Report: Pide Shop Project

ÇAĞRI YILDIZ 1901042630

CSE_344_FINAL

Introduction

The Pide Shop project simulates a food production and delivery system using a multi-threaded server architecture. The system efficiently handles orders from clients, processes them through various stages, and ensures timely delivery by managing multiple threads and synchronization mechanisms.

System Design

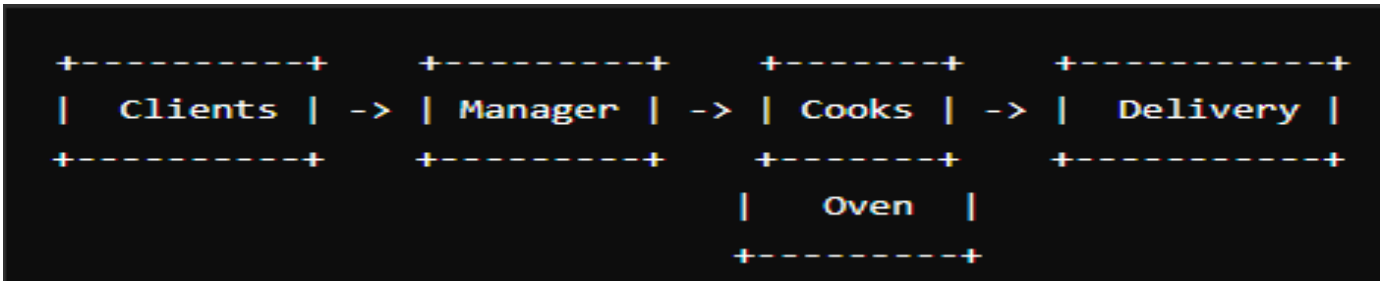
Components

1. **Manager**: Manages orders received via phone, assigns them to available cooks.
2. **Cooks**: Prepare the meal, simulate the preparation time by calculating the pseudo-inverse of a 30x40 complex matrix, then cook the meal for half the preparation time.
3. **Oven**: Holds up to 6 meals, has 2 openings for placing and removing meals.
4. **Delivery Personnel**: Deliver meals and return to the shop. Each delivery person can carry up to 3 meals.
5. **Clients**: Place orders to the server.

Design Decisions

1. **Multithreading**: Separate thread pools for cooks and delivery personnel to ensure efficient processing.
2. **Synchronization**: Mutexes and condition variables to manage shared resources.
3. **Signal Handling**: Graceful shutdown and order cancellation using signal handlers.
4. **Logging**: Maintain logs for tracking activities and debugging.

Diagram



Implementation Details

Manager

The manager receives orders from clients, assigns them to available cooks, and signals when orders are ready for delivery.

```c
void start_manager(void) {
    printf("Initializing manager...\n");
    pthread_t thread;
    if (pthread_create(&thread, NULL, manager_thread, NULL) != 0) {
        perror("Failed to create manager thread");
        exit(EXIT_FAILURE);
    }
    pthread_detach(thread);
    printf("Manager initialized...\n");
}
```

```c
void manager_receive_order() {
    pthread_mutex_lock(&manager_mutex);
    order_received++;
    pthread_cond_signal(&manager_cond);
    pthread_mutex_unlock(&manager_mutex);

    char message[256];
    snprintf(message, sizeof(message), "Manager received an order");
    log_message(message);
}
```

Cooks

Cooks prepare meals by computing the pseudo-inverse of a 30x40 complex matrix, then cook the meal for half the preparation time.

```c
// Thread function for each cook
void *cook_thread(void *arg) {
    Cook *cook = (Cook *)arg;

```

```c
// Function to compute the pseudo-inverse of a 30x40 matrix with complex elements
void compute_pseudo_inverse(void) {
    int m = 30, n = 40;
    double complex A[m][n];
    double complex B[n][m];
```

Oven

The oven holds up to 6 meals and has 2 openings for placing and removing meals.

```
16  typedef struct {
17      int num_aparatus; // Number of available oven aparatus
18      int num_meals;    // Number of meals currently in the oven
19      pthread_mutex_t mutex;
20      pthread_cond_t cond_aparatus;
21      pthread_cond_t cond_meals;
22  } Oven;
23
24  #define MAX_APARATUS 3
25  #define MAX_MEALS 6
```

Delivery Personnel

Delivery personnel receive prepared meals from the manager, deliver them based on customer addresses, and return to the shop.

```
85  void *delivery_thread(void *arg) {
86      DeliveryPerson *person = (DeliveryPerson *)arg;
87
```

```
120 /*
121  * Simulate the delivery process by sleeping for the calculated time based on distance
122  * Side effects:
123  * - Uses sleep to simulate delivery time.
124  */
125 void simulate_delivery(float x, float y, float velocity) {
126     float distance = sqrt(x * x + y * y);
127     int sleepTime = (int)(distance / velocity);
128     sleep(sleepTime);
129 }
```

Multithreaded Internet Server

The server implements thread pools for cooks and delivery personnel.

```
54  void start_server(const char *ip_address, int port) {
55      int socket_desc, client_sock, c;
56      struct sockaddr_in server, client;
```

Signal Handling

The server handles signals for order cancellations and shop closure.

```c
void signal_handler(int sig) {
    log_message("Signal received, shutting down...");
    cancel_all_orders();
    write_log_file();
    log_message("Log file written");
    exit(0);
}
```

Logging

Logs are created to record shop activities and order states.

```c
// Logging function
void log_message(const char *message) {
    time_t now;
    time(&now);
    char timestamp[20];
    strftime(timestamp, sizeof(timestamp), "%Y-%m-%d %H:%M:%S", localtime(&now));
    printf("[%s] %s\n", timestamp, message);
    FILE *log_file = fopen("pide_shop_log.txt", "a");
    if (log_file) {
        fprintf(log_file, "[%s] %s\n", timestamp, message);
        fclose(log_file);
    } else {
        perror("Failed to open log file");
    }
}
```

Server and Client Interaction

Server Command

The server is started with the following command:

```
# Run server with specified arguments
run_server: server
    ./server 127.0.0.1 6 6 1000
```

Client Command

Clients generate orders with the following command:

```
# Run client with specified arguments
run_client: client
    ./client 127.0.0.1 600 6 8
```

Additional Details

Customer Location

Customers are located relative to the mayor's office south entrance (0,0).

Order Cancellation

Orders can be canceled at any stage of preparation, cooking, or delivery.

```
 98  void cancel_order() {
 99      pthread_mutex_lock(&manager_mutex);
100      order_received = 0;
101      order_ready = 0;
102      pthread_cond_broadcast(&manager_cond);
103      pthread_mutex_unlock(&manager_mutex);
104
105      log_message("All orders cancelled");
106  }
```

Server Shutdown

The server handles ^C and ^D signals for orderly shutdown.

```
226      signal(SIGINT, signal_handler);
227      signal(SIGTERM, signal_handler);
228      signal(SIGPIPE, SIG_IGN); // Ignore SIGPIPE signal to prevent crashes on broken pipe
229
```

Implementation Steps

Step-by-Step Execution

1. Server Initialization:
   - The server initializes by parsing command line arguments to get the IP address, port number, cook thread pool size, delivery thread pool size, and delivery speed.
   - Signal handlers are set up to ensure graceful shutdown.

2. Starting Thread Pools:
   - The server starts the thread pools for cooks and delivery personnel.

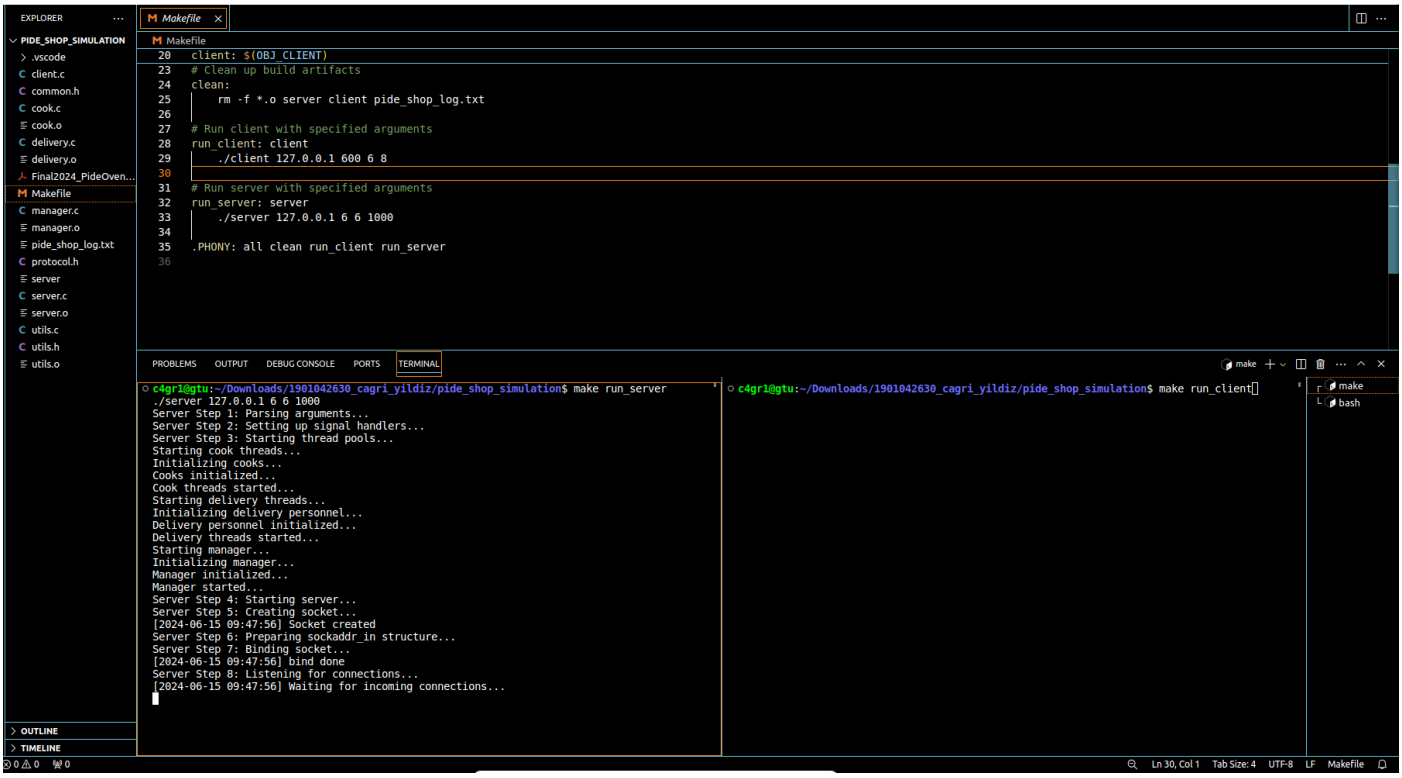
3. Accepting Client Connections:
   - The server listens for incoming client connections and creates a new thread to handle each client.

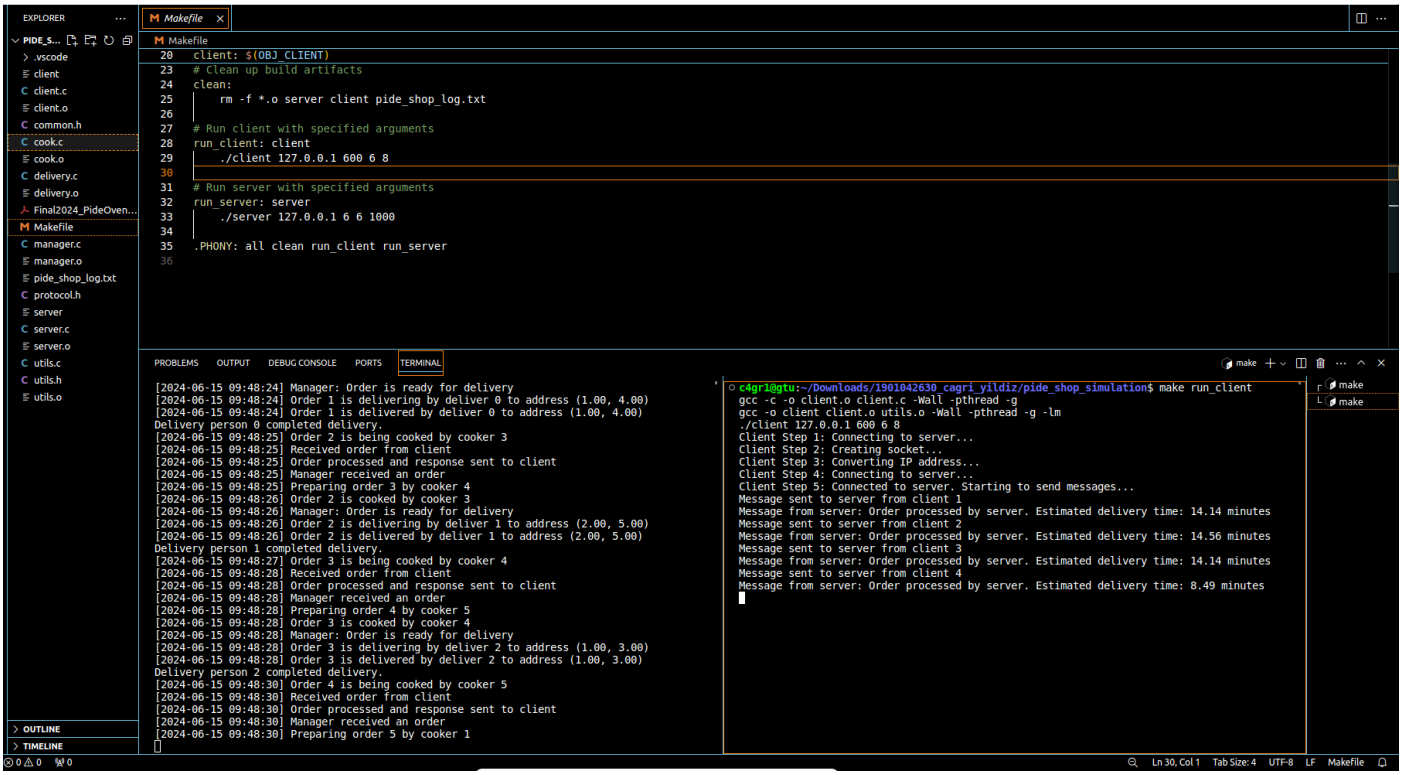
4. Client Handler:
    - Each client handler thread receives orders from the client, processes them, and sends responses back to the client.

5. Order Processing by Cooks:
    - Cooks process the orders by simulating meal preparation and cooking.

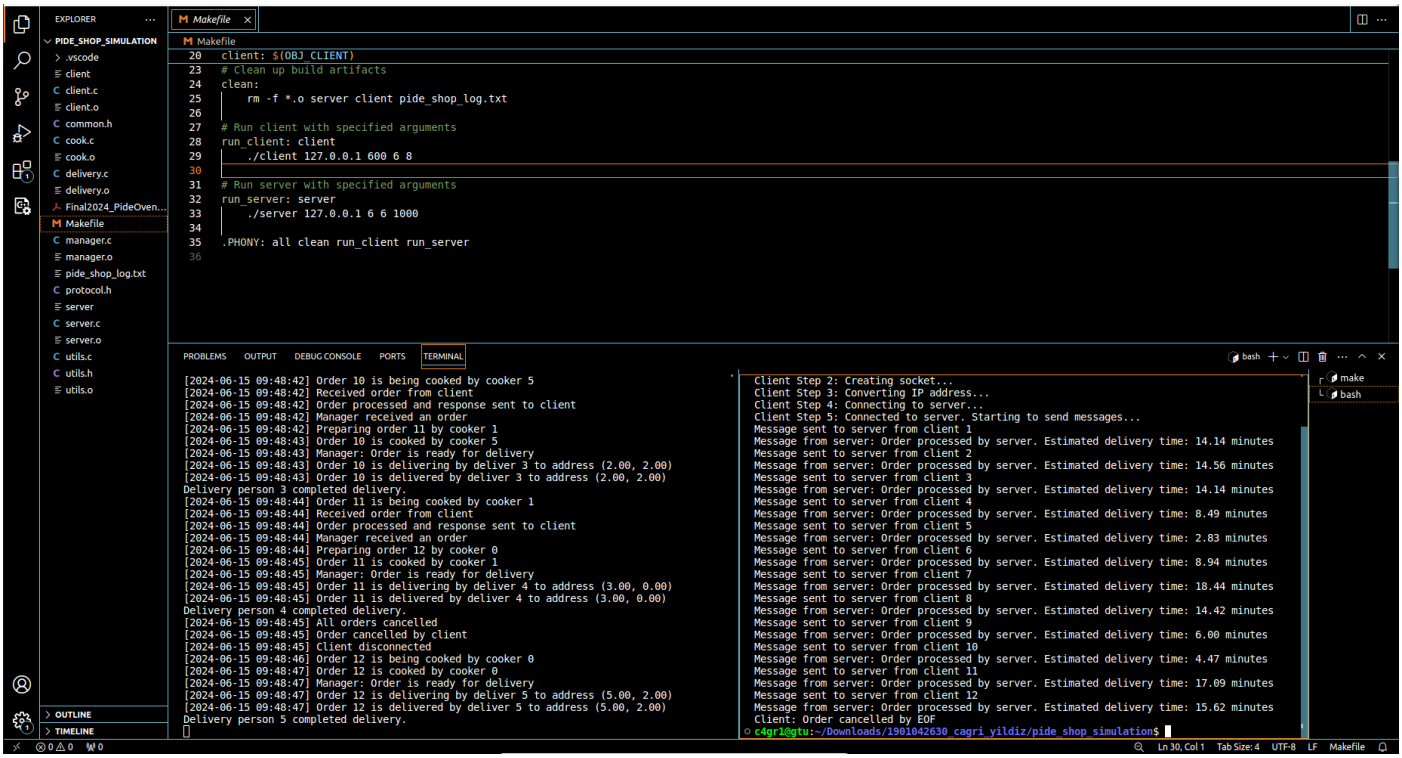
6. Meal Delivery:
    - Delivery personnel deliver the meals to the customers and return to the shop.

Screenshots

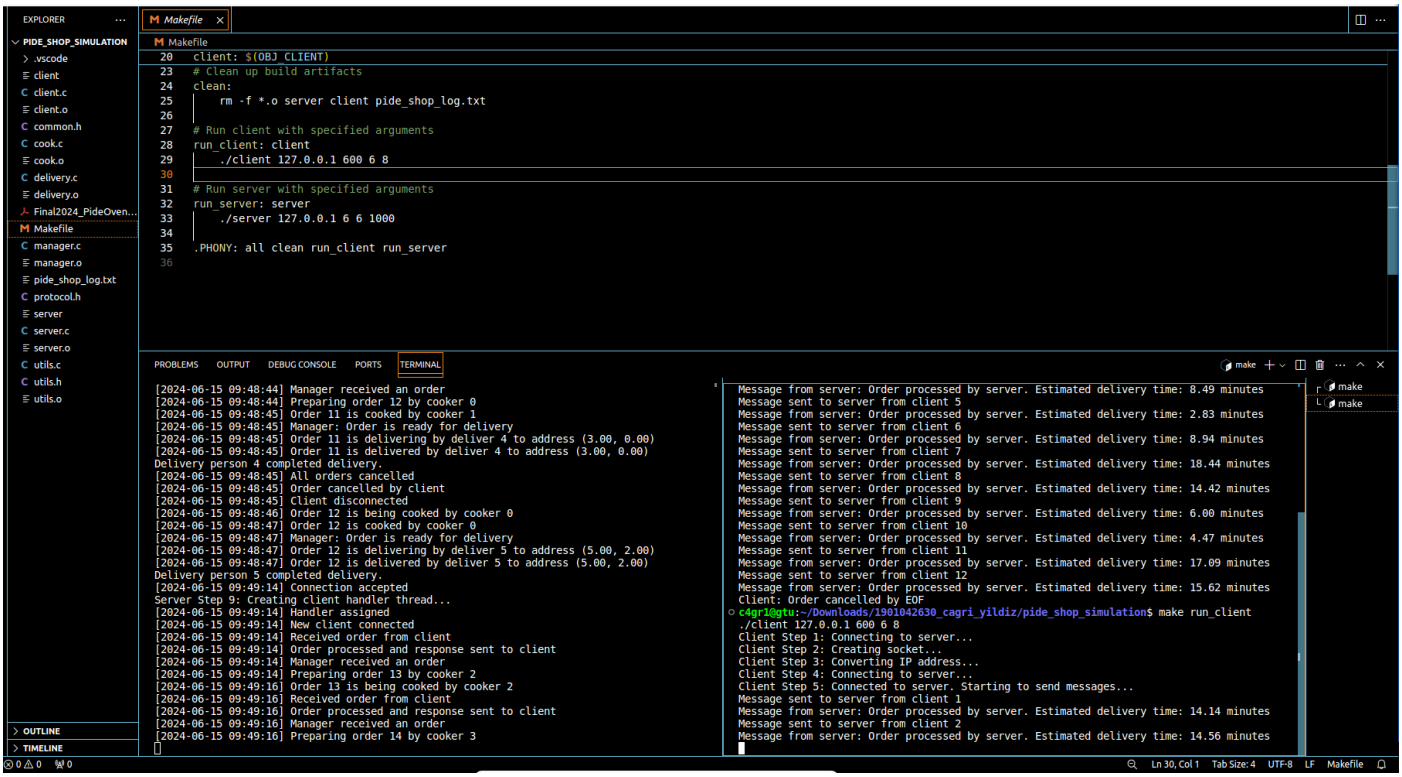




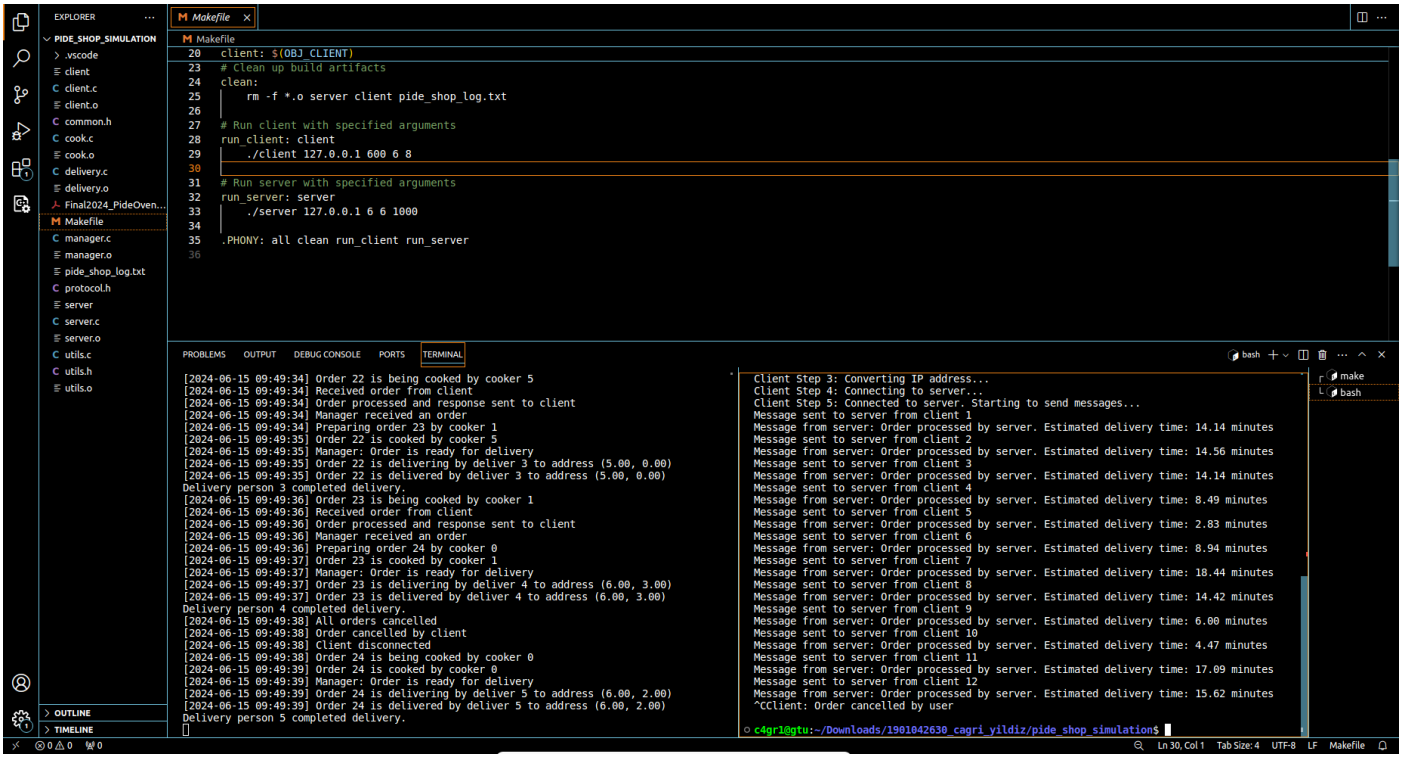
After ctrl+d in client side



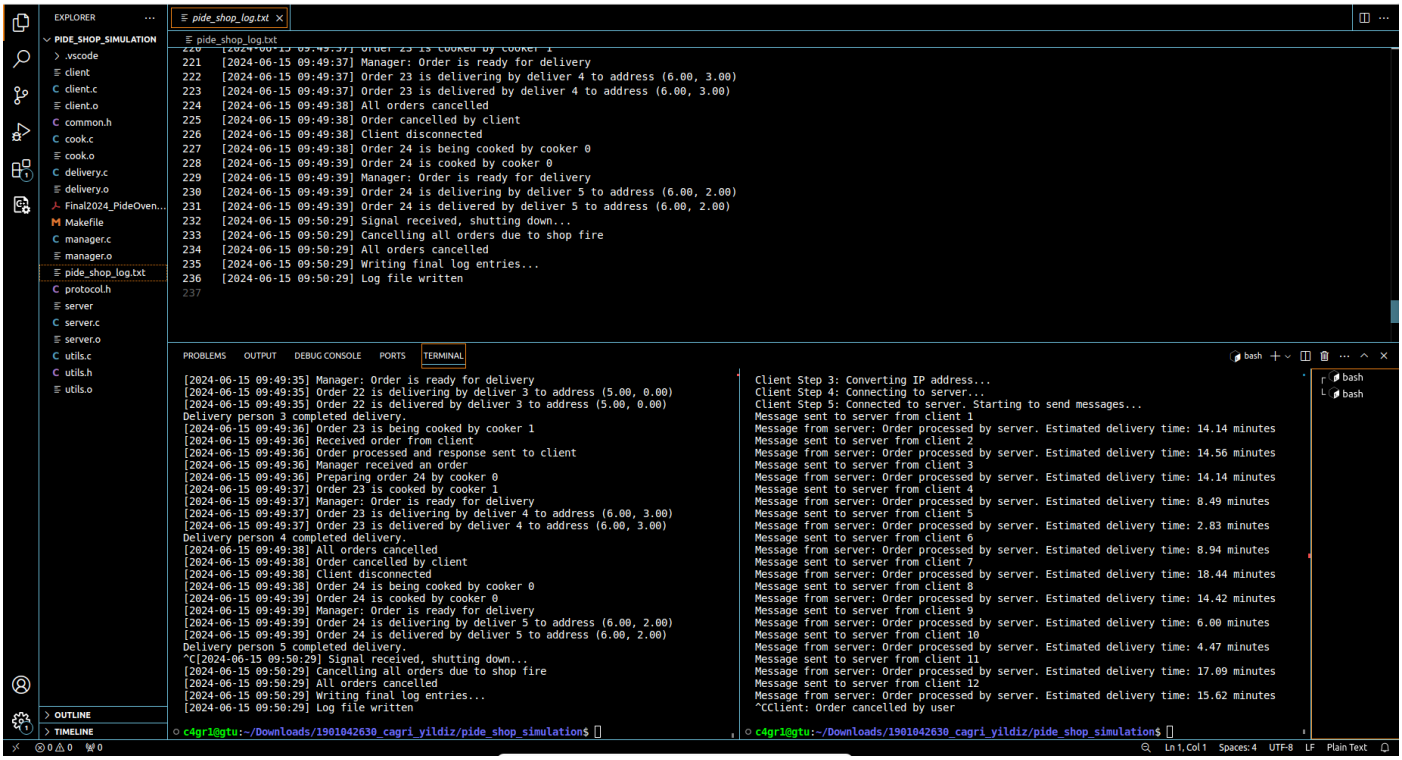
Adding new orders



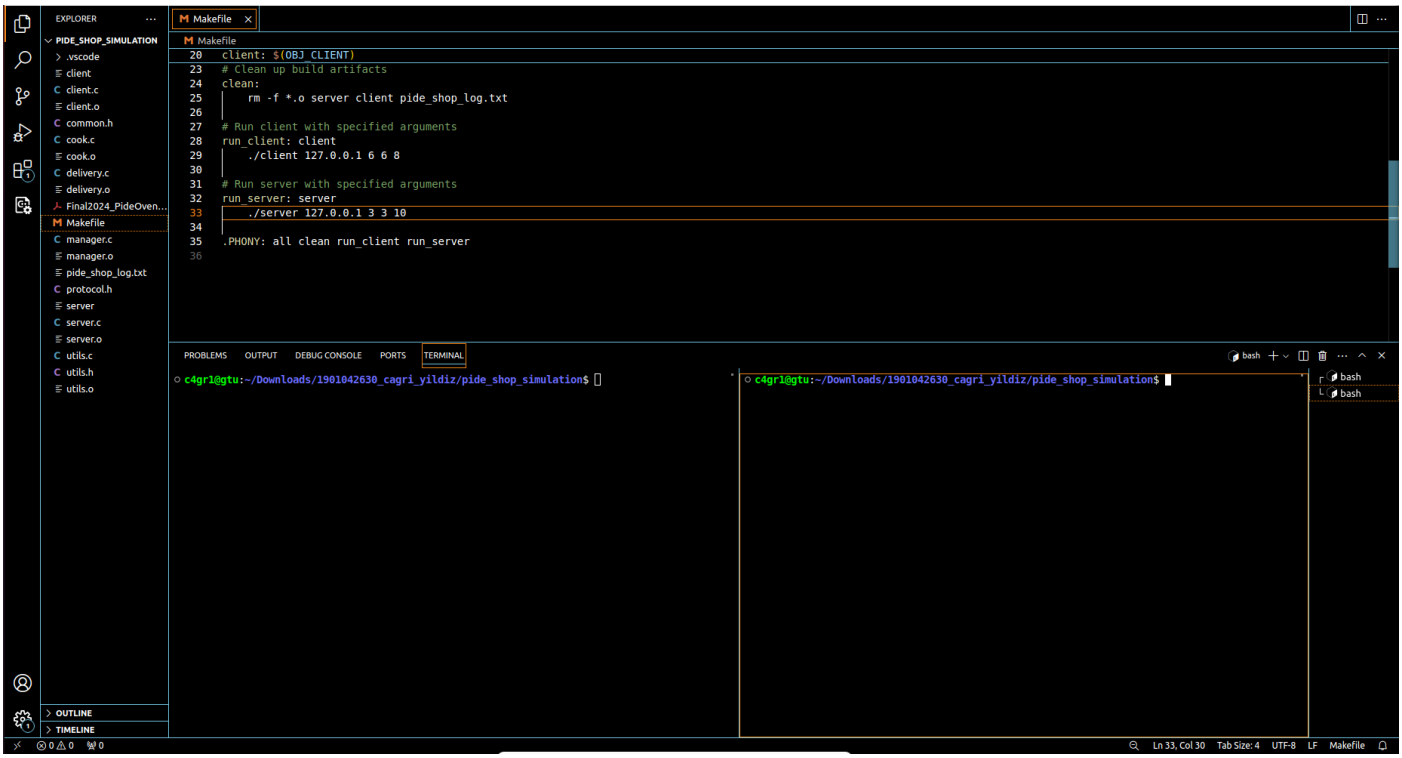
After ctrl+c in client side

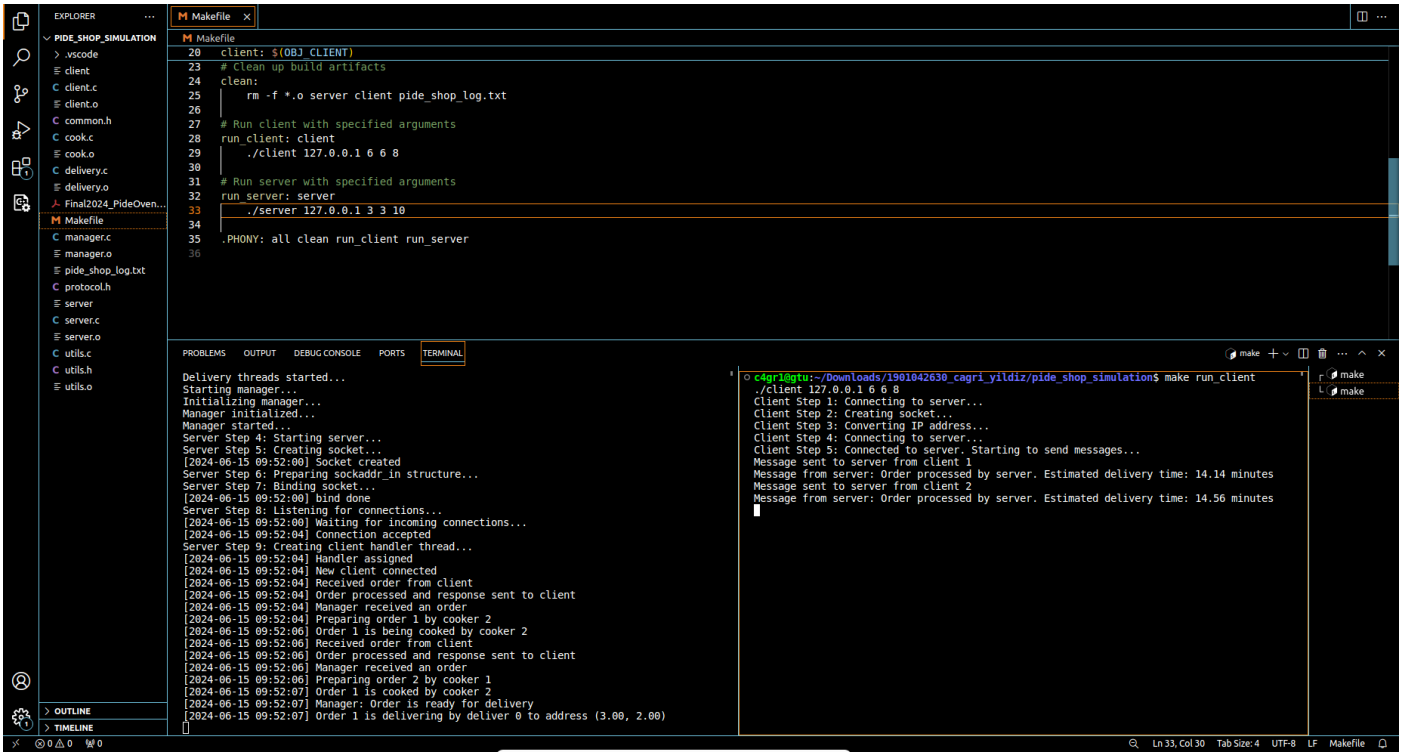


After ctrl + c in server side and log.txt

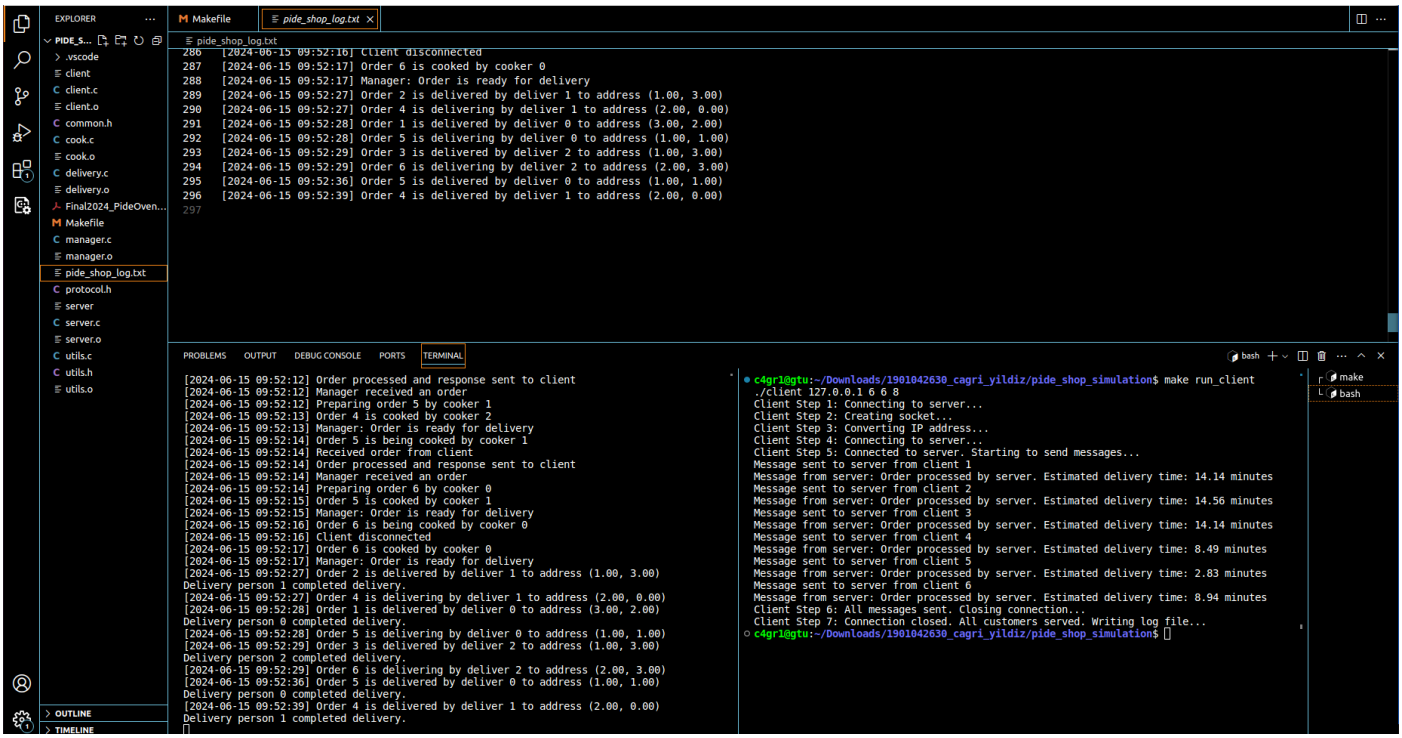


I update the makefile with lower values

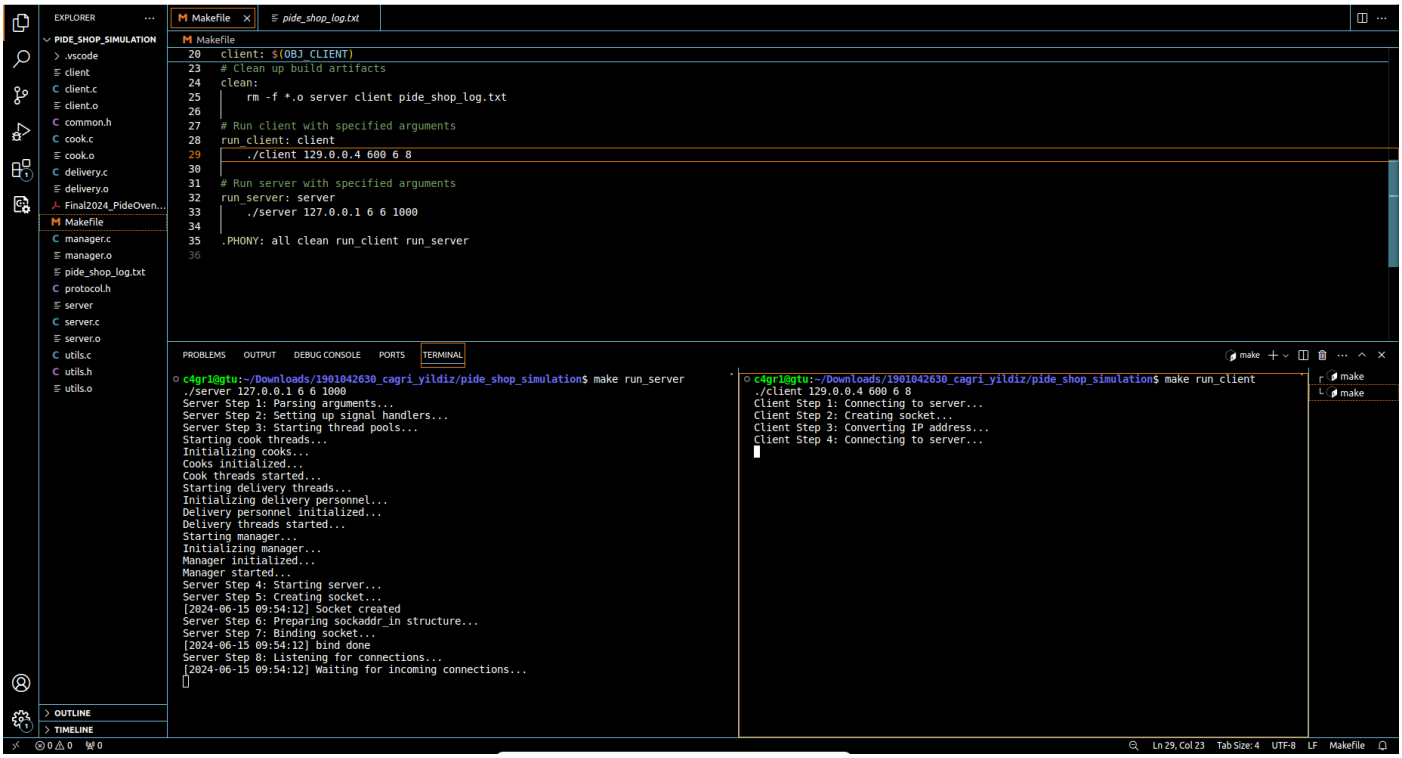




After the all process.



If the client IP is entered incorrectly (can't connect)



Another test case