3 is ready for delivery.
Moto 1 is on the way with 1 orders...
Delivering order 1 to location (4, 14)...
Moto 2 is waiting for orders...

Shutting down PideShop...
Total orders: 3, Delivered: 0
Thanks Cook 3 (Orders: 2)
Thanks Moto 1 (Orders: 0)
Thanks Moto 2 (Orders: 0)
Thanks Moto 3 (Orders: 0)
Thanks Moto 4 (Orders: 0)
Thanks Moto 5 (Orders: 0)
Thanks Moto 6 (Orders: 0)
100 new customers... Serving
Order 1 placed from location (6, 4) by client PID 87006
Cook 1 is cooking order 1...
Cook 1 completed cooking for order 1, taken out of the oven...
Order 1 is ready for delivery.
Moto 1 is waiting for orders...
Order 2 placed from location (1, 1) by client PID 87006
Cook 1 is cooking order 2...
```

Figure 10: Part of Log file.