CSE344-SYSTEMS PROGRAMMİNG

FINAL PROJECT REPORT

ADEM MURAT ÖZDEMİR-200104004110

Compiling and Running

Compile the codes : “make”

Run the Program : Firstly start the server

Run server : “./PideShop <port_number> <num_cook> <num_delivery> <delivery_time>”

Run client : “./HungryVeryMuch <ip_number> <port_number> <order_count> <X_distance> <Y_distance>”

Problem Overview

In this project, we aim to simulate a pide shop where clients can place orders, and the orders are prepared and delivered by the shop's staff. The primary focus is on managing the orders efficiently, ensuring they are prepared and delivered promptly, and handling cancellation requests from clients. The system is designed to handle multiple clients, each placing multiple orders, and it ensures that the shop can process these orders concurrently.

Problem Description

The pide shop simulation involves several key components:

1. **Client Program**:
    - The client program connects to the server and sends multiple orders.
    - Each order contains a unique client ID and delivery coordinates.
    - The client can cancel all orders by sending a specific message ("CANCEL ALL ORDERS").
    - The client program handles signal interrupts (e.g., Ctrl+C) to send a cancellation message to the server.
2. **Server Program**:
    - The server accepts connections from multiple clients.
    - For each client, the server receives the number of orders and the details of each order.
    - The server has a team of cooks and delivery persons working concurrently to prepare and deliver the orders.

- The server manages queues for incoming orders and cooked orders.
- The server listens for cancellation messages from the clients and stops processing if a cancellation message is received.
- The server logs the activities and handles cleanup properly.

How to Solve

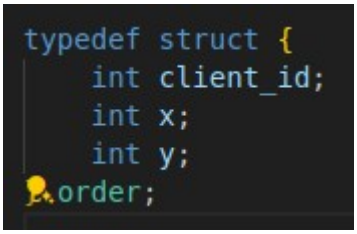
To address the problem, the solution involves the following steps:

1. **Client Implementation**:
    - Create a socket to connect to the server.
    - Generate random orders with unique client IDs and coordinates.
    - Send the orders to the server.
    - Handle signal interrupts to send a cancellation message to the server.
    - Print the server's response and close the connection.
2. **Server Implementation**:
    - Set up a socket to listen for incoming client connections.
    - Accept a client connection and receive the number of orders and their details.
    - Initialize queues for managing orders and cooked orders.
    - Create threads for cooks and delivery persons to process the orders concurrently.
    - Implement a listener thread to handle cancellation messages from the client.
    - Use mutexes and condition variables to synchronize access to shared resources.
    - Log all activities to a log file.
    - Clean up resources and reset global variables after processing each client.
3. **Synchronization and Concurrency**:
    - Use mutexes to protect shared data structures like queues.
    - Use condition variables to signal the availability of new orders or cooked orders.
    - Use semaphores to manage the limited resources (e.g., oven capacity).
    - Ensure that cancellation requests are handled promptly by broadcasting signals to all waiting threads.
4. **Signal Handling**:
    - Set up signal handlers to manage SIGINT (e.g., Ctrl+C) to send cancellation messages from the client.
    - On the server side, handle SIGINT to log the shutdown process and clean up resources properly.

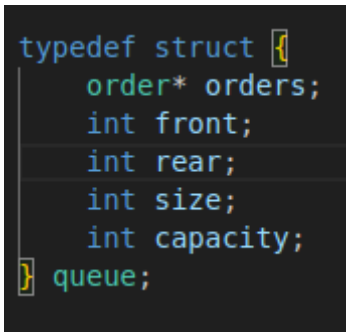
Structs That Are Used :

The order structure holds information for each client's order:

- client_id: Unique identifier for the client.
- x: x-coordinate of the delivery location.
- y: y-coordinate of the delivery location.

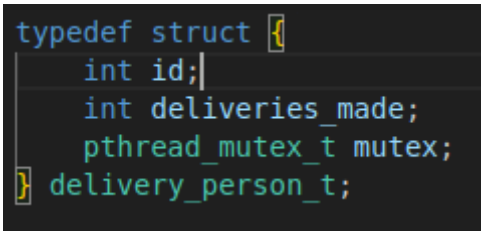


The queue structure is designed to manage a collection of orders in a circular queue format, allowing for efficient order enqueuing and dequeuing. It keeps track of the front and rear indices, the current size, and the maximum capacity of the queue.



The delivery_person_t structure represents a delivery person, including:

- id: Unique identifier for the delivery person.
- deliveries_made: Count of deliveries completed by the delivery person.
- mutex: Mutex for synchronizing access to this structure's data.



cook_function Explanation

Extract Cook ID:

- The function starts by extracting the cook's ID from the argument passed to it.
- **Order Processing Loop**:
- The cook enters an infinite loop where it processes orders from the orders_queue.
- **Check for Orders and Cancellation**:
- The cook locks the queue_mutex to safely access the orders_queue.
- If the queue is empty and all orders have been processed, the cook unlocks the mutex and exits the loop, ending the thread.
- If the queue is empty or a cancellation signal (cancel_all_orders) is received, the cook unlocks the mutex and breaks out of the loop.
- **Dequeue an Order**:
- The cook retrieves an order from the front of the orders_queue and unlocks the mutex.
- **Check for Cancellation**:
- If a cancellation signal is received, the cook breaks out of the loop.
- **Simulate Order Preparation**:
- The cook logs that it is preparing the order and simulates preparation time with usleep.
- **Simulate Cooking Process**:
- The cook waits for an oven apparatus and capacity by acquiring the corresponding semaphores.
- The cook logs that it is cooking the order and simulates cooking time with usleep.
- After cooking, the cook releases the semaphores.
- **Enqueue Cooked Order**:
- The cook locks the cooked_mutex to safely access the cooked_queue.
- The cooked order is enqueued into the cooked_queue, and the cook signals the condition variable cooked_not_empty.
- The cook unlocks the cooked_mutex.
- **Log Order Completion**:
- The cook logs that the order has been finished.
- **Update Processed Orders Count**:
- The cook locks the queue_mutex to safely update the count of processed orders.

- If all orders have been processed, the cook sets the all_orders_cooked flag, broadcasts the condition variable cooked_not_empty, and exits the loop.
- The cook unlocks the mutex.
- **Return from the Function**:
- The function returns NULL, ending the cook's thread.

delivery_function Explanation

Extract Delivery Person Information:

- The function starts by extracting the delivery person's information from the argument passed to it.
- **Order Delivery Loop**:
- The delivery person enters an infinite loop where they check for cooked orders to deliver.
- **Check for Cooked Orders and Cancellation**:
- The delivery person locks the cooked_mutex to safely access the cooked_queue.
- While the cooked_queue is empty and not all orders are cooked or canceled, the delivery person waits on the cooked_not_empty condition variable.
- If the queue is empty, all orders are cooked, or a cancellation signal (cancel_all_orders) is received, the delivery person unlocks the mutex and breaks out of the loop.
- **Dequeue a Cooked Order**:
- If there are cooked orders, the delivery person retrieves an order from the front of the cooked_queue and unlocks the mutex.
- **Check for Cancellation**:
- If a cancellation signal is received, the delivery person breaks out of the loop.
- **Simulate Order Delivery**:
- The delivery person logs that they are delivering the order and calculates the delivery time based on the order's coordinates.
- The delivery person simulates the delivery time with sleep.
- **Log Delivery Completion**:
- The delivery person logs that the order has been delivered and increments the count of deliveries made.
- **Update Delivered Orders Count**:
- The delivery person locks the delivery_mutex to safely update the count of delivered orders.
- If all orders have been delivered, the delivery person unlocks the mutex and breaks out of the loop.

- The delivery person unlocks the mutex.
- **Return from the Function**:
- The function returns NULL, ending the delivery person's thread.

Main :

Argument Validation:

- The function begins by validating the command-line arguments. It expects exactly four arguments: <port>, <cook_count>, <delivery_count>, and <delivery_speed>. If the arguments are incorrect, it prints a usage message and exits.
- **Initialization**:
- The function initializes the server configuration based on the provided arguments:
    - port: Port number for the server to listen on.
    - num_cooks: Number of cook threads to create.
    - num_deliveries: Number of delivery person threads to create.
    - delivery_speed: Speed at which deliveries are made.
- **Open Log File**:
- Opens a log file (pide_shop_log.txt) for writing logs. If the file cannot be opened, it prints an error message and exits.
- **Setup Server Socket**:
- Creates a server socket using socket and configures it to reuse the address using setsockopt.
- Binds the socket to the specified port and starts listening for incoming connections with a backlog of 10.
- **Signal Handling Setup**:
- Sets up a signal handler for SIGINT (e.g., Ctrl+C) to ensure proper cleanup and logging when the server is interrupted.
- **Main Server Loop**:
- Enters an infinite loop to handle client connections.
- **Accept Client Connection**:
- Accepts an incoming client connection. If the connection fails, it prints an error message and continues to the next iteration of the loop.
- **Receive Client PID**:
- Receives the client's process ID (PID) and prints a message indicating a new customer connection.

- **Receive Number of Orders**:
- Receives the number of orders from the client.
- **Receive Orders**:
- Allocates memory for the orders and receives them from the client.
- **Initialize Queues**:
- Initializes the orders_queue and cooked_queue to manage incoming and cooked orders.
- **Enqueue Orders**:
- Locks the queue_mutex, enqueues all received orders into the orders_queue, and then unlocks the mutex.
- **Log Order Reception**:
- Logs the number of received and queued orders.
- **Initialize Semaphores**:
- Initializes semaphores for managing oven apparatus and capacity.
- **Create Cook Threads**:
- Creates and starts the specified number of cook threads.
- **Create Delivery Threads**:
- Creates and starts the specified number of delivery person threads.
- **Create Cancel Listener Thread**:
- Creates and starts a thread to listen for cancellation messages from the client.
- **Wait for Cook and Delivery Threads**:
- Waits for all cook and delivery person threads to finish using pthread_join.
- **Cancel Listener Thread if Necessary**:
- If all cook and delivery threads are done, cancels the listener thread.
- **Wait for Cancel Listener Thread**:
- Waits for the cancel listener thread to finish.
- **Cleanup Resources**:
- Calls cleanup_resources to release all allocated resources.
- **Log Completion**:
- Logs whether the client orders were processed and delivered or canceled.
- **Send Response to Client**:
- Sends a response message to the client indicating the result of order processing.
- **Close Client Socket**:

- Closes the client socket.
- **Reset Global Variables**:
- Resets global variables for the next client connection.
- **Close Log File and Server Socket**:
- Closes the log file and the server socket before exiting.

Other Important Functions

```c
void cleanup_resources() {
    sem_destroy(&oven_aparatus);
    sem_destroy(&oven_capacity);

    queue_free(&orders_queue);
    queue_free(&cooked_queue);

    if (delivery_persons) {
        for (int i = 0; i < num_deliveries; i++) {
            pthread_mutex_destroy(&delivery_persons[i].mutex);
        }
        free(delivery_persons);
        delivery_persons = NULL;
    }
}

void sigint_handler(int sig) {
    printf(" .. Upps quiting.. writing log file\n");
    log_file_fd = open("pide_shop_log.txt", O_WRONLY | O_CREAT | O_APPEND, 0644);
    char buffer[BUFFER_SIZE];
    sprintf(buffer, "> Server is shut down\n");
    write(log_file_fd, buffer, strlen(buffer));
    pthread_mutex_destroy(&queue_mutex);
    pthread_mutex_destroy(&cooked_mutex);
    pthread_mutex_destroy(&delivery_mutex);
    pthread_cond_destroy(&queue_not_empty);
    pthread_cond_destroy(&cooked_not_empty);

    close(server_sock);
    cleanup_resources();
    close(log_file_fd);
    exit(0);
}

void setup_signal_handling() {
    struct sigaction sa;
    sa.sa_handler = sigint_handler;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;  // No flags
    if (sigaction(SIGINT, &sa, NULL) == -1) {
        perror("Error setting up signal handler");
        exit(1);
    }
}
```

Mutexes , Condition Variables , Semaphores and Shared Variables That Are Used :

```
pthread_mutex_t queue_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t queue_not_empty = PTHREAD_COND_INITIALIZER;
pthread_mutex_t cooked_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cooked_not_empty = PTHREAD_COND_INITIALIZER;
pthread_mutex_t delivery_mutex = PTHREAD_MUTEX_INITIALIZER;

sem_t oven_aparatus;
sem_t oven_capacity;
int num_cooks, num_deliveries, delivery_speed;
int orders_processed = 0;
int orders_delivered = 0;
int all_orders_cooked = 0;
int cancel_all_orders = 0;
```

Test Results

```
ademmurat@ademmurat-GL553VD:~/Desktop/sistemfinal$ ./PideShop 8080 4 6 10

>PideShop active waiting for connection...
```

```
ademmurat@ademmurat-GL553VD:~/Desktop/sistemfinal$ ./HungryVeryMuch 192.168.215.156 8080 50 10 10
Client PID: 13279...
All orders sent successfully.
Server response: All customers served
ademmurat@ademmurat-GL553VD:~/Desktop/sistemfinal$
```

```
ademmurat@ademmurat-GL553VD:~/Desktop/sistemfinal$ ./PideShop 8080 4 6 10

>PideShop active waiting for connection...
>New customer connected... client # 13279
> 50 new customers... Serving
>done serving client 13279

>PideShop active waiting for connection...
```

```
ademmurat@ademmurat-GL553VD:~/Desktop/sistemfinal$ ./HungryVeryMuch 192.168.215.156 8080 20 20 20
Client PID: 13333...
All orders sent successfully.
Server response: All customers served
ademmurat@ademmurat-GL553VD:~/Desktop/sistemfinal$
```

```
ademmurat@ademmurat-GL553VD:~/Desktop/sistemfinal$ ./PideShop 8080 4 6 10

>PideShop active waiting for connection...
>New customer connected... client # 13279
> 50 new customers... Serving
>done serving client 13279

>PideShop active waiting for connection...
>New customer connected... client # 13333
> 20 new customers... Serving
>done serving client 13333

>PideShop active waiting for connection...
```

```
ademmurat@ademmurat-GL553VD:~/Desktop/sistemfinal$ ./HungryVeryMuch 192.168.215.156 8080 100 10 10
Client PID: 13427...
All orders sent successfully.
Server response: All customers served
ademmurat@ademmurat-GL553VD:~/Desktop/sistemfinal$
```


```
ademmurat@ademmurat-GL553VD:~/Desktop/sistemfinal$ ./PideShop 8080 4 6 10

>PideShop active waiting for connection...
>New customer connected... client # 13279
> 50 new customers... Serving
>done serving client 13279

>PideShop active waiting for connection...
>New customer connected... client # 13333
> 20 new customers... Serving
>done serving client 13333
```


```
>PideShop active waiting for connection...
>New customer connected... client # 13427
> 100 new customers... Serving
>done serving client 13427

>PideShop active waiting for connection...
```

When Orders Cancelled by Client

```
ademmurat@ademmurat-GL553VD:~/Desktop/sistemfinal$ ./HungryVeryMuch 192.168.215.156 8080 30 10 10
Client PID: 13567...
All orders sent successfully.
^C signal .. cancelling orders.. editing log..
ademmurat@ademmurat-GL553VD:~/Desktop/sistemfinal$
```

```
>PideShop active waiting for connection...
>New customer connected... client # 13567
> 30 new customers... Serving
>Client order cancellation received

>PideShop active waiting for connection...
```

When Server is Shut Down :

```
ademmurat@ademmurat-GL553VD:~/Desktop/sistemfinal$ ./PideShop 8080 4 6 10

>PideShop active waiting for connection...
>New customer connected... client # 13279
> 50 new customers... Serving
>done serving client 13279

>PideShop active waiting for connection...
>New customer connected... client # 13333
> 20 new customers... Serving
>done serving client 13333

>PideShop active waiting for connection...
>New customer connected... client # 13427
> 100 new customers... Serving
>done serving client 13427

>PideShop active waiting for connection...
>New customer connected... client # 13567
> 30 new customers... Serving
>Client order cancellation received

>PideShop active waiting for connection...
^C .. Upps quiting.. writing log file
ademmurat@ademmurat-GL553VD:~/Desktop/sistemfinal$
```