Gebze Technical University
Department of Computer Engineering
CSE344 - Spring 2024
Final Project Report
AHMET ALPER UZUNTEPE
1901042669

```
Order for client 26 is get order into aparatus
Order 26 at (4, 5): Ready for delivery by thread 0 at Sat Jun 15 03:07:11 2024
Order 27 at (3, 9): Preparing by thread 0 at Sat Jun 15 03:07:11 2024
Order for client 27 is get order into aparatus
Order 27 at (3, 9): Cooking by thread 0 at Sat Jun 15 03:07:11 2024
Order for client 27 is get order into aparatus
Order 27 at (3, 9): Ready for delivery by thread 0 at Sat Jun 15 03:07:11 2024
Order 28 at (8, 7): Preparing by thread 0 at Sat Jun 15 03:07:11 2024
Order for client 28 is get order into aparatus
Order 28 at (8, 7): Cooking by thread 0 at Sat Jun 15 03:07:11 2024
Order for client 28 is get order into aparatus
Order 28 at (8, 7): Ready for delivery by thread 0 at Sat Jun 15 03:07:11 2024
Order 29 at (5, 7): Preparing by thread 0 at Sat Jun 15 03:07:11 2024
Order 30 at (5, 2): Preparing by thread 1 at Sat Jun 15 03:07:11 2024
Order for client 30 is get order into aparatus
Order 30 at (5, 2): Cooking by thread 1 at Sat Jun 15 03:07:11 2024
Order for client 30 is get order into aparatus
Order 30 at (5, 2): Ready for delivery by thread 1 at Sat Jun 15 03:07:11 2024
Order for client 29 is get order into aparatus
Order 29 at (5, 7): Cooking by thread 0 at Sat Jun 15 03:07:11 2024
Order for client 29 is get order into aparatus
Order 29 at (5, 7): Ready for delivery by thread 0 at Sat Jun 15 03:07:11 2024
Order 10 at (8, 2): Delivered by thread 3 at Sat Jun 15 03:07:08 2024
Order 5 at (2, 4): Delivered by thread 1 at Sat Jun 15 03:07:07 2024
Order 13 at (3, 6): Delivered by thread 5 at Sat Jun 15 03:07:09 2024
Order 8 at (2, 6): Delivered by thread 2 at Sat Jun 15 03:07:08 2024
Order 2 at (4, 7): Delivered by thread 0 at Sat Jun 15 03:07:07 2024
Order 11 at (6, 2): Delivered by thread 3 at Sat Jun 15 03:07:08 2024
Order 16 at (8, 4): Delivered by thread 4 at Sat Jun 15 03:07:09 2024
Order 3 at (3, 0): Delivered by thread 0 at Sat Jun 15 03:07:07 2024
Order 19 at (4, 2): Out for delivery by thread 0 at Sat Jun 15 03:07:10 2024
Order 20 at (2, 8): Out for delivery by thread 0 at Sat Jun 15 03:07:10 2024
Order 21 at (8, 5): Out for delivery by thread 0 at Sat Jun 15 03:07:10 2024
Order 17 at (2, 2): Delivered by thread 4 at Sat Jun 15 03:07:09 2024
Order 6 at (0, 5): Delivered by thread 1 at Sat Jun 15 03:07:07 2024
Order 22 at (6, 4): Out for delivery by thread 1 at Sat Jun 15 03:07:10 2024
Order 24 at (3, 9): Out for delivery by thread 1 at Sat Jun 15 03:07:10 2024
Order 23 at (0, 8): Out for delivery by thread 1 at Sat Jun 15 03:07:10 2024
Order 19 at (4, 2): Delivered by thread 0 at Sat Jun 15 03:07:10 2024
Order 12 at (5, 1): Delivered by thread 3 at Sat Jun 15 03:07:08 2024
Order 25 at (6, 1): Out for delivery by thread 3 at Sat Jun 15 03:07:11 2024
Order 26 at (4, 5): Out for delivery by thread 3 at Sat Jun 15 03:07:11 2024
Order 27 at (3, 9): Out for delivery by thread 3 at Sat Jun 15 03:07:11 2024
Order 18 at (1, 5): Delivered by thread 4 at Sat Jun 15 03:07:09 2024
Order 28 at (8, 7): Out for delivery by thread 4 at Sat Jun 15 03:07:11 2024
Order 30 at (5, 2): Out for delivery by thread 4 at Sat Jun 15 03:07:11 2024
Order 29 at (5, 7): Out for delivery by thread 4 at Sat Jun 15 03:07:11 2024
Order 22 at (6, 4): Delivered by thread 1 at Sat Jun 15 03:07:10 2024
Order 9 at (9, 6): Delivered by thread 2 at Sat Jun 15 03:07:08 2024
Order 14 at (9, 8): Delivered by thread 5 at Sat Jun 15 03:07:09 2024
Order 25 at (6, 1): Delivered by thread 3 at Sat Jun 15 03:07:11 2024
Order 20 at (2, 8): Delivered by thread 0 at Sat Jun 15 03:07:10 2024
Order 26 at (4, 5): Delivered by thread 3 at Sat Jun 15 03:07:11 2024
Order 28 at (8, 7): Delivered by thread 4 at Sat Jun 15 03:07:11 2024
Order 24 at (3, 9): Delivered by thread 1 at Sat Jun 15 03:07:10 2024
Order 15 at (4, 9): Delivered by thread 5 at Sat Jun 15 03:07:09 2024
Order 21 at (8, 5): Delivered by thread 0 at Sat Jun 15 03:07:10 2024
Order 30 at (5, 2): Delivered by thread 4 at Sat Jun 15 03:07:11 2024
Order 23 at (0, 8): Delivered by thread 1 at Sat Jun 15 03:07:10 2024
Order 27 at (3, 9): Delivered by thread 3 at Sat Jun 15 03:07:11 2024
Order 29 at (5, 7): Delivered by thread 4 at Sat Jun 15 03:07:11 2024
```

Overview

This report details the implementation and functionality of a multithreaded food production and delivery system in C. The system simulates a food delivery service where multiple clients can place orders, which are then processed by a team of cooks and delivered by delivery personnel. The server leverages POSIX threads (pthreads) to manage concurrent order processing, ensuring efficient handling of multiple client requests.

The system's architecture consists of a server that handles incoming client connections and manages the order processing pipeline. Clients can connect to the server to place orders, which are then managed by separate threads for cooks and delivery personnel. This setup ensures that the system can handle multiple orders simultaneously, providing a realistic simulation of a busy food delivery service

Overview 1
Initialization 2
Server 3
Client 14
Conclusion 17
OUTPUTS 19
MAKEFILE : 30
NOTE!! 31

Initialization

The program starts by initializing the necessary data structures and variables, including setting up the server socket, binding it to an IP address and port, and preparing the environment for handling multiple client connections. The main steps are:

- **Command-line Argument Parsing**
  - The program requires the IP address and port number to be specified as command-line arguments. This allows the server to know where to bind the socket for incoming connections.
  - If the correct number of arguments is not provided, the program prints a usage message and exits.
- **Socket Creation**
  - A socket is created using the socket() function with parameters specifying the use of the IPv4 protocol (AF_INET) and TCP (SOCK_STREAM). This socket will be used to listen for incoming client connections.
- **Setting Socket Options**
  - The setsockopt() function is used to set options on the socket, such as allowing the address to be reused (SO_REUSEADDR). This ensures that the server can be restarted without waiting for the socket to be released by the operating system.
- **Server Address Setup**
  - A sockaddr_in structure is initialized to hold the server's IP address and port number. The structure is zeroed out using memset() to ensure there are no leftover values.
  - The sin_family field is set to AF_INET to indicate the use of the IPv4 protocol.
  - The inet_pton() function converts the IP address from text to binary form and stores it in the sinaddr() field of the structure.
  - The port number is converted to network byte order using htons() and stored in the sin_port field.
- **Binding the Socket**
  - The bind() function assigns the address specified in the sockaddr_in structure to the socket. This allows the socket to listen for incoming connections on the specified IP address and port.
- **Listening for Connections**
  - The listen()function is called to put the server socket in a passive mode, allowing it to accept incoming connection requests. The backlog parameter specifies the maximum number of pending connections that can be queued.
- **Accepting and Handling Client Connections**
  - The server enters an infinite loop, continuously accepting new client connections using the accept function.
  - For each new connection, a new thread is created using pthread_create(), which runs the manager function to handle the client's requests independently.

Server

1. **Socket Initialization and Binding**: The server initializes a socket, sets options, binds it to a specified IP address and port, and listens for incoming connections.
2. **Client Handling**: The server accepts new client connections and delegates the handling of each client to a separate thread.
3. **Order Management**: Orders are managed using a queue system, with separate queues for preparation and delivery.
4. **Cook and Delivery Routines**: Dedicated threads for cooks and delivery personnel handle the preparation and delivery of orders.

Server Functions:

1. **Main Function**
   - **Initialization**: Initializes server socket and sets options for reuse.
   - **Binding and Listening**: Binds the socket to an IP address and port, then listens for incoming connections.
   - **Client Acceptance**: Accepts new client connections and creates a new thread for each client.
2. **Manager Function**
   - **Order Management**: Handles incoming orders, adds them to the preparation queue, and signals available cooks.
3. **Cook Routine**
   - **Order Preparation**: Waits for orders in the preparation queue, updates the order status, and moves completed orders to the delivery queue.
4. **Delivery Routine**
   - **Order Delivery**: Waits for orders in the delivery queue, updates the order status, and manages the delivery process.
5. **Queue Management Functions**
   - **Enqueue/Dequeue Functions**: Manage the addition and removal of orders from the preparation and delivery queues.
6. **Synchronization Primitives**

Synchronization mechanisms are crucial for ensuring that multiple threads can safely access shared resources without causing data corruption or inconsistencies.

1. **Mutex**
   - Ensures only one thread accesses the buffer at a time.
   - Mutexes (pthread_mutex_t) are used to lock shared resources (such as order queues) before accessing them. This prevents race conditions by ensuring that only one thread can modify the resource at a time.
2. **Condition Variables**
   - Used to block threads when the buffer is access empty and wake them up when the state changes.
   - Condition variables (pthread_cond_t) are used to signal changes in the state of shared resources.

Global Variables and Structures

```c
// Global variables
int port; //server port
int cook_pool_size ;//number of cooks, and number of delivery persons
int delivery_pool_size; //number of delivery persons
Cook *cooks; //array of cooks
DeliveryPerson *delivery_persons; //array of delivery persons
Order orders[MAX_ORDERS]; //array of all orders
Order *prep_queue[MAX_ORDERS]; //queue for waiting for prepared
Order *cook_queue[MAX_ORDERS]; //queue for waiting for cooked
Order *delivery_queue[MAX_ORDERS]; //queue for waiting for delivered
int order_count = 0; //total number of orders
int prep_queue_start = 0, prep_queue_end = 0; //indices for the preparation queue
int cook_queue_start = 0, cook_queue_end = 0; //indices for the cooking queue
int delivery_queue_start = 0, delivery_queue_end = 0; //indices for the delivery queue
int delivered_count = 0; //count of delivered orders

pthread_mutex_t order_mutex = PTHREAD_MUTEX_INITIALIZER; //mutex to protect order-related operations
pthread_cond_t order_cond = PTHREAD_COND_INITIALIZER; //condition variable for order processing
pthread_cond_t delivery_cond = PTHREAD_COND_INITIALIZER; //condition variable for delivery processing
sem_t oven_sem; //semaphore to manage the oven capacity

FILE *log_file; //log file to record order status

// Function prototypes
void *cook_routine(void *arg);
void *delivery_routine(void *arg);
void manager(int socket);
void log_order_status(Order *order, int status, int thread_id);
int calculate_delivery_time(int x, int y, int speed);
void signal_handler(int signal);
void enqueue_preparation(Order *order);
Order *dequeue_preparation();
void enqueue_cooking(Order *order);
Order *dequeue_cooking();
void enqueue_delivery(Order *order);
Order *dequeue_delivery();
void simulate_computation_delay_prep();
void simulate_computation_delay_cook();
void notify_clients_all_orders_completed();
void cancel_order(Order *order);
void print_most_efficient_workers();
```

- MAX_ORDERS and MAX_CLIENTS define the maximum number of orders and clients the server can handle.
- The Order structure stores details of an order, including its ID, associated client socket, and status.
- The Cook and DeliveryPerson structures store details of cooks and delivery personnel, including their Ids and thread Ids.
- preparation_queue and delivery_queue are arrays used to store orders awaiting preparation and delivery, respectively.
- prep_front, prep_rear, delivery_front, and delivery_rear are indices used for managing the queues.
- order_mutex is a mutex for synchronizing access to the order queues.
- order_cond is a condition variable used to signal when there are new orders to process.

Signal Handler :

```c
// Signal handler for graceful shutdown
void signal_handler(int signal) {
    if (signal == SIGINT) {
        pthread_mutex_lock(&order_mutex);
        printf("\nRIP PIDE SHOP...\n");
        for (int i = 0; i < order_count; i++) {
            if (orders[i].status != 5) {
                orders[i].status = 6;
                log_order_status(&orders[i], 6, -1);
            }
        }
        print_most_efficient_workers();
        pthread_mutex_unlock(&order_mutex);
        fclose(log_file);
        exit(0);
    }
}
```

- The function checks if the received signal is SIGINT. If it is, the following steps are executed.
  1. Lock the mutex
  2. Print shutdown message
  3. Update order statuses
  4. Print most efficient workers
  5. Unlock the mutex
  6. Close the log file
  7. Exit the program

Main :

```c
char *ip_address = argv[1]; //IP address
port = atoi(argv[2]); //port number
cook_pool_size = atoi(argv[3]); //number of cooks
delivery_pool_size = atoi(argv[4]); //number of delivery persons
int delivery_speed = atoi(argv[5]); //speed of delivery
```

- Parse the command line arguments .

```c
signal(SIGINT, signal_handler); //register signal handler for SIGINT
signal(SIGQUIT, signal_handler); //register signal handler for SIGQUIT
```

- The program can handle SIGINT and SIGQUIT signals in a controlled manner, ensuring that important cleanup operations are performed before the program exits.

```c
//create cook threads
for (int i = 0; i < cook_pool_size; i++) {
    cooks[i].id = i;
    pthread_create(&cooks[i].thread, NULL, cook_routine, &cooks[i]);
}

//create delivery person threads
for (int i = 0; i < delivery_pool_size; i++) {
    delivery_persons[i].id = i;
    delivery_persons[i].speed = delivery_speed;
    delivery_persons[i].bag_count = 0;
    pthread_create(&delivery_persons[i].thread, NULL, delivery_routine, &delivery_persons[i]);
}
```

- Creating Thread Pool.

```c
int server_socket, client_socket;
struct sockaddr_in server_addr, client_addr;
socklen_t addr_len;

server_socket = socket(AF_INET, SOCK_STREAM, 0); //create server socket
if (server_socket < 0) {
    perror("Failed to create socket");
    return 1;
}

int opt = 1;
if (setsockopt(server_socket, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt)) < 0) {
    perror("Failed to set SO_REUSEADDR");
    return 1;
}

memset(&server_addr, 0, sizeof(server_addr));
server_addr.sin_family = AF_INET;//IP adres for local network(dynamik) and static
if (inet_pton(AF_INET, ip_address, &server_addr.sin_addr) <= 0) {
    perror("Invalid IP address");
    return 1;
}
server_addr.sin_port = htons(port);

if (bind(server_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
    perror("Failed to bind socket");
    return 1;
}

listen(server_socket, 5); //start listening for incoming connections
printf("Pide Shop server listening on %s IP Address and %d Port\n", ip_address, port);

while (1) {
    addr_len = sizeof(client_addr);
    client_socket = accept(server_socket, (struct sockaddr *)&client_addr, &addr_len); //accept a new client
    if (client_socket < 0) {
        perror("Failed to accept connection");
        continue;
    }

    printf("New customer connected\n");
    manager(client_socket); //handle the new customer
}

return 0;
```

1. Creating the Server Socket
2. Setting Socket Options
3. Server Address Setup
4. Binding the Socket
5. Listening for Incoming Connections
6. Accepting Client Connections

Cook Routine

```c
//routine for cooks to prepare and cook orders
void *cook_routine(void *arg) {
    Cook *cook = (Cook *)arg;

    while (1) {
        pthread_mutex_lock(&order_mutex);

        Order *order = dequeue_preparation(); //get the next order to prepare
        while (order == NULL) {
            pthread_cond_wait(&order_cond, &order_mutex);
            order = dequeue_preparation();
        }

        log_order_status(order, 1, cook->id); //log that the order is being prepared
        order->status = 1;
        if (send(order->client_socket, &order->status, sizeof(int), 0) == -1) {
            printf("%d th order canceled.\n", order->order_id);
            order->canceled_flag = 1;
            cancel_order(order);
            pthread_mutex_unlock(&order_mutex);
            continue;
        }
        pthread_mutex_unlock(&order_mutex);
        simulate_computation_delay_prep(); //simulate preparation time
        sem_wait(&oven_sem); //wait for an oven to become available
        pthread_mutex_lock(&order_mutex);
        log_order_status(order, 2, cook->id); //log that the order is being cooked
        order->status = 2;
        if (send(order->client_socket, &order->status, sizeof(int), 0) == -1) {
            if (order->canceled_flag == 0) {
                printf("%d th order canceled.\n", order->order_id);
                order->canceled_flag = 1;
                cancel_order(order);
            }
            pthread_mutex_unlock(&order_mutex);
            sem_post(&oven_sem); //release the oven
            continue;
        }
        pthread_mutex_unlock(&order_mutex);
        simulate_computation_delay_cook(); //simulate cooking time

        pthread_mutex_lock(&order_mutex);
        log_order_status(order, 3, cook->id); //log that the order is ready for delivery
        order->status = 3;
        if (send(order->client_socket, &order->status, sizeof(int), 0) == -1) {
            if (order->canceled_flag == 0) {
                printf("%d th order canceled.\n", order->order_id);
                order->canceled_flag = 1;
                cancel_order(order);
            }
            pthread_mutex_unlock(&order_mutex);
            sem_post(&oven_sem); //release the oven
            continue;
        }
        enqueue_delivery(order); //add the order to the delivery queue
        cook->prepared_orders++;
        pthread_cond_signal(&delivery_cond); //signal delivery persons
        pthread_mutex_unlock(&order_mutex);

        sem_post(&oven_sem); //release the oven
    }

    return NULL;
```

The cook routine handles the preparation of orders by retrieving them from the preparation queue, simulating cooking time, and then moving them to the delivery queue.

1. **Queue Dequeue**
    - Retrieves orders from the preparation queue.
    - The cook thread dequeues an order from the preparation queue. If the queue is empty, it waits on a condition variable.
2. **Order Processing**
    - Simulates cooking time and updates order status.
    - The order status is updated to indicate it is being prepared. The thread sleeps to simulate cooking time, then updates the status to indicate the order is ready.
3. **Queue Enqueue**
    - Moves prepared orders to the delivery queue.
    - The prepared order is enqueued into the delivery queue. Mutexes and condition variables ensure thread-safe 7ccess and signal other threads that a new order is ready for delivery.

Delivery Routine

```c
// Routine for delivery persons to deliver orders
void *delivery_routine(void *arg) {
    DeliveryPerson *delivery_person = (DeliveryPerson *)arg;

    while (1) {
        pthread_mutex_lock(&order_mutex);

        //fill the delivery bag with orders
        while (delivery_person->bag_count < MAX_DELIVERY_BAG && delivery_queue_start != delivery_queue_end) {
            Order *order = dequeue_delivery();
            if (order != NULL) {
                delivery_person->bag[delivery_person->bag_count++] = order;
                log_order_status(order, 4, delivery_person->id); //log that the order is out for delivery
                order->status = 4;
                if (send(order->client_socket, &order->status, sizeof(int), 0) == -1) {
                    if (order->canceled_flag == 0) {
                        printf("%d th order canceled.\n", order->order_id);
                        order->canceled_flag = 1;
                        cancel_order(order);
                    }
                    pthread_mutex_unlock(&order_mutex);
                    continue;
                }
            }
        }

        //if there are orders in the bag deliver them
        if (delivery_person->bag_count > 0) {
            pthread_mutex_unlock(&order_mutex);
            for (int i = 0; i < delivery_person->bag_count; i++) {
                Order *order = delivery_person->bag[i];
                int delivery_time = calculate_delivery_time(order->x, order->y, delivery_person->speed); //calculate delivery time
                //sleep(delivery_time);
                usleep(delivery_time);
                pthread_mutex_lock(&order_mutex);
                log_order_status(order, 5, delivery_person->id); //log that the order was delivered
                order->status = 5;
                if (send(order->client_socket, &order->status, sizeof(int), 0) == -1) {
                    if (order->canceled_flag == 0) {
                        printf("%d th order canceled.\n", order->order_id);
                        order->canceled_flag = 1;
                        cancel_order(order);
                    }
                    pthread_mutex_unlock(&order_mutex);
                    continue;
                }
                delivered_count++;
                delivery_person->delivered_orders++;
                if (delivered_count == order_count) {
                    notify_clients_all_orders_completed(); //notify all clients if all orders are delivered
                }
                pthread_mutex_unlock(&order_mutex);
            }
            delivery_person->bag_count = 0; //empty the bag
        } else {
            pthread_mutex_unlock(&order_mutex);
            sleep(1); //sleep for before checking again
        }
    }

    return NULL;
}
```

The delivery routine manages the delivery of prepared orders by retrieving them from the delivery queue, simulating delivery time, and then updating the order status.

1. **Queue Dequeue**
    - Retrieves orders from the delivery queue.
    - The delivery thread dequeues an order from the delivery queue. If the queue is empty, it waits on a condition variable.
2. **Order Delivery**
    - Simulates delivery time and updates order status.
    - The order status is updated to indicate it is out for delivery. The thread sleeps to simulate delivery time, then updates the status to indicate the order has been delivered.

Manager

```c
// Handle a new customer order
void manager(int socket) {
    int x, y;
    if (recv(socket, &x, sizeof(int), 0) <= 0 || recv(socket, &y, sizeof(int), 0) <= 0) {
        printf("Failed to receive customer coordinates\n");
        close(socket);
        return;
    }

    pthread_mutex_lock(&order_mutex);

    if (order_count < MAX_ORDERS) {
        orders[order_count].order_id = order_count + 1;
        orders[order_count].x = x;
        orders[order_count].y = y;
        orders[order_count].order_time = time(NULL);
        orders[order_count].status = 0; //order received
        orders[order_count].client_socket = socket;
        orders[order_count].canceled_flag = 0; //flag not canceled
        log_order_status(&orders[order_count], 0, -1);
        enqueue_preparation(&orders[order_count]); //add order to preparation queue
        order_count++;
        pthread_cond_signal(&order_cond); //signal cooks
    } else {
        printf("Maximum orders reached. Cannot accept new order.\n");
        close(socket);
    }

    pthread_mutex_unlock(&order_mutex);
}
```

Manager Function

The manager function is responsible for handling client requests and managing orders from creation to preparation.

1. **Order Creation**
    - **Purpose**: Creates a new order access client.
    - **Explanation**: An Order structure is initialized with a unique order ID and the client's socket. The initial status of the order is set to 0 (pending).
2. **Queue Management**
    - **Purpose**: Adds the order to the preparation queue.
    - **Explanation**: The order is enqueued into the preparation queue. Mutexes and condition variables ensure thread-safe access and signal other threads that a new order is available.
3. **Status Updates**
    - **Purpose**: Receives and updates order status.
    - **Explanation**: The function enters a loop where it waits for status updates from the client. If the connection is lost, the loop breaks, and the client socket is closed.

Log Status

```c
//log the status of an order
void log_order_status(Order *order, int status, int thread_id) {
    const char *status_str;
    switch (status) {
        case 0:
            status_str = "Order received";
            break;
        case 1:
            status_str = "Preparing";
            break;
        case 2:
            fprintf(log_file, "Order for client %d is get order into aparatus\n", order->order_id);
            fflush(log_file);
            status_str = "Cooking";
            break;
        case 3:
            fprintf(log_file, "Order for client %d is get order into aparatus\n", order->order_id);
            fflush(log_file);
            status_str = "Ready for delivery";
            break;
        case 4:
            status_str = "Out for delivery";
            break;
        case 5:
            status_str = "Delivered";
            break;
        case 6:
            status_str = "Canceled";
            break;
        default:
            status_str = "Unknown status";
    }

    fprintf(log_file, "Order %d at (%d, %d): %s by thread %d at %s", order->order_id, order->x, order->y, status_str, thread_id, ctime(&order->order_time));
    fflush(log_file);
}
```

This function logs the status of an order to a log file. This is useful for tracking the progress and handling of each order throughout the system.

Enqueue And Dequeue

```c
//enqueue an order for preparation
void enqueue_preparation(Order *order) {
    prep_queue[prep_queue_end++] = order;
    if (prep_queue_end == MAX_ORDERS) prep_queue_end = 0;
}

//dequeue an order for preparation
Order *dequeue_preparation() {
    if (prep_queue_start == prep_queue_end) return NULL;
    Order *order = prep_queue[prep_queue_start++];
    if (prep_queue_start == MAX_ORDERS) prep_queue_start = 0;
    return order;
}

//enqueue an order for cooking
void enqueue_cooking(Order *order) {
    cook_queue[cook_queue_end++] = order;
    if (cook_queue_end == MAX_ORDERS) cook_queue_end = 0;
}

//dequeue an order for cooking
Order *dequeue_cooking() {
    if (cook_queue_start == cook_queue_end) return NULL;
    Order *order = cook_queue[cook_queue_start++];
    if (cook_queue_start == MAX_ORDERS) cook_queue_start = 0;
    return order;
}

//enqueue an order for delivery
void enqueue_delivery(Order *order) {
    delivery_queue[delivery_queue_end++] = order;
    if (delivery_queue_end == MAX_ORDERS) delivery_queue_end = 0;
}

//dequeue an order for delivery
Order *dequeue_delivery() {
    if (delivery_queue_start == delivery_queue_end) return NULL;
    Order *order = delivery_queue[delivery_queue_start++];
    if (delivery_queue_start == MAX_ORDERS) delivery_queue_start = 0;
    return order;
}
```

The functions manage three separate queues for different stages of order processing: preparation, cooking, and delivery. Each queue is implemented as a circular buffer, allowing efficient and continuous use of the array space.

- **Enqueue**: Adds an order to the end of the queue and wraps around if necessary.
- **Dequeue**: Removes an order from the start of the queue and wraps around if necessary.

By using circular buffers, these functions ensure efficient memory usage and provide a way to handle a fixed number of orders in each stage of the process.

Timer With Pseudo Inverse Matrix

```c
//simulate a delay for preparation(30 a 40)
void simulate_computation_delay_prep() {
    int n = 30, m = 40;
    double complex matrix[30][40];
    double complex result[30][30];

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            matrix[i][j] = rand() / (double)RAND_MAX + I * (rand() / (double)RAND_MAX);
        }
    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            result[i][j] = 0;
            for (int k = 0; k < m; k++) {
                result[i][j] += conj(matrix[i][k]) * matrix[j][k];
            }
        }
    }
}

//simulate a delay for cooking half of it (15 e 40)
void simulate_computation_delay_cook() {
    int n = 15, m = 40;
    double complex matrix[15][40];
    double complex result[15][15];

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            matrix[i][j] = rand() / (double)RAND_MAX + I * (rand() / (double)RAND_MAX);
        }
    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            result[i][j] = 0;
            for (int k = 0; k < m; k++) {
                result[i][j] += conj(matrix[i][k]) * matrix[j][k];
            }
        }
    }
}
```

- Both functions simulate computational delays by performing complex matrix multiplications. The simulate_computation_delay_prep function with a 30x40 matrix, while the simulate_computation_delay_cook function with a 15x40 matrix. The purpose of these delays is to mimic the time required for order preparation and cooking

Cancel Order

```c
//cancel an order and update its status
void cancel_order(Order *order) {
    order->status = 6;
    log_order_status(order, 6, -1);
    if (send(order->client_socket, &order->status, sizeof(int), 0) == -1) {
        close(order->client_socket);
        order->client_socket = -1;
    }
}
```

- This function provides a way to cancel an order, update its status, and notify the client.
- The status change is logged for tracking

Print Most Efficient Workers

```c
//print the most efficient workers
void print_most_efficient_workers() {
    int max_prepared_orders = 0;
    int most_efficient_cook_id = -1;
    for (int i = 0; i < cook_pool_size; i++) {
        if (cooks[i].prepared_orders > max_prepared_orders) {
            max_prepared_orders = cooks[i].prepared_orders;
            most_efficient_cook_id = cooks[i].id;
        }
    }
    if (most_efficient_cook_id != -1) {
        printf("Most efficient cook: Cook %d with %d orders prepared\n", most_efficient_cook_id, max_prepared_orders);
    }

    int max_delivered_orders = 0;
    int most_efficient_delivery_person_id = -1;
    for (int i = 0; i < delivery_pool_size; i++) {
        if (delivery_persons[i].delivered_orders > max_delivered_orders) {
            max_delivered_orders = delivery_persons[i].delivered_orders;
            most_efficient_delivery_person_id = delivery_persons[i].id;
        }
    }
    if (most_efficient_delivery_person_id != -1) {
        printf("Most efficient delivery person: Delivery Person %d with %d orders delivered\n", most_efficient_delivery_person_id, max_delivered_orders);
    }
}
```

- The system can track and highlight the contributions of the most productive workers

Client

The client application simulates customer interactions:

1. **Socket Connection**: The client connects to the server using a specified IP address and port.
2. **Order Placement**: Customers place orders, which are sent to the server.
3. **Status Updates**: Clients receive and display status updates for their orders.

Main Function

- **Initialization**: Parses command-line arguments and establishes a connection to the server.
- **Order Placement**: Sends order details to the server and waits for status updates.
- **Status Handling**: Receives and displays order status updates from the server.

```c
int main(int argc, char *argv[]) {
    if (argc != 6) { //check if the correct number of arguments is provided
        printf("Usage: %s <server_ip> <port> <num_clients> <town_size_x> <town_size_y>\n", argv[0]);
        return 1;
    }

    char *server_ip = argv[1]; //server IP address
    int port = atoi(argv[2]); //server port number
    int num_clients = atoi(argv[3]); //number of clients

    if (num_clients > MAX_CLIENTS) { //limit the number of clients if it exceeds the maximum
        printf("Warning: Limiting number of clients to %d\n", MAX_CLIENTS);
        num_clients = MAX_CLIENTS;
    }

    int town_size_x = atoi(argv[4]); //town size x
    int town_size_y = atoi(argv[5]); //town size y
```

- Parse the command line arguments .

```c
signal(SIGINT, signal_handler); //register signal handler for SIGINT
signal(SIGQUIT, signal_handler); //register signal handler for SIGQUIT
```

- The program can handle SIGINT and SIGQUIT signals in a controlled manner, ensuring that important cleanup operations are performed before the program exits.

```c
// Loop to initialize and connect clients
for (int i = 0; i < num_clients; i++) {
    int x = rand() % town_size_x;
    int y = rand() % town_size_y;
    client_sockets[i] = socket(AF_INET, SOCK_STREAM, 0); //create a new socket for the client
    if (client_sockets[i] < 0) { //check if socket creation failed
        printf("Failed to create socket for client %d\n", i + 1);
        continue;
    }

    struct sockaddr_in server_addr; //server address structure
    memset(&server_addr, 0, sizeof(server_addr)); //clear the server address structure
    server_addr.sin_family = AF_INET; //set address family to AF_INET
    server_addr.sin_port = htons(port); //set server port
    inet_pton(AF_INET, server_ip, &server_addr.sin_addr); //convert and set server IP address

    //attempt to connect to the server
    if (connect(client_sockets[i], (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
        printf("Failed to connect client to server\n");
        close(client_sockets[i]); //close the socket if connection fails
        client_sockets[i] = -1; //mark socket as closed
        raise(SIGINT); //raise SIGINT signal to shutdown
    }

    //send coordinates to the server
    if (send(client_sockets[i], &x, sizeof(int), 0) < 0 || send(client_sockets[i], &y, sizeof(int), 0) < 0) {
        printf("Failed to send order coordinates for client %d\n", i + 1);
        close(client_sockets[i]); //close the socket if sending fails
        client_sockets[i] = -1; //mark socket as closed
    } else {
        printf("Order placed at (%d, %d) for client %d\n", x, y, i + 1); //confirm order placement
    }
}
```

1. Loop to Initialize and Connect Clients.
2. Server Address Setup
3. Attempt to Connect to the Server
4. Send Coordinates to the Server

The system can manage multiple client connections efficiently, handle errors appropriately, and maintain robust communication with the server.

```c
int main(int argc, char *argv[]) {

    //handle communication with the server
    while (1) {
        FD_ZERO(&readfds); //clear the set of file descriptors

        //add client sockets to the set
        for (int i = 0; i < num_clients; i++) {
            if (client_sockets[i] != -1) {
                FD_SET(client_sockets[i], &readfds);
            }
        }

        int activity = select(FD_SETSIZE, &readfds, NULL, NULL, NULL); //wait for activity on any socket

        if (activity < 0) { //check if select failed
            perror("select");
            exit(1);
        }

        //loop through clients to check for activity
        for (int i = 0; i < num_clients; i++) {
            if (client_sockets[i] != -1 && FD_ISSET(client_sockets[i], &readfds)) { //check if the socket has activity
                int order_status;
                int recv_result = recv(client_sockets[i], &order_status, sizeof(int), 0); //receive order status

                if (recv_result > 0) { //check if data was received
                    switch (order_status) {
                        case 0:
                            printf("Order placed for client %d\n", i + 1);
                            break;
                        case 1:
                            printf("Order for client %d is being prepared\n", i + 1);
                            break;
                        case 2:
                            printf("Order for client %d is get order into apparatus\n", i + 1);
                            printf("Order for client %d is being cooked\n", i + 1);
                            break;
                        case 3:
                            printf("Order for client %d is get order into apparatus\n", i + 1);
                            printf("Order for client %d is ready for delivery\n", i + 1);
                            break;
                        case 4:
                            printf("Order for client %d is out for delivery\n", i + 1);
                            break;
                        case 5:
                            printf("Order for client %d has been delivered\n", i + 1);
                            close(client_sockets[i]); //close the socket after delivery
                            client_sockets[i] = -1; //mark socket as closed
                            break;
                        case 6:
                            printf("Order for client %d has been canceled\n", i + 1);
                            close(client_sockets[i]); //close the socket after cancellation
                            client_sockets[i] = -1; //mark socket as closed
                            break;
                        default:
                            printf("Unknown status for order of client %d\n", i + 1);
                            break;
                    }
                } else if (recv_result == 0) { //check if the connection was closed
                    close(client_sockets[i]); //close the socket
                    client_sockets[i] = -1; //mark socket as closed
                    canceled_flag = 1; //set the cancel flag
                } else { //chek error
                    perror("recv");
                    close(client_sockets[i]); //close the socket
                    client_sockets[i] = -1; //mark socket as closed
                }
            }
        }

        if (canceled_flag == 1) { //check if any order was canceled
            printf("RIP PIDE SHOP ...\n");
            raise(SIGINT); //raise SIGINT signal to shutdown
        }

        //check if all sockets are closed
        bool all_sockets_closed = true;
        for (int i = 0; i < num_clients; i++) {
            if (client_sockets[i] != -1) {
                all_sockets_closed = false;
                break;
            }
        }

        if (all_sockets_closed) { //if all sockets are closed, shut down the client
            printf("All orders processed. Shutting down client.\n");
            raise(SIGINT); //raise SIGINT signal to shutdown
        }
    }

    //close all client sockets before exiting
    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (client_sockets[i] != -1) {
            close(client_sockets[i]);
        }
    }

    return 0;
}
```

1. Add Client Sockets to the Set
2. Wait for Activity on Any Socket
3. Loop Through Clients to Check for Activity
4. Handle Received Data
5. Check if the Connection was Closed
6. Check if Any Order was Canceled
7. Check if All Sockets are Closed
8. Initiate Shutdown if All Sockets are Closed
9. Close All Client Sockets Before Exiting

Conclusion

This multithreaded food production and delivery system effectively demonstrates the principles of concurrent programming and synchronization using POSIX threads. The server application is designed to handle multiple client requests concurrently, ensuring that the order processing pipeline from order placement to preparation and delivery is efficient and robust.

Efficient Handling of Multiple Client Requests

The server employs a multithreaded architecture where each client connection is managed by a dedicated thread. This design allows the server to handle numerous clients simultaneously without blocking. When a new client connects, the server spawns a new thread using pthread_create() to handle the client's requests independently of other clients. This approach maximizes the server's ability to process multiple orders concurrently, improving overall responsiveness and throughput.

Seamless Interaction Between Server and Client

The interaction between the server and client applications is seamless due to the clear protocol for order submission and status updates. The client sends order details to the server, which are then processed by the server's order management system. The server provides real-time status updates back to the client, ensuring that clients are always informed about the status of their orders. This interaction is facilitated by reliable socket communication, which is established using the socket(), bind(), listen(), and accept() functions on the server side, and the socket() and connect() functions on the client side.

Well-Coordinated Process Involving Cooks and Delivery Personnel

The server coordinates the activities of cooks and delivery personnel through distinct routines:

- **Cook Routine**: Cooks retrieve orders from the preparation queue, simulate preparation time, update the order status, and move the prepared orders to the delivery queue.
- **Delivery Routine**: Delivery personnel retrieve orders from the delivery queue, simulate delivery time, update the order status to "delivered" and notify the client.

These routines are managed by separate threads, ensuring that the preparation and delivery processes can occur concurrently. The use of queues ensures that orders are processed in the correct sequence, maintaining order integrity.

Synchronization Mechanisms for Safe Concurrent Access

Synchronization is a critical aspect of the system to prevent race conditions and ensure data consistency. The following synchronization mechanisms are employed:

- Mutexes: Mutexes (pthread_mutex_t) are used to protect access to shared resources such as the preparation and delivery queues. By locking a mutex before accessing these resources and unlocking it afterward, the system ensures that only one thread can modify the queues at any given time.

- Condition Variables: Condition variables (pthread_cond_t) are used to signal state changes in the queues. For example, when a cook adds an order to the delivery queue, it signals the delivery routine to start processing the order. Similarly, when the preparation queue is empty, cooks wait on a condition variable until new orders are enqueued.

These synchronization mechanisms work together to manage the state transitions of orders from being received, to being prepared, and finally to being delivered. They ensure that threads are efficiently blocked and awakened based on the availability of orders, thereby avoiding busy-waiting and reducing CPU usage.

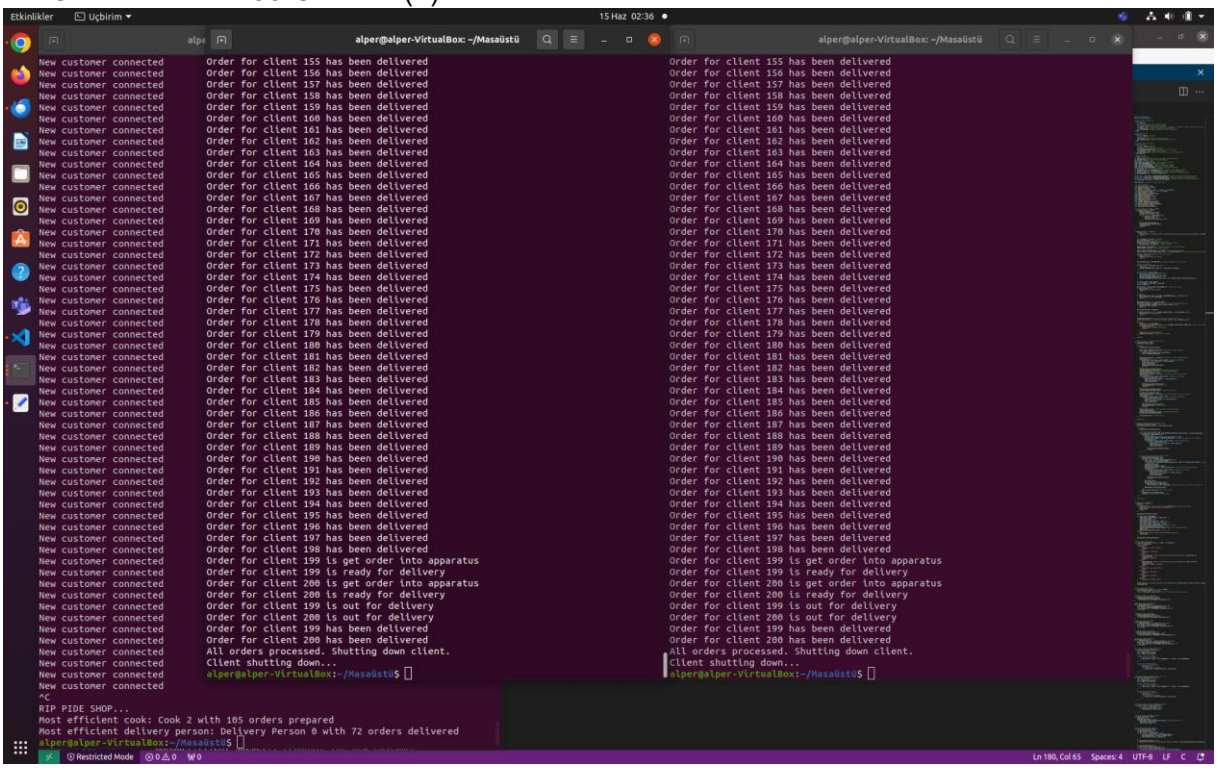
Effective Simulation of a Food Delivery Service

The system effectively simulates a real-world food delivery service by modeling the key components and interactions involved in order management, preparation, and delivery. Clients interact with the server to place orders, which are then handled by a team of cooks and delivery personnel in a coordinated manner. The use of multithreading and synchronization ensures that the system can handle high loads and maintain accurate status tracking for each order.

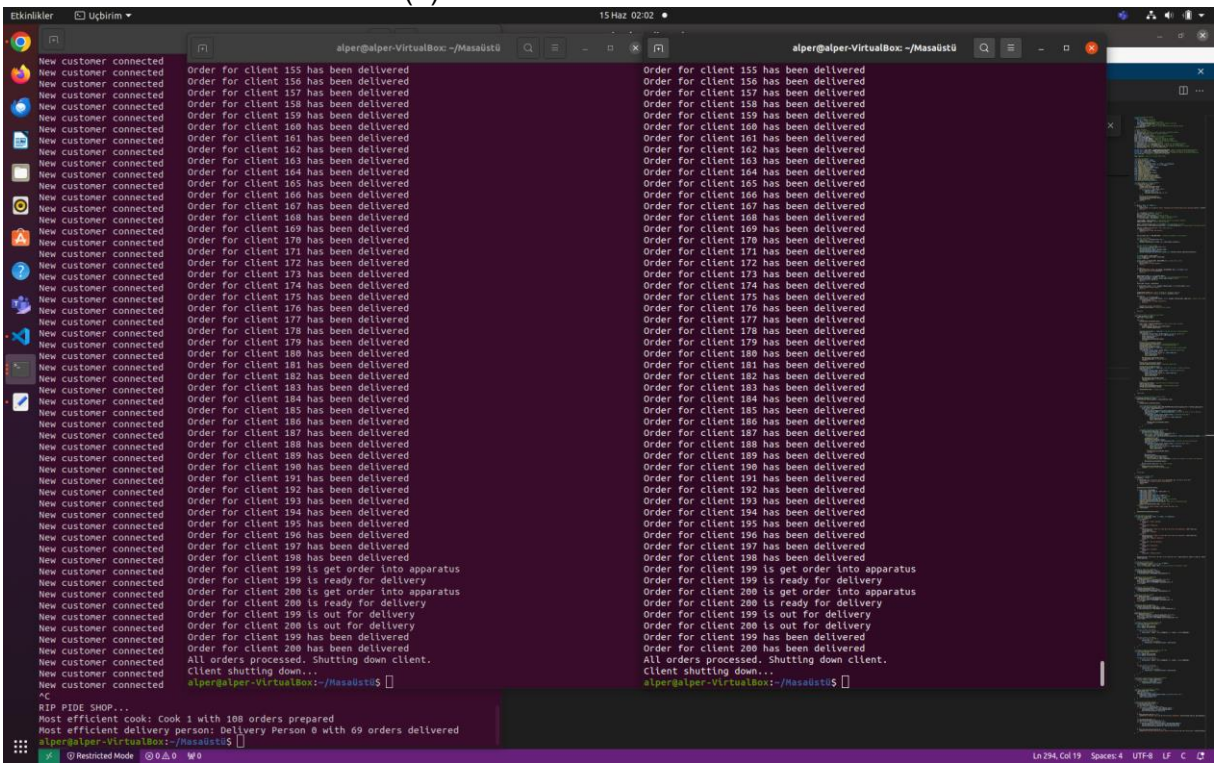
By leveraging POSIX threads and synchronization primitives, the system provides a scalable and efficient solution for managing concurrent tasks, making it a practical example of how multithreaded programming can be applied to solve real-world problems. The careful design of the server and client applications, along with the use of well defined communication protocols, ensures that the system is both robust and responsive.

OUTPUTS

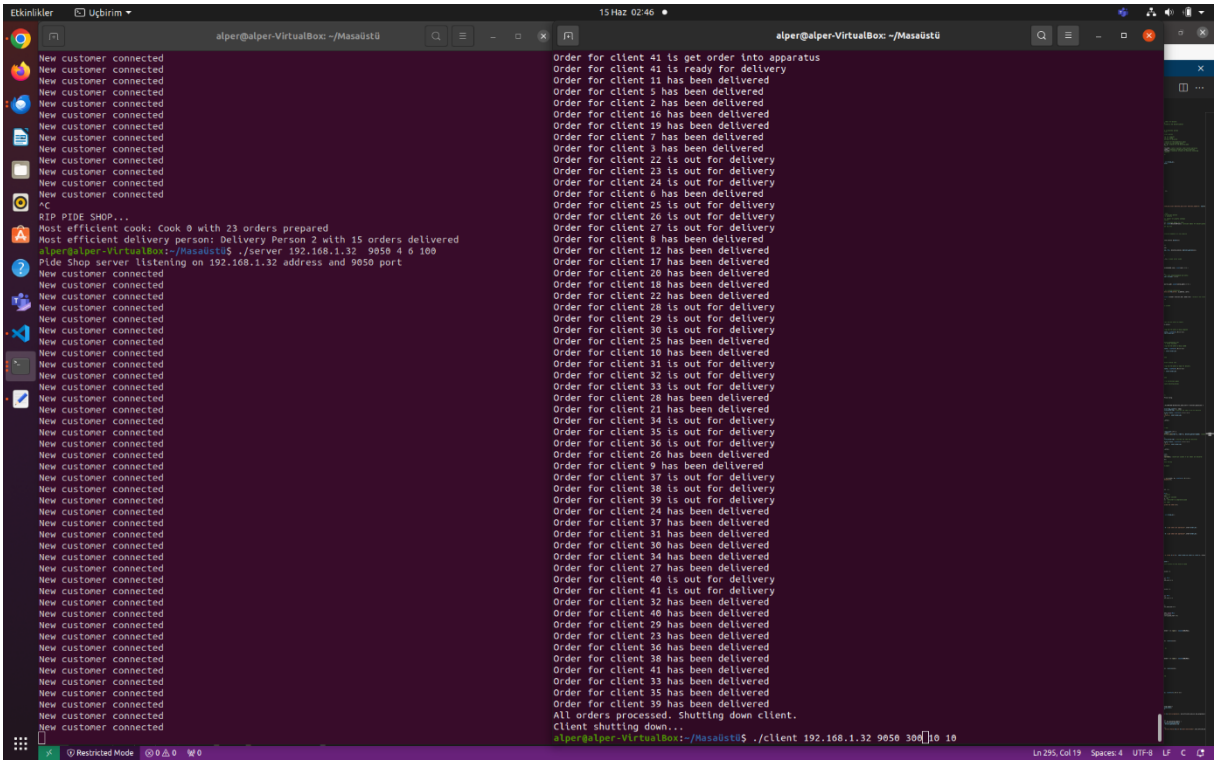
1. TWO TERMINAL 200 CLIENT (1)

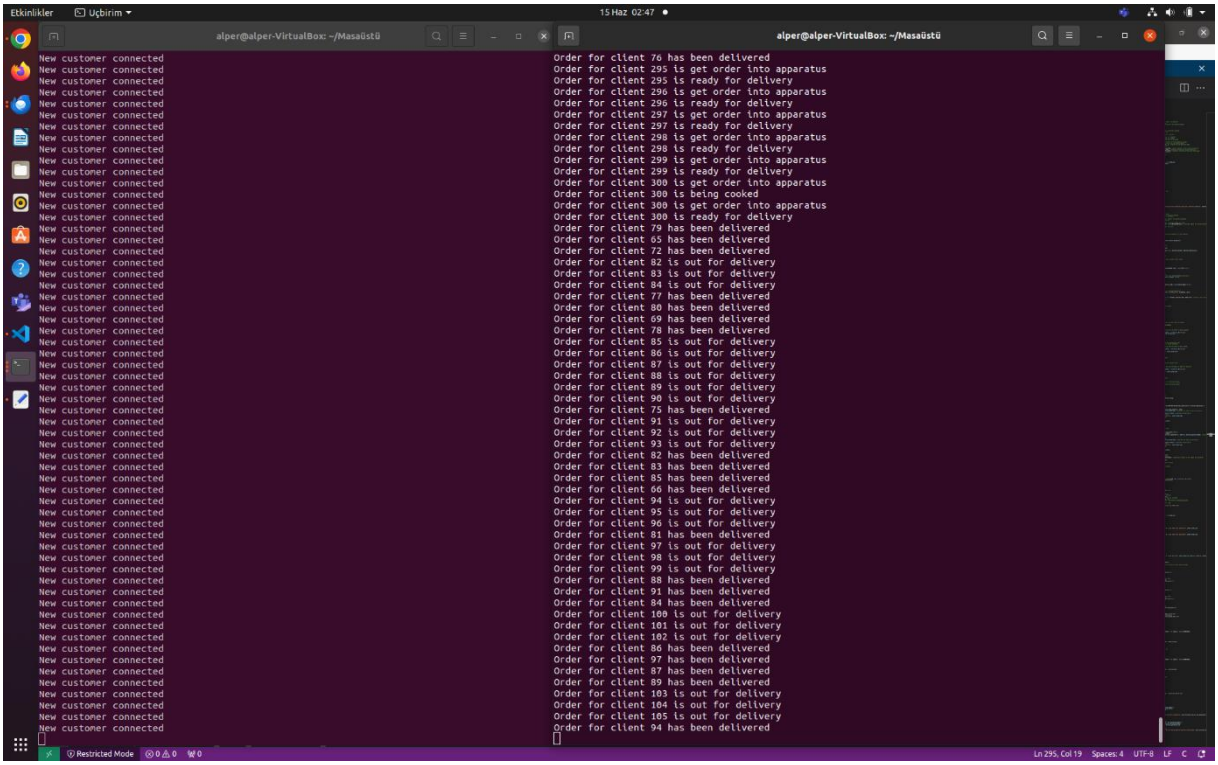


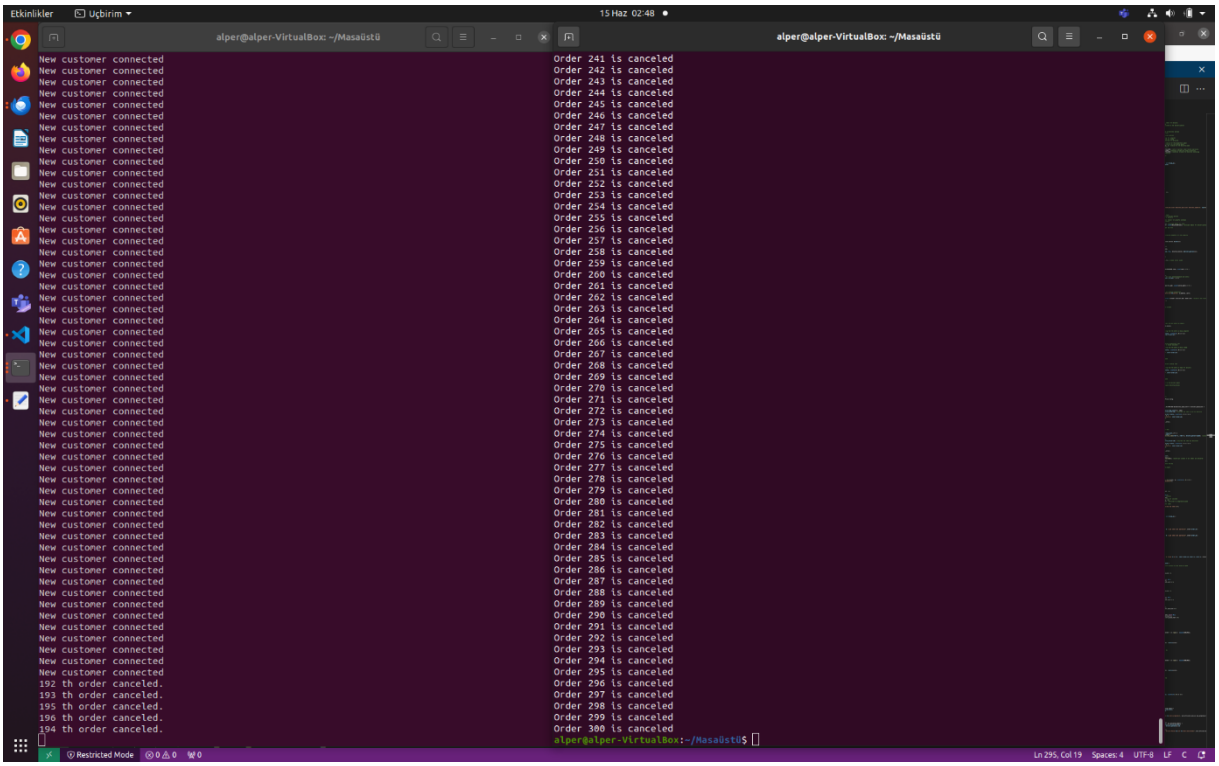
2. TWO TERMINAL 200 CLIENT (2)



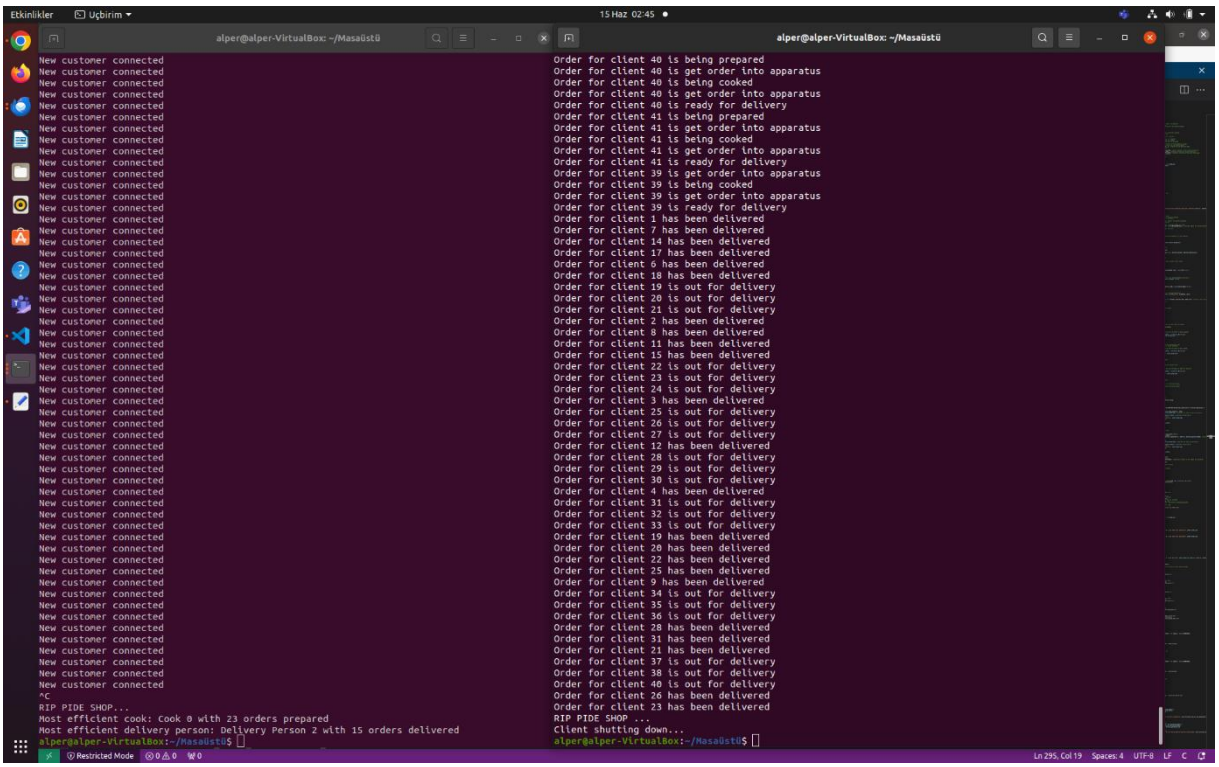
3. 300 ORDER(Include Cancelation)



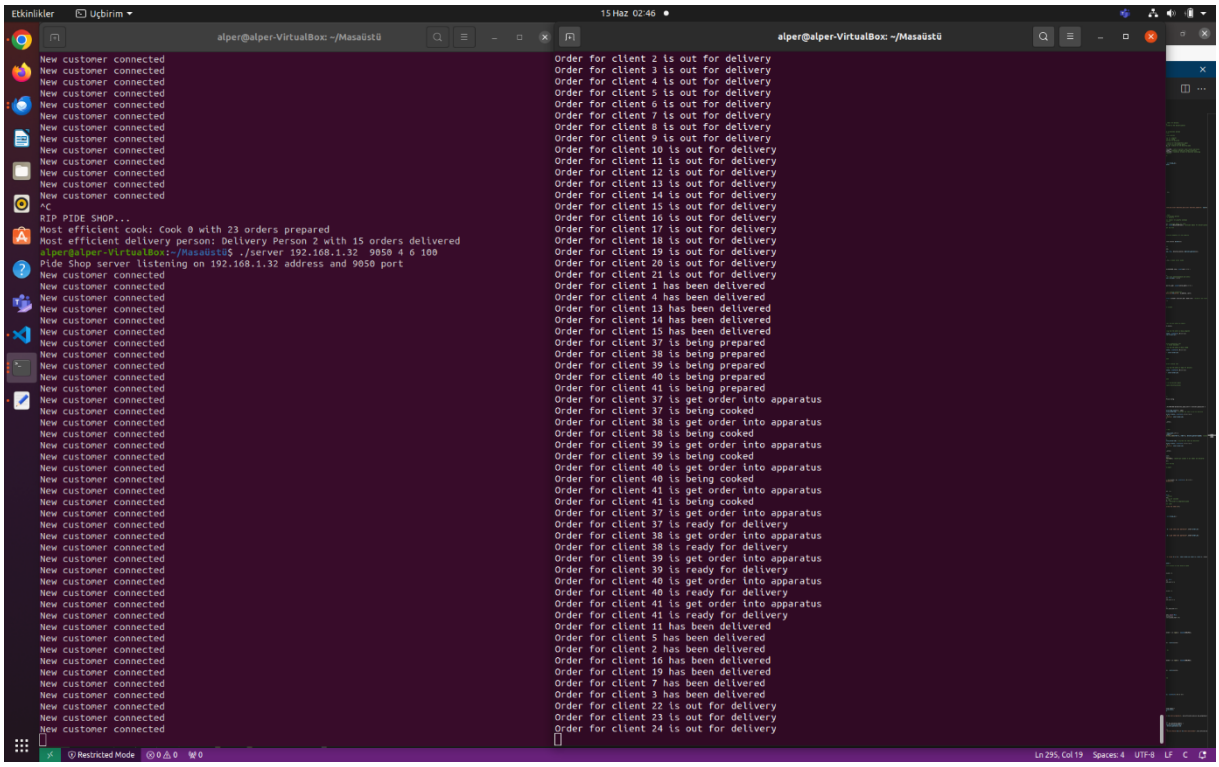


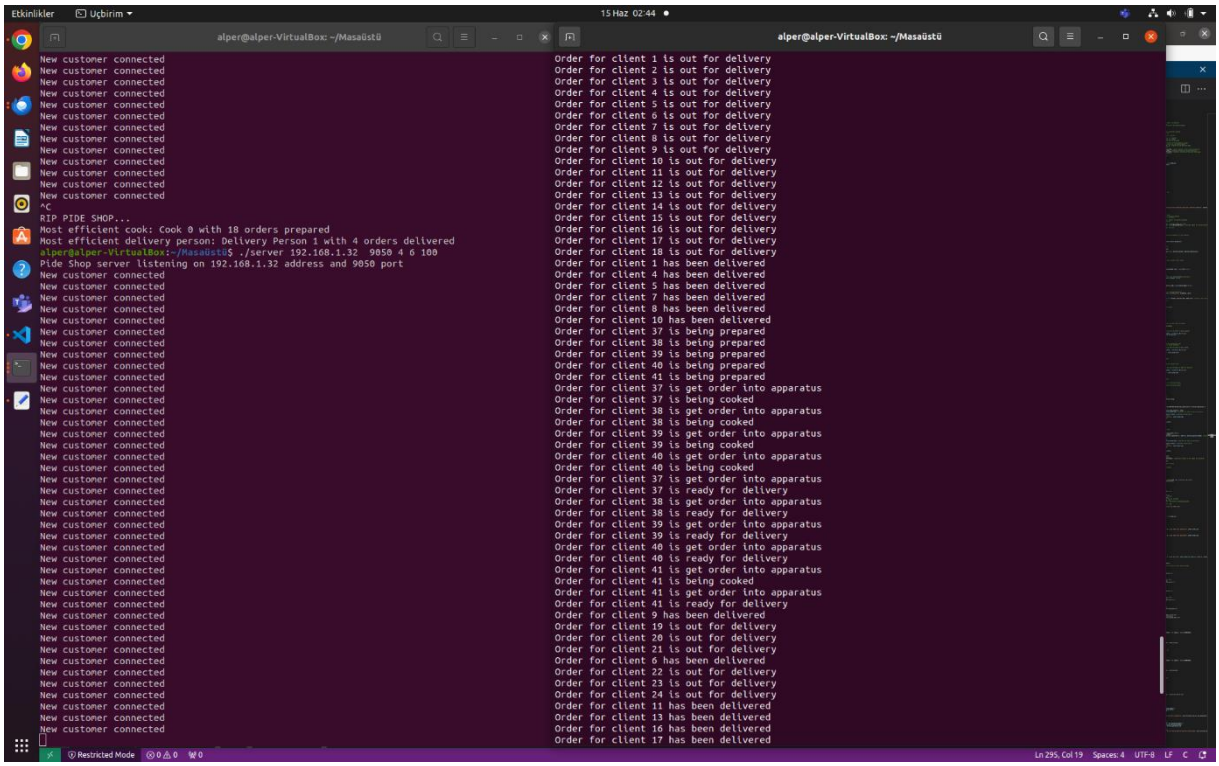


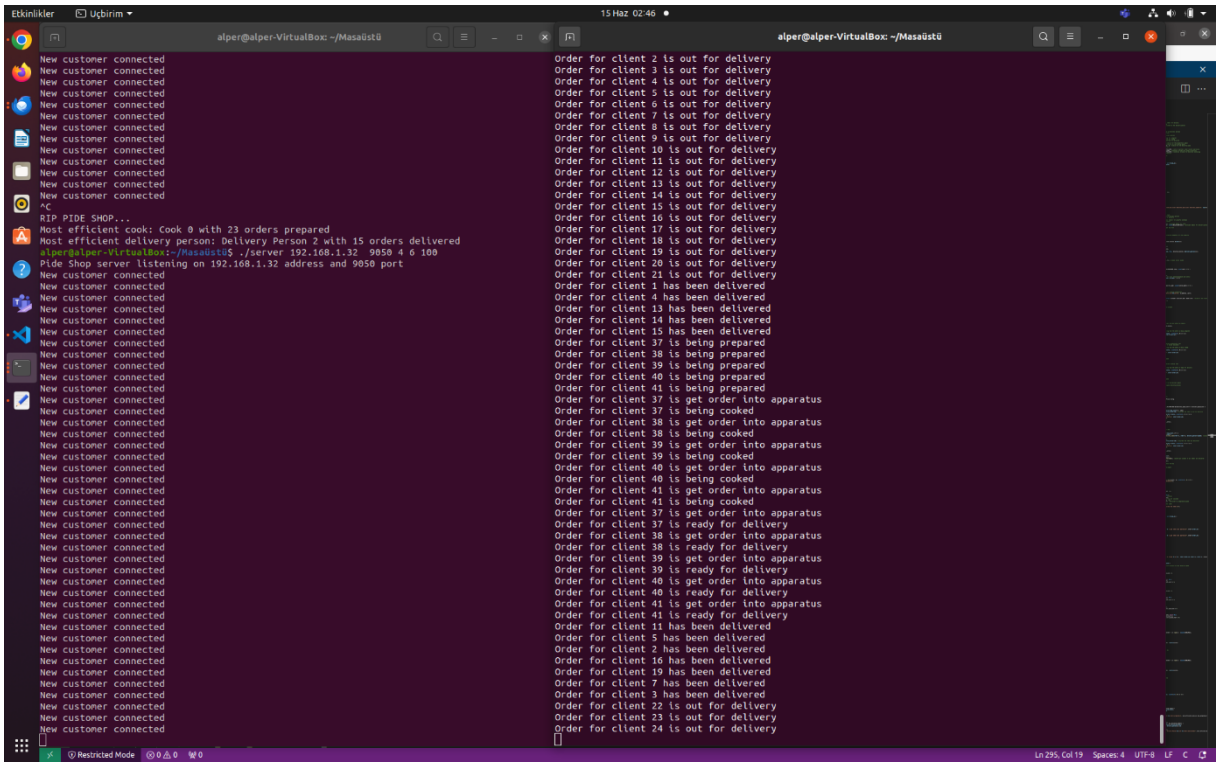
4. SERVER CTRL+C

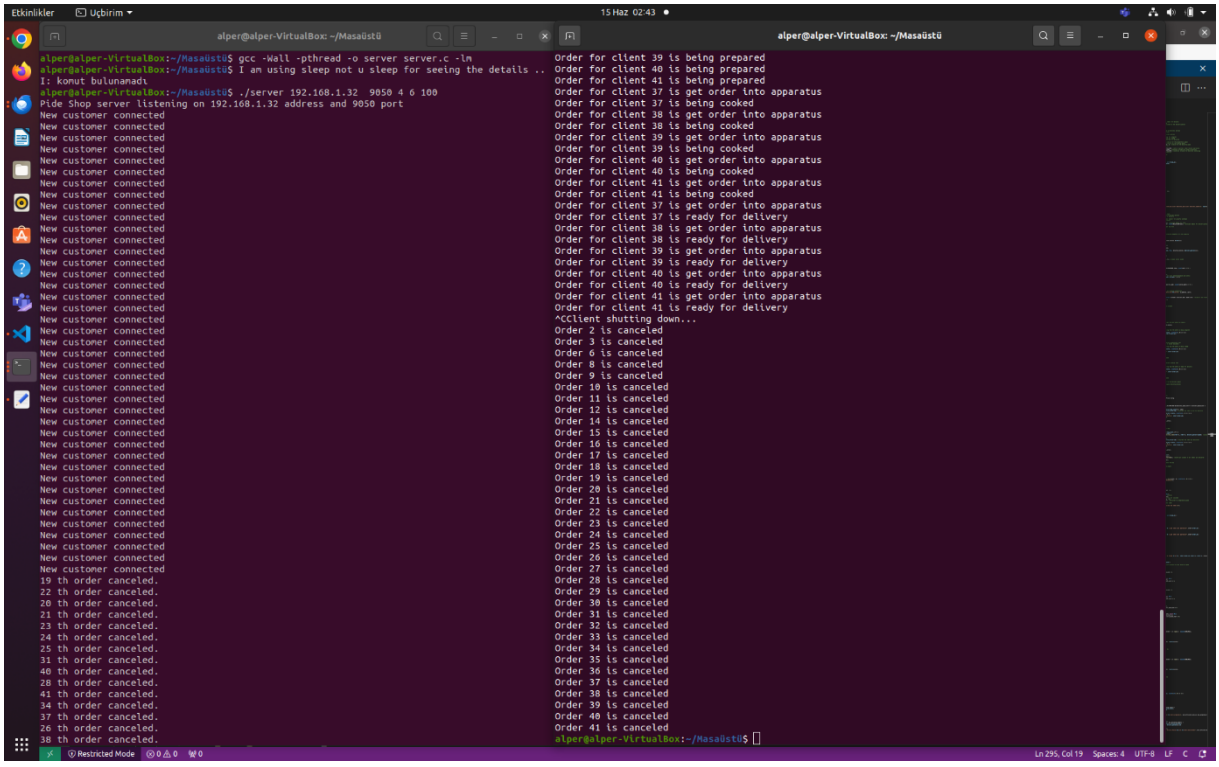


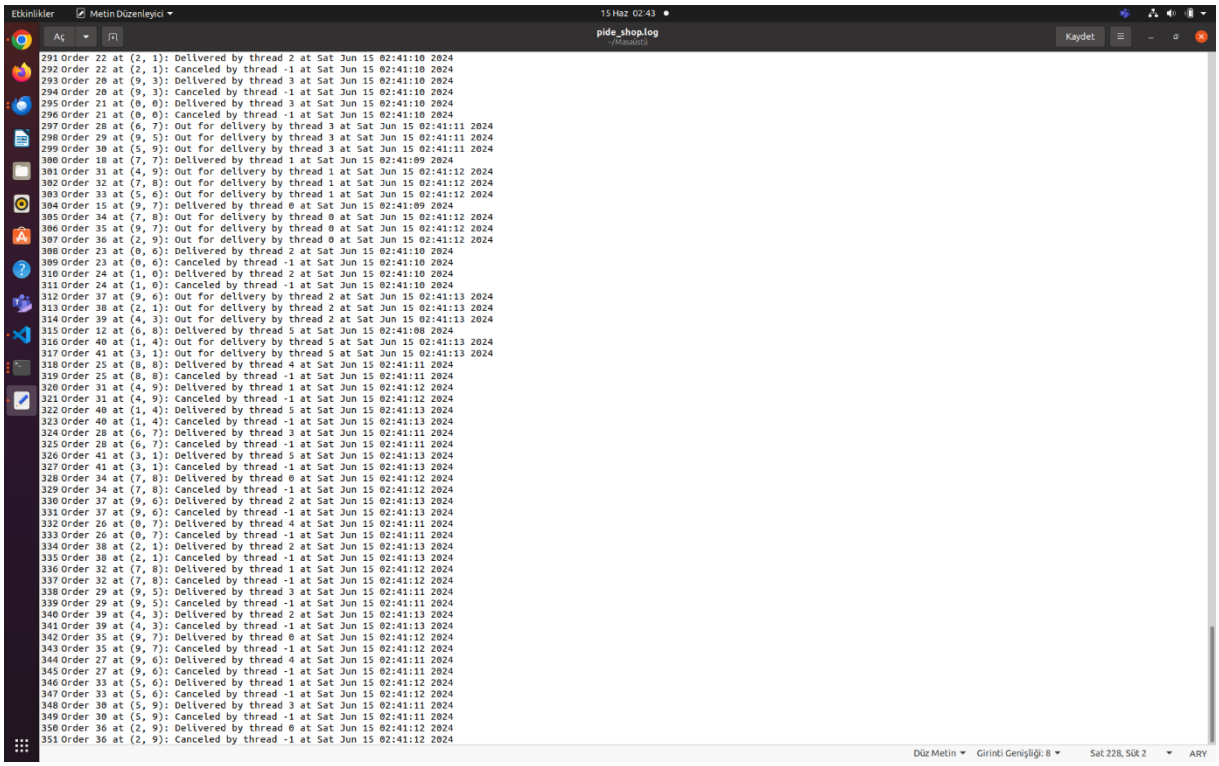
5. 41 ORDER(Include Cancelation)



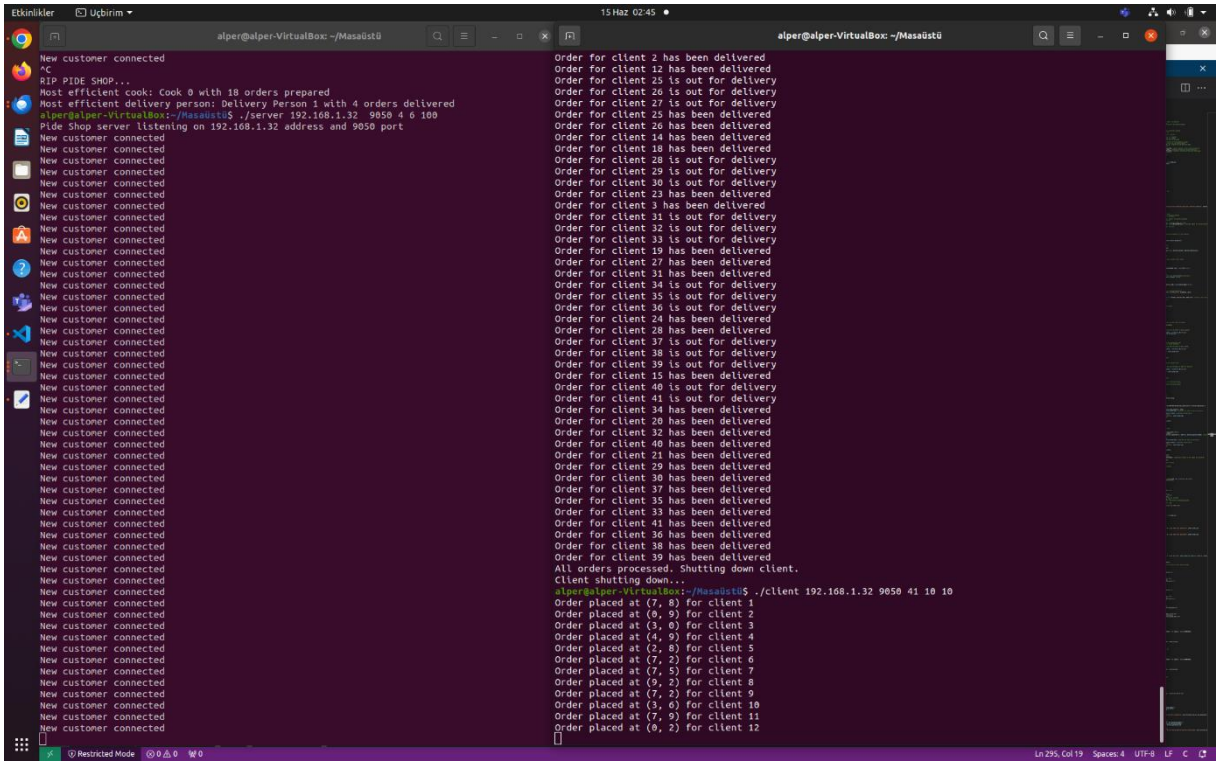



Etkinlikler
Uçbirim
15 Haz 02:46
alper@alper-VirtualBox: ~/Masaüstü
alper@alper-VirtualBox: ~/Masaüstü
New customer connected
^C
RIP PIDE SHOP...
Most efficient cook: Cook 0 with 23 orders prepared
Most efficient delivery person: Delivery Person 2 with 15 orders delivered
alper@alper-VirtualBox:~/Masaüstü$ ./server 192.168.1.32 9050 4 6 100
Pide Shop server listening on 192.168.1.32 address and 9050 port
Order for client 2 is out for delivery
Order for client 3 is out for delivery
Order for client 4 is out for delivery
Order for client 5 is out for delivery
Order for client 6 is out for delivery
Order for client 7 is out for delivery
Order for client 8 is out for delivery
Order for client 9 is out for delivery
Order for client 10 is out for delivery
Order for client 11 is out for delivery
Order for client 12 is out for delivery
Order for client 13 is out for delivery
Order for client 14 is out for delivery
Order for client 15 is out for delivery
Order for client 16 is out for delivery
Order for client 17 is out for delivery
Order for client 18 is out for delivery
Order for client 19 is out for delivery
Order for client 20 is out for delivery
Order for client 21 is out for delivery
Order for client 1 has been delivered
Order for client 4 has been delivered
Order for client 13 has been delivered
Order for client 14 has been delivered
Order for client 15 has been delivered
Order for client 37 is being prepared
Order for client 38 is being prepared
Order for client 39 is being prepared
Order for client 40 is being prepared
Order for client 41 is being prepared
Order for client 37 is get order into apparatus
Order for client 37 is being cooked
Order for client 38 is get order into apparatus
Order for client 38 is being cooked
Order for client 39 is get order into apparatus
Order for client 39 is being cooked
Order for client 40 is get order into apparatus
Order for client 40 is being cooked
Order for client 41 is get order into apparatus
Order for client 41 is being cooked
Order for client 37 is get order into apparatus
Order for client 37 is ready for delivery
Order for client 38 is get order into apparatus
Order for client 38 is ready for delivery
Order for client 39 is get order into apparatus
Order for client 39 is ready for delivery
Order for client 40 is get order into apparatus
Order for client 40 is ready for delivery
Order for client 41 is get order into apparatus
Order for client 41 is ready for delivery
Order for client 11 has been delivered
Order for client 5 has been delivered
Order for client 2 has been delivered
Order for client 16 has been delivered
Order for client 19 has been delivered
Order for client 7 has been delivered
Order for client 3 has been delivered
Order for client 22 is out for delivery
Order for client 23 is out for delivery
Order for client 24 is out for delivery
Restricted Mode
Ln 295, Col 19
Spaces: 4
UTF-8
LF

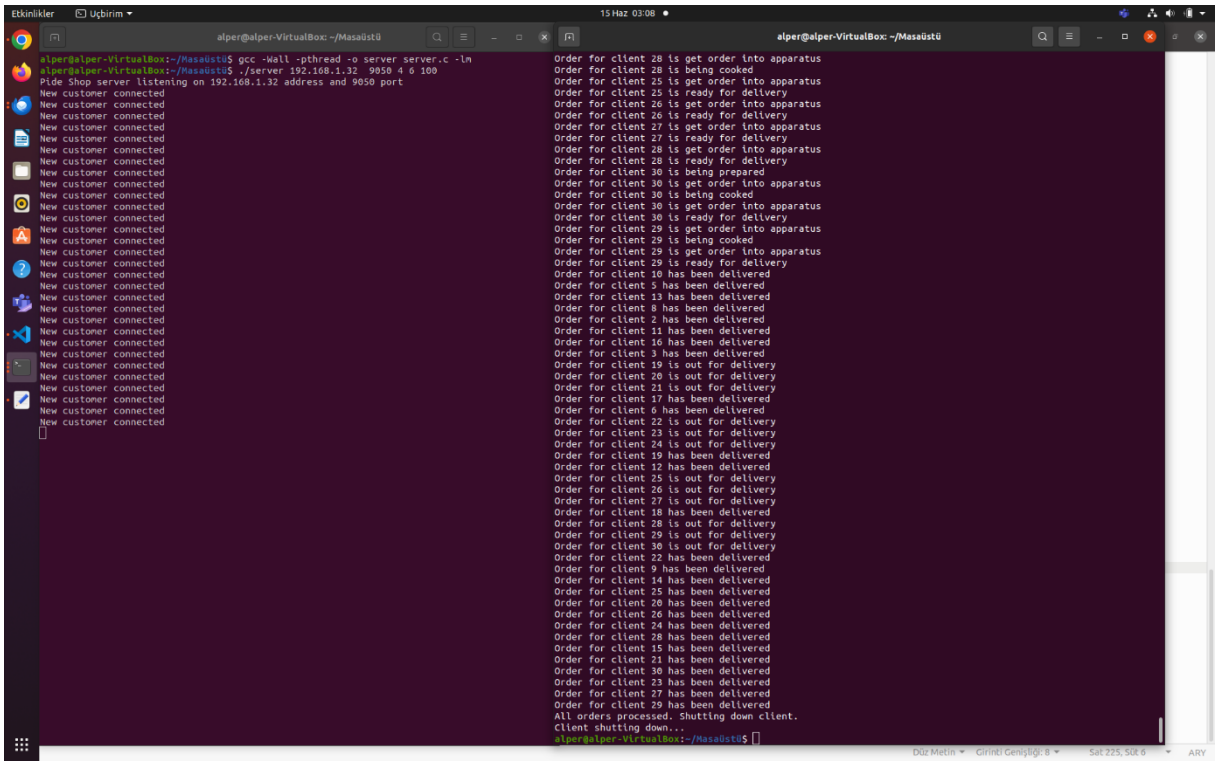

Etkinlikler
Uçbirim
15 Haz 02:43
alper@alper-VirtualBox: ~/Masaüstü
alper@alper-VirtualBox: ~/Masaüstü
alper@alper-VirtualBox:~/Masaüstü$ gcc -Wall -pthread -o server server.c -lm
alper@alper-VirtualBox:~/Masaüstü$ I am using sleep not u sleep for seeing the details ..
I: komut bulunamadı
alper@alper-VirtualBox:~/Masaüstü$ ./server 192.168.1.32 9050 4 6 100
Pide Shop server listening on 192.168.1.32 address and 9050 port
New customer connected
19 th order canceled.
22 th order canceled.
20 th order canceled.
21 th order canceled.
23 th order canceled.
24 th order canceled.
25 th order canceled.
31 th order canceled.
40 th order canceled.
28 th order canceled.
41 th order canceled.
34 th order canceled.
37 th order canceled.
26 th order canceled.
38 th order canceled.
Order for client 39 is being prepared
Order for client 40 is being prepared
Order for client 41 is being prepared
Order for client 37 is get order into apparatus
Order for client 37 is being cooked
Order for client 38 is get order into apparatus
Order for client 38 is being cooked
Order for client 39 is get order into apparatus
Order for client 39 is being cooked
Order for client 40 is get order into apparatus
Order for client 40 is being cooked
Order for client 41 is get order into apparatus
Order for client 41 is being cooked
Order for client 37 is get order into apparatus
Order for client 37 is ready for delivery
Order for client 38 is get order into apparatus
Order for client 38 is ready for delivery
Order for client 39 is get order into apparatus
Order for client 39 is ready for delivery
Order for client 40 is get order into apparatus
Order for client 40 is ready for delivery
Order for client 41 is get order into apparatus
Order for client 41 is ready for delivery
^CClient shutting down...
Order 2 is canceled
Order 3 is canceled
Order 6 is canceled
Order 8 is canceled
Order 9 is canceled
Order 10 is canceled
Order 11 is canceled
Order 12 is canceled
Order 14 is canceled
Order 15 is canceled
Order 16 is canceled
Order 17 is canceled
Order 18 is canceled
Order 19 is canceled
Order 20 is canceled
Order 21 is canceled
Order 22 is canceled
Order 23 is canceled
Order 24 is canceled
Order 25 is canceled
Order 26 is canceled
Order 27 is canceled
Order 28 is canceled
Order 29 is canceled
Order 30 is canceled
Order 31 is canceled
Order 32 is canceled
Order 33 is canceled
Order 34 is canceled
Order 35 is canceled
Order 36 is canceled
Order 37 is canceled
Order 38 is canceled
Order 39 is canceled
Order 40 is canceled
Order 41 is canceled
alper@alper-VirtualBox:~/Masaüstü$
Restricted Mode
Ln 295, Col 19
Spaces: 4
UTF-8
LF

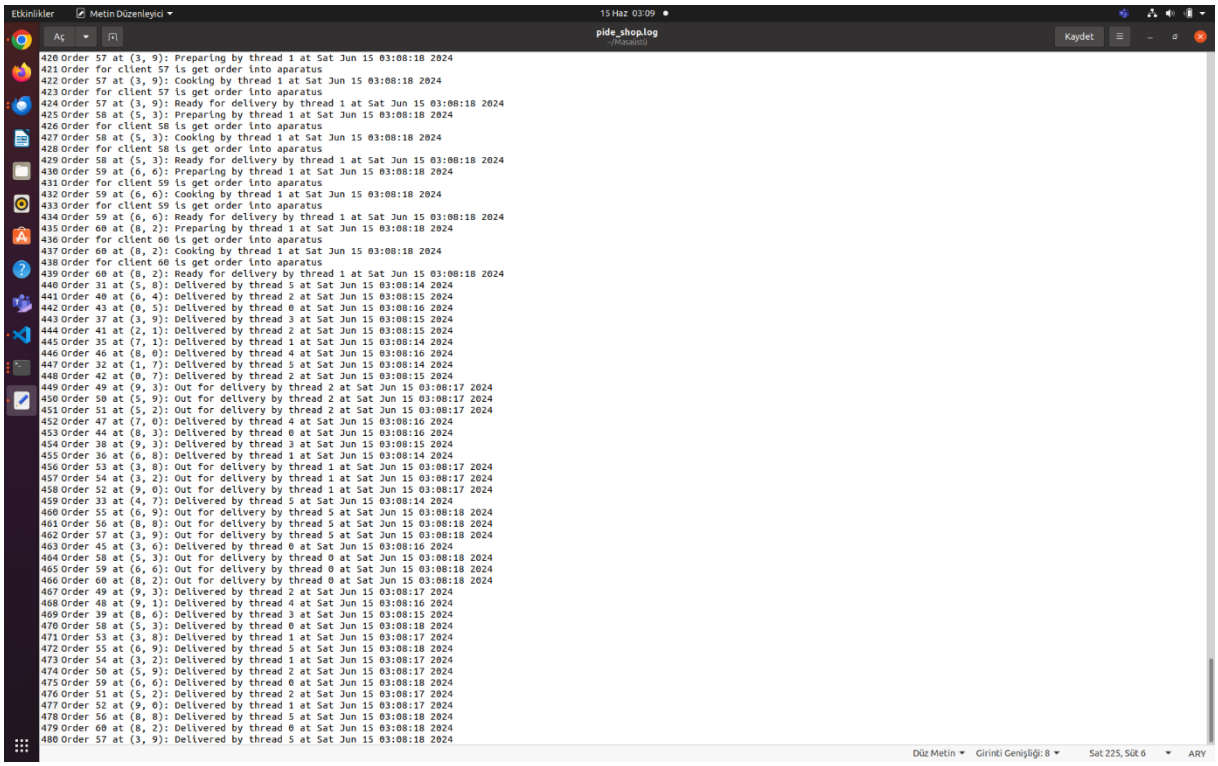


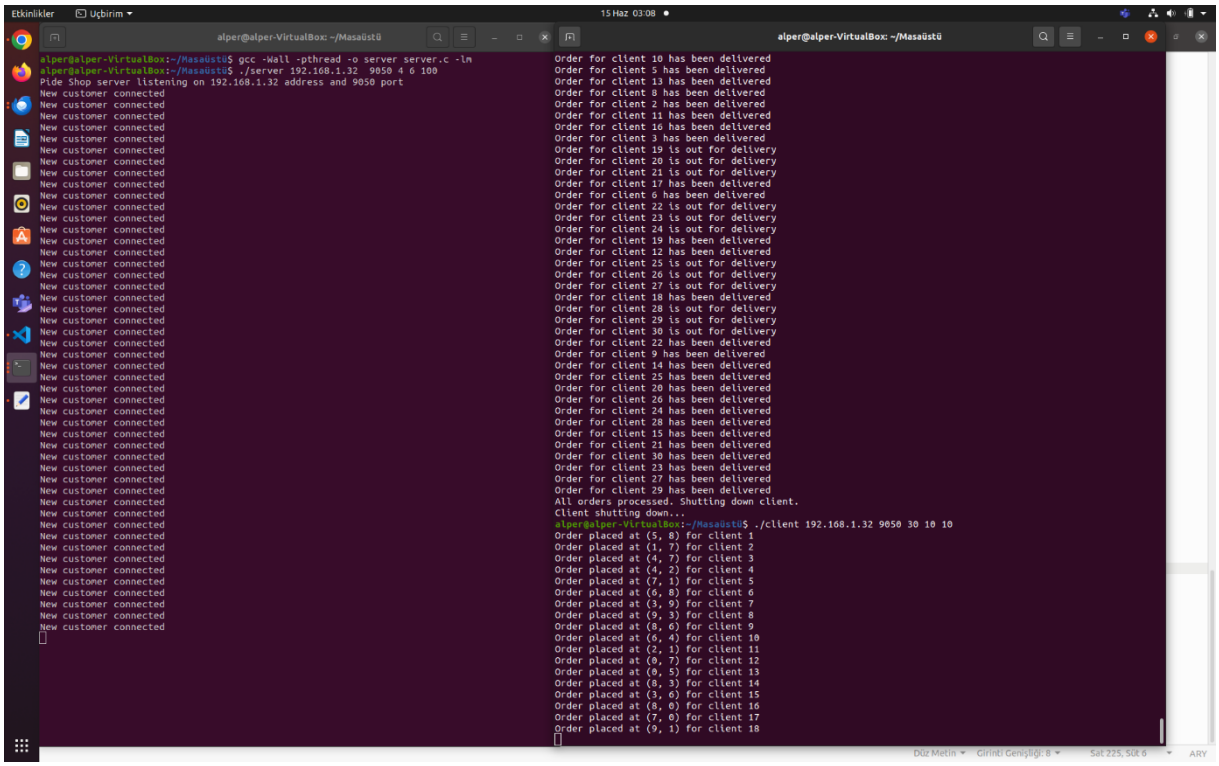
6. After Client Complete Server Still Running(Proof)

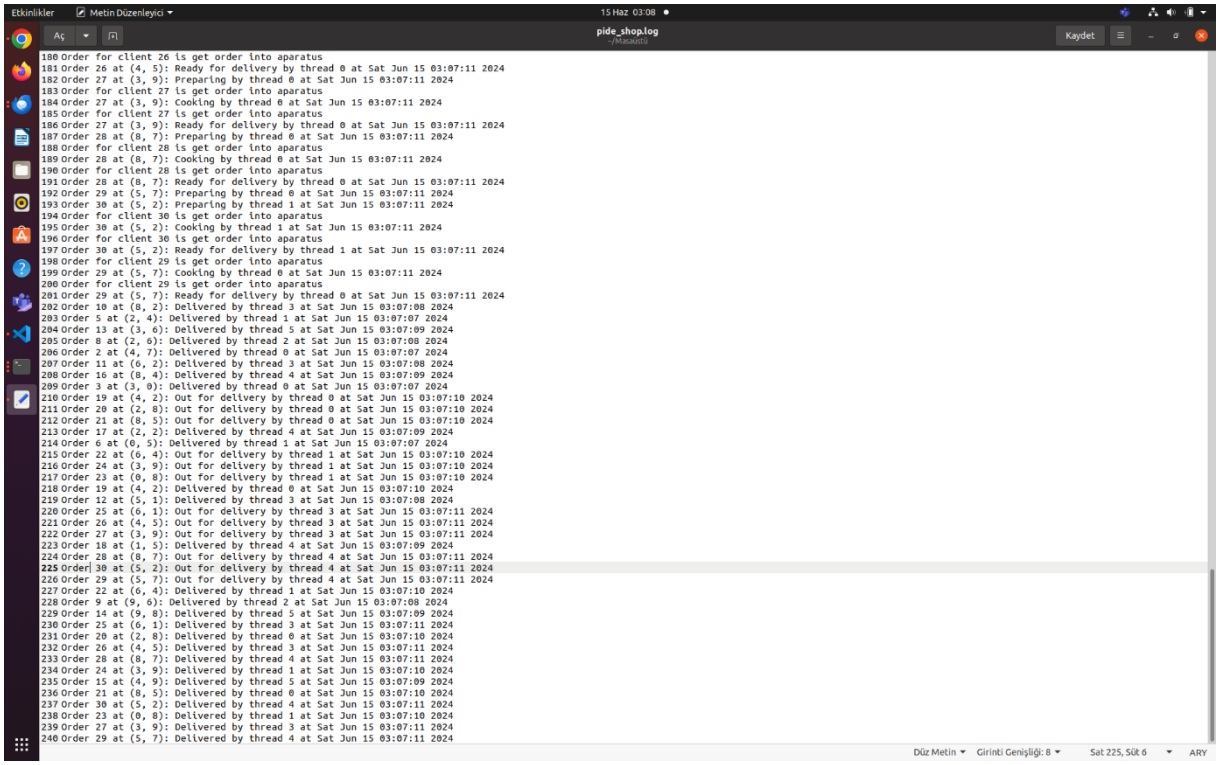


7. 30 ORDER

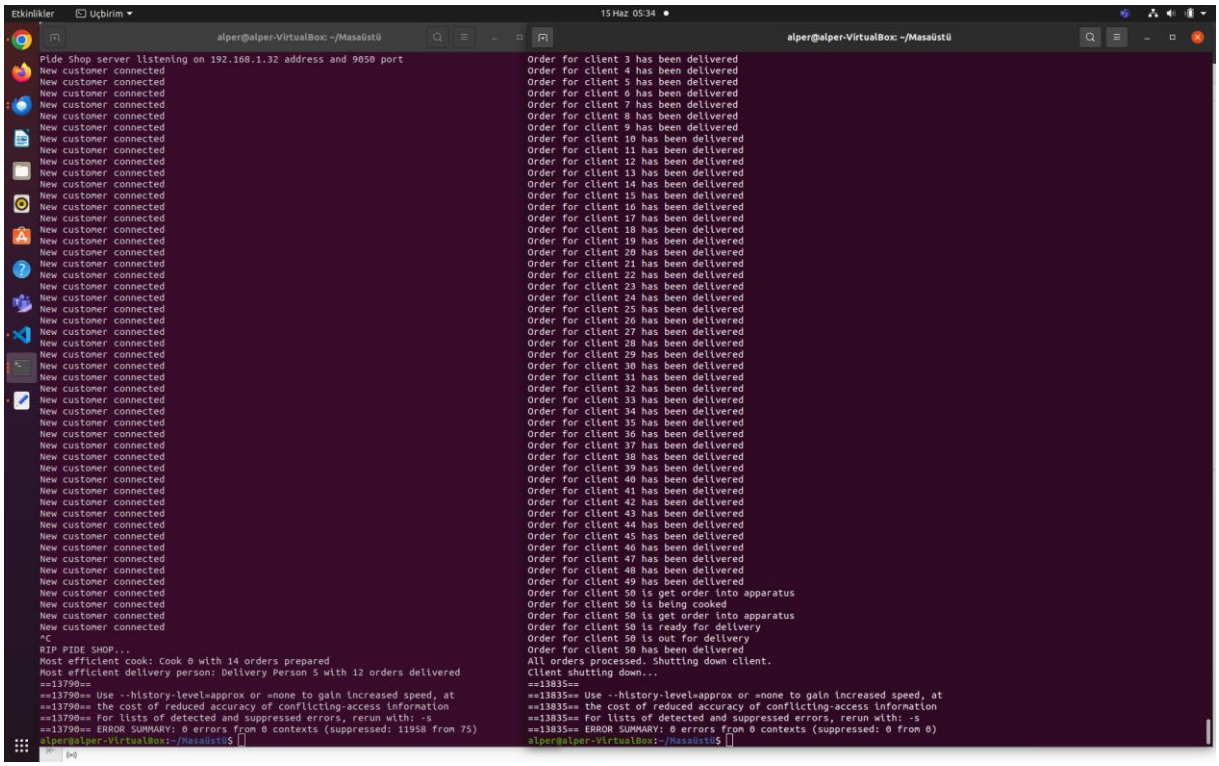

alper@alper-VirtualBox:~/Masaüstü$ gcc -Wall -pthread -o server server.c -lm
alper@alper-VirtualBox:~/Masaüstü$ ./server 192.168.1.32 9050 4 6 100
Pide Shop server listening on 192.168.1.32 address and 9050 port
New customer connected
Order for client 28 is get order into apparatus
Order for client 28 is being cooked
Order for client 25 is get order into apparatus
Order for client 25 is ready for delivery
Order for client 26 is get order into apparatus
Order for client 26 is ready for delivery
Order for client 27 is get order into apparatus
Order for client 27 is ready for delivery
Order for client 28 is get order into apparatus
Order for client 28 is ready for delivery
Order for client 30 is being prepared
Order for client 30 is get order into apparatus
Order for client 30 is being cooked
Order for client 30 is get order into apparatus
Order for client 30 is ready for delivery
Order for client 29 is get order into apparatus
Order for client 29 is being cooked
Order for client 29 is get order into apparatus
Order for client 29 is ready for delivery
Order for client 10 has been delivered
Order for client 5 has been delivered
Order for client 13 has been delivered
Order for client 8 has been delivered
Order for client 2 has been delivered
Order for client 11 has been delivered
Order for client 16 has been delivered
Order for client 3 has been delivered
Order for client 19 is out for delivery
Order for client 20 is out for delivery
Order for client 21 is out for delivery
Order for client 17 has been delivered
Order for client 6 has been delivered
Order for client 22 is out for delivery
Order for client 23 is out for delivery
Order for client 24 is out for delivery
Order for client 19 has been delivered
Order for client 12 has been delivered
Order for client 25 is out for delivery
Order for client 26 is out for delivery
Order for client 27 is out for delivery
Order for client 18 has been delivered
Order for client 28 is out for delivery
Order for client 29 is out for delivery
Order for client 30 is out for delivery
Order for client 22 has been delivered
Order for client 9 has been delivered
Order for client 14 has been delivered
Order for client 25 has been delivered
Order for client 20 has been delivered
Order for client 26 has been delivered
Order for client 24 has been delivered
Order for client 28 has been delivered
Order for client 15 has been delivered
Order for client 21 has been delivered
Order for client 30 has been delivered
Order for client 23 has been delivered
Order for client 27 has been delivered
Order for client 29 has been delivered
All orders processed. Shutting down client.
Client shutting down...
alper@alper-VirtualBox:~/Masaüstü$


pide_shop.log
426 Order 57 at (3, 9): Preparing by thread 1 at Sat Jun 15 03:08:18 2024
427 Order for client 57 is get order into aparatus
428 Order 57 at (3, 9): Cooking by thread 1 at Sat Jun 15 03:08:18 2024
429 Order for client 57 is get order into aparatus
430 Order 57 at (3, 9): Ready for delivery by thread 1 at Sat Jun 15 03:08:18 2024
431 Order 58 at (5, 3): Preparing by thread 1 at Sat Jun 15 03:08:18 2024
432 Order for client 58 is get order into aparatus
433 Order 58 at (5, 3): Cooking by thread 1 at Sat Jun 15 03:08:18 2024
434 Order for client 58 is get order into aparatus
435 Order 58 at (5, 3): Ready for delivery by thread 1 at Sat Jun 15 03:08:18 2024
436 Order 59 at (6, 6): Preparing by thread 1 at Sat Jun 15 03:08:18 2024
437 Order for client 59 is get order into aparatus
438 Order 59 at (6, 6): Cooking by thread 1 at Sat Jun 15 03:08:18 2024
439 Order for client 59 is get order into aparatus
440 Order 59 at (6, 6): Ready for delivery by thread 1 at Sat Jun 15 03:08:18 2024
441 Order 60 at (8, 2): Preparing by thread 1 at Sat Jun 15 03:08:18 2024
442 Order for client 60 is get order into aparatus
443 Order 60 at (8, 2): Cooking by thread 1 at Sat Jun 15 03:08:18 2024
444 Order for client 60 is get order into aparatus
445 Order 60 at (8, 2): Ready for delivery by thread 1 at Sat Jun 15 03:08:18 2024
446 Order 31 at (5, 8): Delivered by thread 5 at Sat Jun 15 03:08:14 2024
447 Order 40 at (6, 4): Delivered by thread 2 at Sat Jun 15 03:08:15 2024
448 Order 43 at (0, 5): Delivered by thread 0 at Sat Jun 15 03:08:16 2024
449 Order 37 at (3, 9): Delivered by thread 3 at Sat Jun 15 03:08:15 2024
450 Order 41 at (2, 1): Delivered by thread 2 at Sat Jun 15 03:08:15 2024
451 Order 35 at (7, 1): Delivered by thread 1 at Sat Jun 15 03:08:14 2024
452 Order 46 at (8, 0): Delivered by thread 4 at Sat Jun 15 03:08:16 2024
453 Order 32 at (1, 7): Delivered by thread 5 at Sat Jun 15 03:08:14 2024
454 Order 42 at (0, 7): Delivered by thread 2 at Sat Jun 15 03:08:15 2024
455 Order 49 at (9, 3): Out for delivery by thread 2 at Sat Jun 15 03:08:17 2024
456 Order 50 at (5, 9): Out for delivery by thread 2 at Sat Jun 15 03:08:17 2024
457 Order 51 at (5, 2): Out for delivery by thread 2 at Sat Jun 15 03:08:17 2024
458 Order 47 at (7, 0): Delivered by thread 4 at Sat Jun 15 03:08:16 2024
459 Order 44 at (8, 3): Delivered by thread 0 at Sat Jun 15 03:08:16 2024
460 Order 38 at (9, 3): Delivered by thread 3 at Sat Jun 15 03:08:15 2024
461 Order 36 at (6, 8): Delivered by thread 1 at Sat Jun 15 03:08:14 2024
462 Order 53 at (3, 8): Out for delivery by thread 1 at Sat Jun 15 03:08:17 2024
463 Order 54 at (3, 2): Out for delivery by thread 1 at Sat Jun 15 03:08:17 2024
464 Order 52 at (9, 0): Out for delivery by thread 1 at Sat Jun 15 03:08:17 2024
465 Order 33 at (4, 7): Delivered by thread 5 at Sat Jun 15 03:08:14 2024
466 Order 55 at (6, 9): Out for delivery by thread 5 at Sat Jun 15 03:08:18 2024
467 Order 56 at (8, 8): Out for delivery by thread 5 at Sat Jun 15 03:08:18 2024
468 Order 57 at (3, 9): Out for delivery by thread 5 at Sat Jun 15 03:08:18 2024
469 Order 45 at (3, 6): Delivered by thread 0 at Sat Jun 15 03:08:16 2024
470 Order 58 at (5, 3): Out for delivery by thread 0 at Sat Jun 15 03:08:18 2024
471 Order 59 at (6, 6): Out for delivery by thread 0 at Sat Jun 15 03:08:18 2024
472 Order 60 at (8, 2): Out for delivery by thread 0 at Sat Jun 15 03:08:18 2024
473 Order 49 at (9, 3): Delivered by thread 2 at Sat Jun 15 03:08:17 2024
474 Order 48 at (9, 1): Delivered by thread 4 at Sat Jun 15 03:08:16 2024
475 Order 39 at (8, 6): Delivered by thread 3 at Sat Jun 15 03:08:15 2024
476 Order 58 at (5, 3): Delivered by thread 0 at Sat Jun 15 03:08:18 2024
477 Order 53 at (3, 8): Delivered by thread 1 at Sat Jun 15 03:08:17 2024
478 Order 55 at (6, 9): Delivered by thread 5 at Sat Jun 15 03:08:18 2024
479 Order 54 at (3, 2): Delivered by thread 1 at Sat Jun 15 03:08:17 2024
480 Order 50 at (5, 9): Delivered by thread 2 at Sat Jun 15 03:08:17 2024
481 Order 59 at (6, 6): Delivered by thread 0 at Sat Jun 15 03:08:18 2024
482 Order 51 at (5, 2): Delivered by thread 2 at Sat Jun 15 03:08:17 2024
483 Order 52 at (9, 0): Delivered by thread 1 at Sat Jun 15 03:08:17 2024
484 Order 56 at (8, 8): Delivered by thread 5 at Sat Jun 15 03:08:18 2024
485 Order 60 at (8, 2): Delivered by thread 0 at Sat Jun 15 03:08:18 2024
486 Order 57 at (3, 9): Delivered by thread 5 at Sat Jun 15 03:08:18 2024

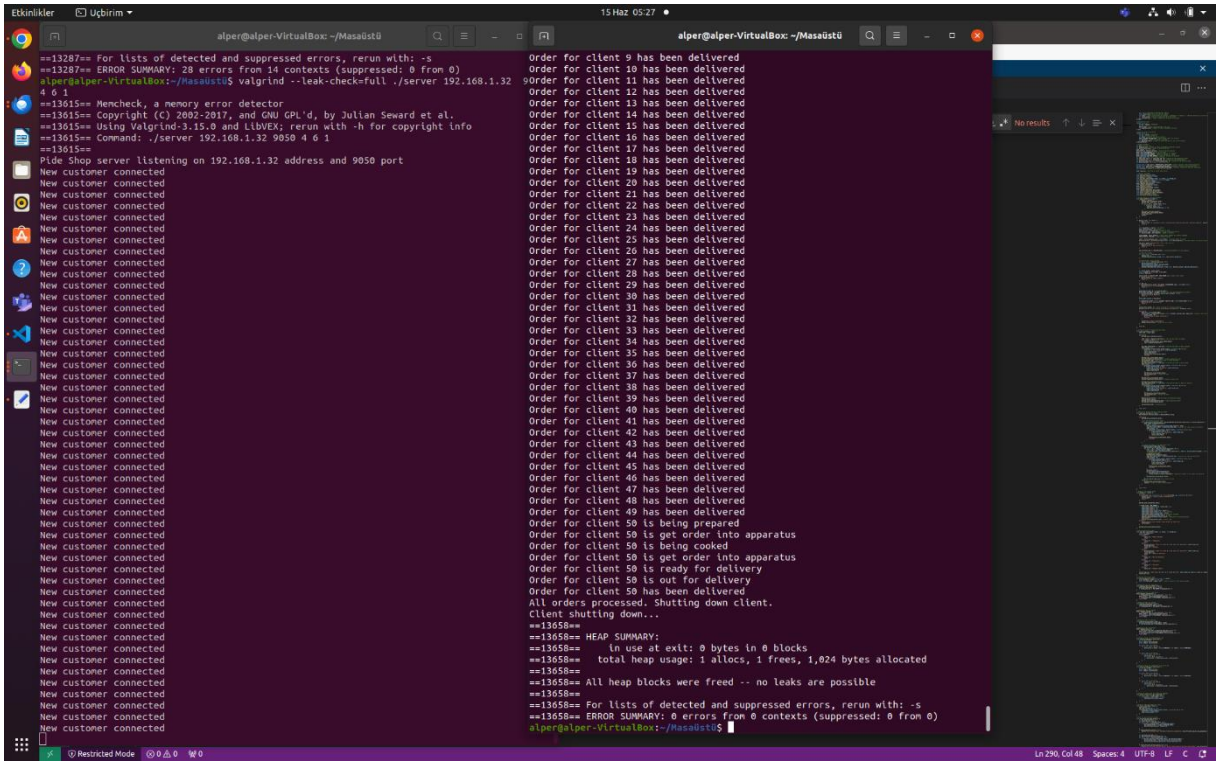

alper@alper-VirtualBox:~/Masaüstü$ gcc -Wall -pthread -o server server.c -lm
alper@alper-VirtualBox:~/Masaüstü$ ./server 192.168.1.32 9050 4 6 100
Pide Shop server listening on 192.168.1.32 address and 9050 port
New customer connected
Order for client 10 has been delivered
Order for client 5 has been delivered
Order for client 13 has been delivered
Order for client 8 has been delivered
Order for client 2 has been delivered
Order for client 11 has been delivered
Order for client 16 has been delivered
Order for client 3 has been delivered
Order for client 19 is out for delivery
Order for client 20 is out for delivery
Order for client 21 is out for delivery
Order for client 17 has been delivered
Order for client 6 has been delivered
Order for client 22 is out for delivery
Order for client 23 is out for delivery
Order for client 24 is out for delivery
Order for client 19 has been delivered
Order for client 12 has been delivered
Order for client 25 is out for delivery
Order for client 26 is out for delivery
Order for client 27 is out for delivery
Order for client 18 has been delivered
Order for client 28 is out for delivery
Order for client 29 is out for delivery
Order for client 30 is out for delivery
Order for client 22 has been delivered
Order for client 9 has been delivered
Order for client 14 has been delivered
Order for client 25 has been delivered
Order for client 20 has been delivered
Order for client 26 has been delivered
Order for client 24 has been delivered
Order for client 28 has been delivered
Order for client 15 has been delivered
Order for client 21 has been delivered
Order for client 30 has been delivered
Order for client 23 has been delivered
Order for client 27 has been delivered
Order for client 29 has been delivered
All orders processed. Shutting down client.
Client shutting down...
alper@alper-VirtualBox:~/Masaüstü$ ./client 192.168.1.32 9050 30 10 10
Order placed at (5, 8) for client 1
Order placed at (1, 7) for client 2
Order placed at (4, 7) for client 3
Order placed at (4, 2) for client 4
Order placed at (7, 1) for client 5
Order placed at (6, 8) for client 6
Order placed at (3, 9) for client 7
Order placed at (9, 3) for client 8
Order placed at (8, 6) for client 9
Order placed at (6, 4) for client 10
Order placed at (2, 1) for client 11
Order placed at (8, 7) for client 12
Order placed at (8, 5) for client 13
Order placed at (8, 3) for client 14
Order placed at (3, 6) for client 15
Order placed at (8, 0) for client 16
Order placed at (7, 0) for client 17
Order placed at (9, 1) for client 18

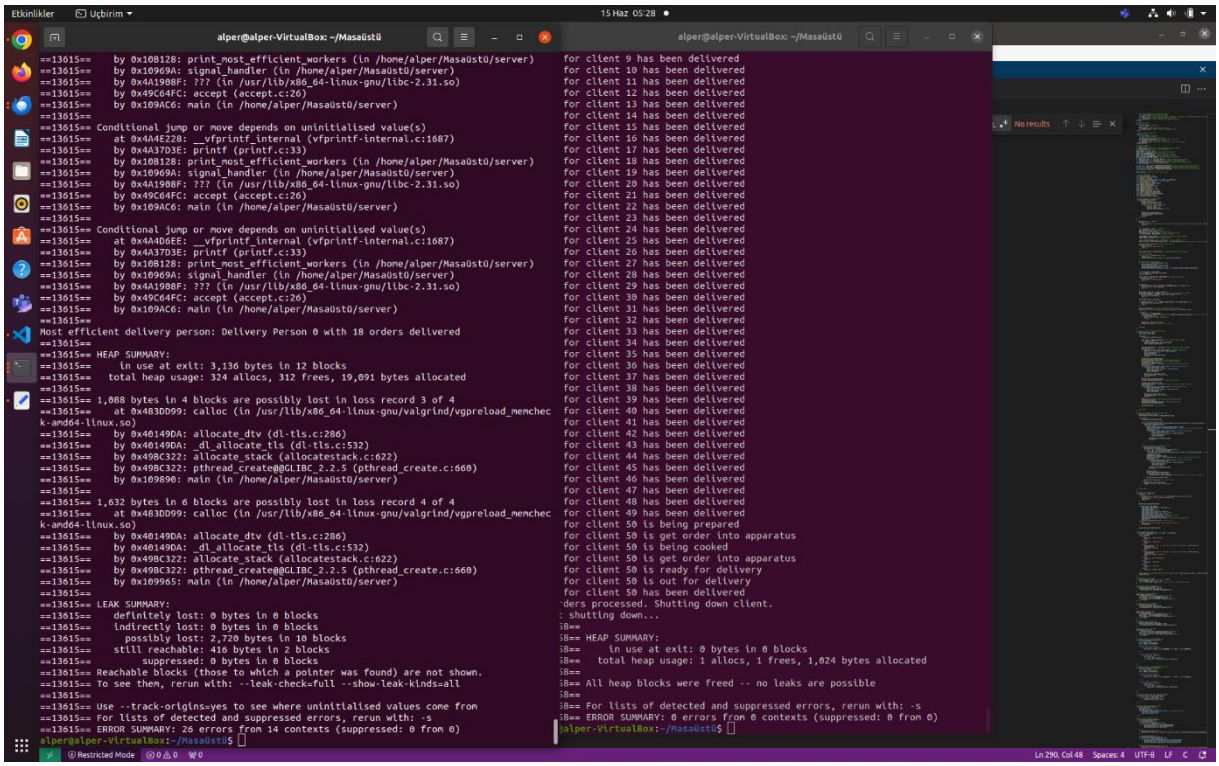

pide_shop.log
180 Order for client 26 is get order into aparatus
181 Order 26 at (4, 5): Ready for delivery by thread 0 at Sat Jun 15 03:07:11 2024
182 Order 27 at (3, 9): Preparing by thread 0 at Sat Jun 15 03:07:11 2024
183 Order for client 27 is get order into aparatus
184 Order 27 at (3, 9): Cooking by thread 0 at Sat Jun 15 03:07:11 2024
185 Order for client 27 is get order into aparatus
186 Order 27 at (3, 9): Ready for delivery by thread 0 at Sat Jun 15 03:07:11 2024
187 Order 28 at (8, 7): Preparing by thread 0 at Sat Jun 15 03:07:11 2024
188 Order for client 28 is get order into aparatus
189 Order 28 at (8, 7): Cooking by thread 0 at Sat Jun 15 03:07:11 2024
190 Order for client 28 is get order into aparatus
191 Order 28 at (8, 7): Ready for delivery by thread 0 at Sat Jun 15 03:07:11 2024
192 Order 29 at (5, 7): Preparing by thread 0 at Sat Jun 15 03:07:11 2024
193 Order 30 at (5, 2): Preparing by thread 1 at Sat Jun 15 03:07:11 2024
194 Order for client 30 is get order into aparatus
195 Order 30 at (5, 2): Cooking by thread 1 at Sat Jun 15 03:07:11 2024
196 Order for client 30 is get order into aparatus
197 Order 30 at (5, 2): Ready for delivery by thread 1 at Sat Jun 15 03:07:11 2024
198 Order for client 29 is get order into aparatus
199 Order 29 at (5, 7): Cooking by thread 0 at Sat Jun 15 03:07:11 2024
200 Order for client 29 is get order into aparatus
201 Order 29 at (5, 7): Ready for delivery by thread 0 at Sat Jun 15 03:07:11 2024
202 Order 10 at (8, 2): Delivered by thread 3 at Sat Jun 15 03:07:08 2024
203 Order 5 at (2, 4): Delivered by thread 1 at Sat Jun 15 03:07:07 2024
204 Order 13 at (3, 6): Delivered by thread 5 at Sat Jun 15 03:07:09 2024
205 Order 8 at (2, 6): Delivered by thread 2 at Sat Jun 15 03:07:08 2024
206 Order 2 at (4, 7): Delivered by thread 0 at Sat Jun 15 03:07:07 2024
207 Order 11 at (6, 2): Delivered by thread 3 at Sat Jun 15 03:07:08 2024
208 Order 16 at (8, 4): Delivered by thread 4 at Sat Jun 15 03:07:09 2024
209 Order 3 at (3, 0): Delivered by thread 0 at Sat Jun 15 03:07:07 2024
210 Order 19 at (4, 2): Out for delivery by thread 0 at Sat Jun 15 03:07:10 2024
211 Order 20 at (2, 8): Out for delivery by thread 0 at Sat Jun 15 03:07:10 2024
212 Order 21 at (8, 5): Out for delivery by thread 0 at Sat Jun 15 03:07:10 2024
213 Order 17 at (2, 2): Delivered by thread 4 at Sat Jun 15 03:07:09 2024
214 Order 6 at (8, 5): Delivered by thread 1 at Sat Jun 15 03:07:07 2024
215 Order 22 at (6, 4): Out for delivery by thread 1 at Sat Jun 15 03:07:10 2024
216 Order 24 at (3, 9): Out for delivery by thread 1 at Sat Jun 15 03:07:10 2024
217 Order 23 at (0, 8): Out for delivery by thread 1 at Sat Jun 15 03:07:10 2024
218 Order 19 at (4, 2): Delivered by thread 0 at Sat Jun 15 03:07:10 2024
219 Order 12 at (5, 1): Delivered by thread 3 at Sat Jun 15 03:07:08 2024
220 Order 25 at (6, 1): Out for delivery by thread 3 at Sat Jun 15 03:07:11 2024
221 Order 26 at (4, 5): Out for delivery by thread 3 at Sat Jun 15 03:07:11 2024
222 Order 27 at (3, 9): Out for delivery by thread 3 at Sat Jun 15 03:07:11 2024
223 Order 18 at (1, 5): Delivered by thread 4 at Sat Jun 15 03:07:09 2024
224 Order 28 at (8, 7): Out for delivery by thread 4 at Sat Jun 15 03:07:11 2024
225 Order 30 at (5, 2): Out for delivery by thread 4 at Sat Jun 15 03:07:11 2024
226 Order 29 at (5, 7): Out for delivery by thread 4 at Sat Jun 15 03:07:11 2024
227 Order 22 at (6, 4): Delivered by thread 1 at Sat Jun 15 03:07:10 2024
228 Order 9 at (9, 6): Delivered by thread 2 at Sat Jun 15 03:07:08 2024
229 Order 14 at (9, 8): Delivered by thread 5 at Sat Jun 15 03:07:09 2024
230 Order 25 at (6, 1): Delivered by thread 3 at Sat Jun 15 03:07:11 2024
231 Order 20 at (2, 8): Delivered by thread 0 at Sat Jun 15 03:07:10 2024
232 Order 26 at (4, 5): Delivered by thread 3 at Sat Jun 15 03:07:11 2024
233 Order 28 at (8, 7): Delivered by thread 4 at Sat Jun 15 03:07:11 2024
234 Order 24 at (3, 9): Delivered by thread 1 at Sat Jun 15 03:07:10 2024
235 Order 15 at (4, 9): Delivered by thread 5 at Sat Jun 15 03:07:09 2024
236 Order 21 at (8, 5): Delivered by thread 0 at Sat Jun 15 03:07:10 2024
237 Order 30 at (5, 2): Delivered by thread 4 at Sat Jun 15 03:07:11 2024
238 Order 23 at (0, 8): Delivered by thread 1 at Sat Jun 15 03:07:10 2024
239 Order 27 at (3, 9): Delivered by thread 3 at Sat Jun 15 03:07:11 2024
240 Order 29 at (5, 7): Delivered by thread 4 at Sat Jun 15 03:07:11 2024

8) helgrind

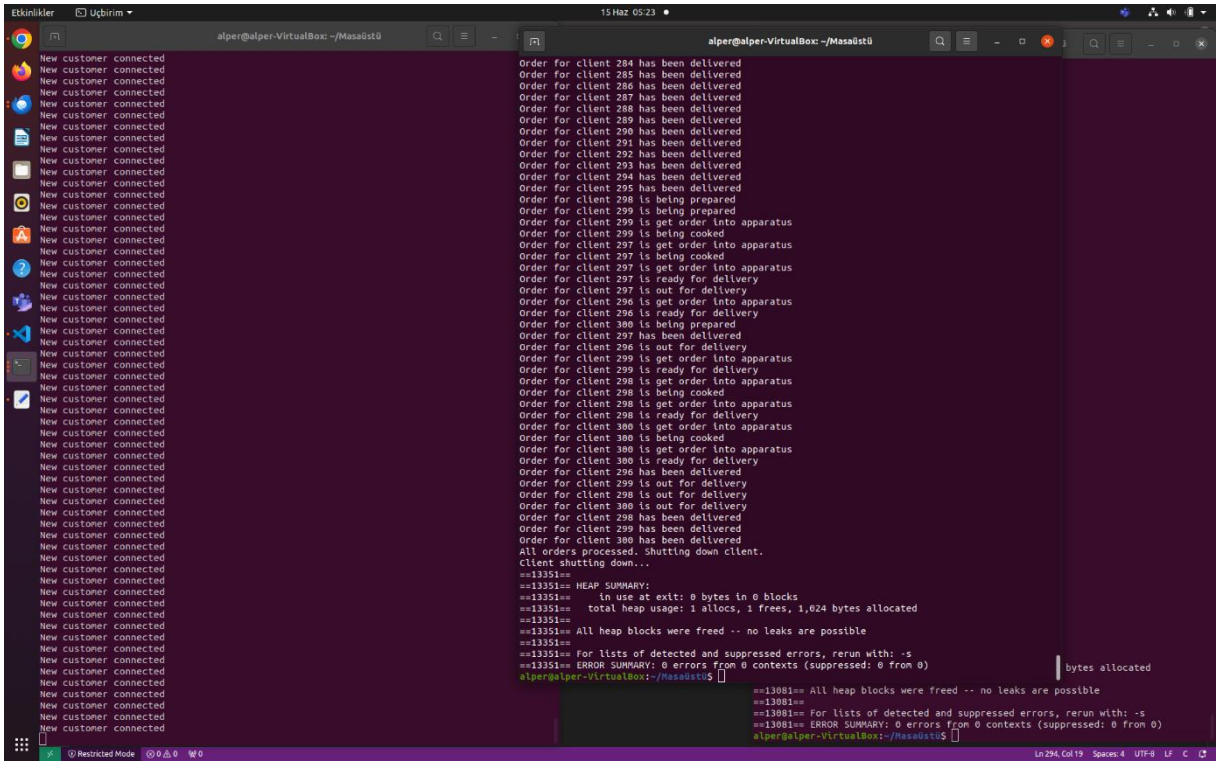


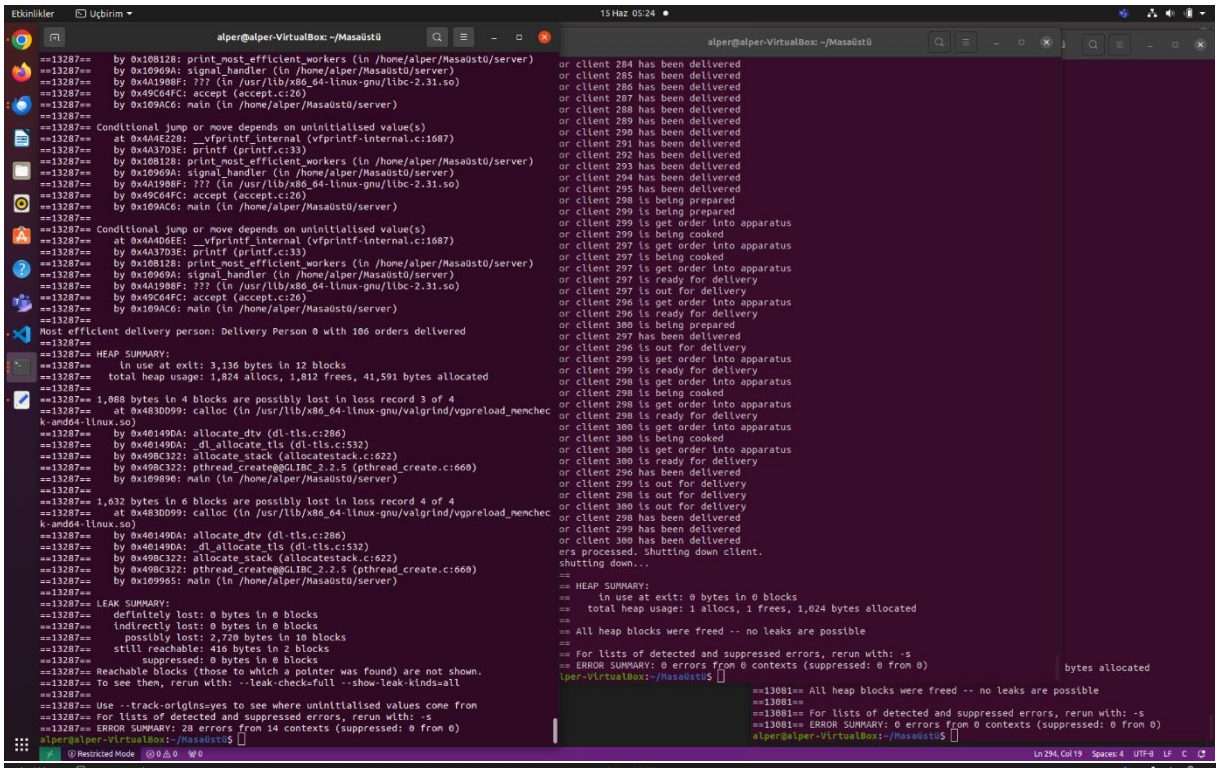
9)valgrind(Unnecessary)

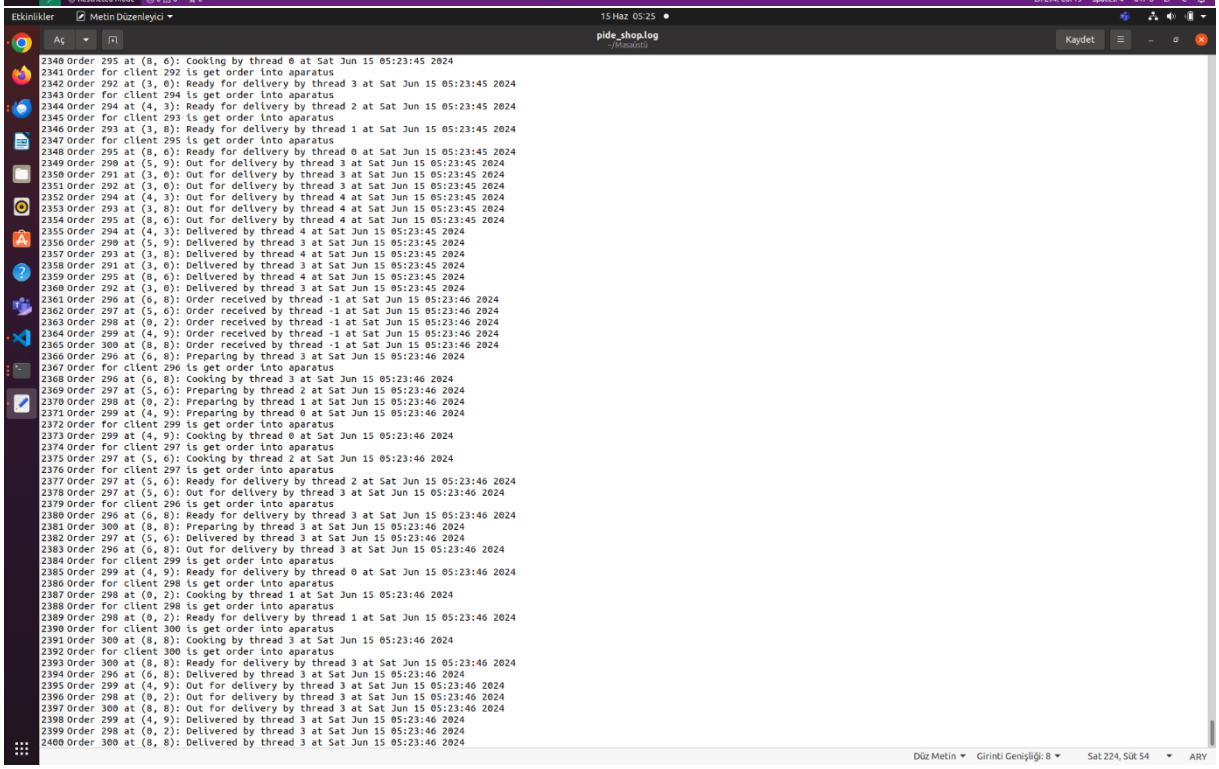




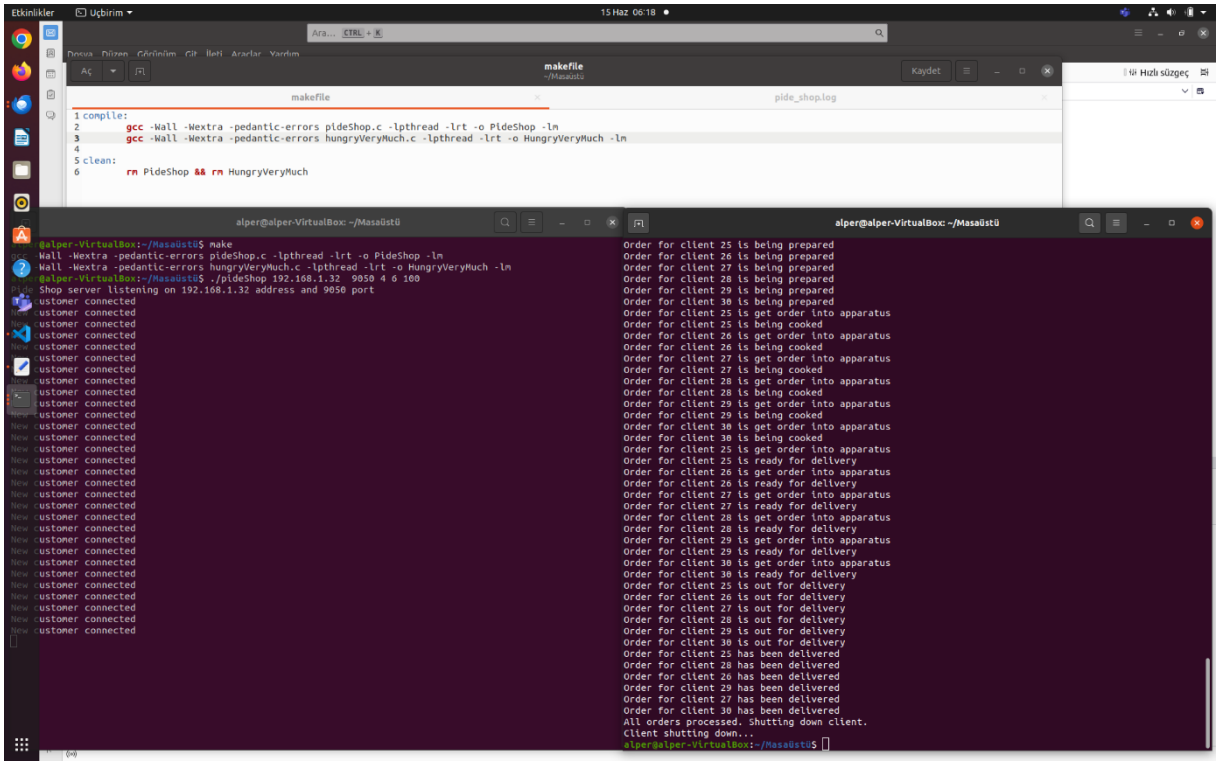
10)valgrind(Two Client.c)

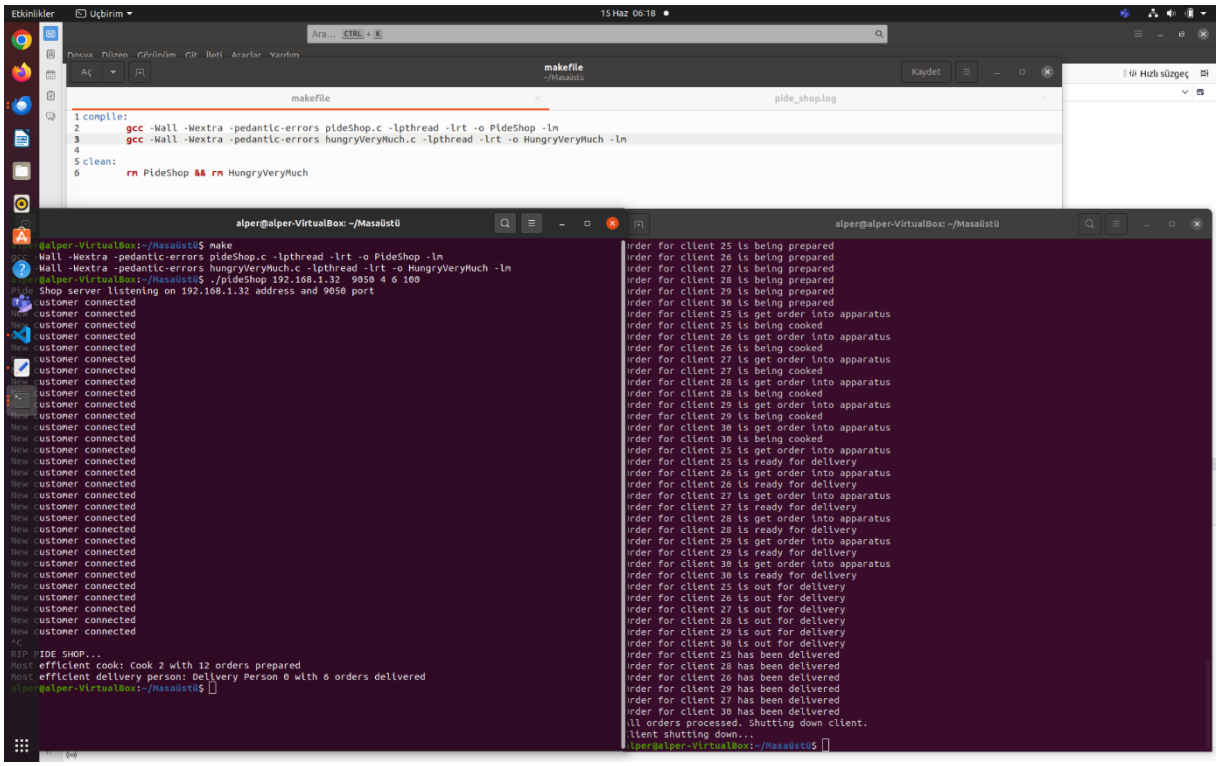



Etkinlikler
Uçbirim
15 Haz 05:24
alper@alper-VirtualBox: ~/Masaüstü
==13287== by 0x10B128: print_most_efficient_workers (in /home/alper/MasaüstÜ/server)
==13287== by 0x10969A: signal_handler (in /home/alper/MasaüstÜ/server)
==13287== by 0x4A1908F: ??? (in /usr/lib/x86_64-linux-gnu/libc-2.31.so)
==13287== by 0x49C64FC: accept (accept.c:26)
==13287== by 0x109AC6: main (in /home/alper/MasaüstÜ/server)
==13287==
==13287== Conditional jump or move depends on uninitialised value(s)
==13287== at 0x4A4E228: __vfprintf_internal (vfprintf-internal.c:1687)
==13287== by 0x4A37D3E: printf (printf.c:33)
==13287== by 0x10B128: print_most_efficient_workers (in /home/alper/MasaüstÜ/server)
==13287== by 0x10969A: signal_handler (in /home/alper/MasaüstÜ/server)
==13287== by 0x4A1908F: ??? (in /usr/lib/x86_64-linux-gnu/libc-2.31.so)
==13287== by 0x49C64FC: accept (accept.c:26)
==13287== by 0x109AC6: main (in /home/alper/MasaüstÜ/server)
==13287==
==13287== Conditional jump or move depends on uninitialised value(s)
==13287== at 0x4A4D6EE: __vfprintf_internal (vfprintf-internal.c:1687)
==13287== by 0x4A37D3E: printf (printf.c:33)
==13287== by 0x10B128: print_most_efficient_workers (in /home/alper/MasaüstÜ/server)
==13287== by 0x10969A: signal_handler (in /home/alper/MasaüstÜ/server)
==13287== by 0x4A1908F: ??? (in /usr/lib/x86_64-linux-gnu/libc-2.31.so)
==13287== by 0x49C64FC: accept (accept.c:26)
==13287== by 0x109AC6: main (in /home/alper/MasaüstÜ/server)
==13287==
Most efficient delivery person: Delivery Person 0 with 106 orders delivered
==13287==
==13287== HEAP SUMMARY:
==13287== in use at exit: 3,136 bytes in 12 blocks
==13287== total heap usage: 1,824 allocs, 1,812 frees, 41,591 bytes allocated
==13287==
==13287== 1,088 bytes in 4 blocks are possibly lost in loss record 3 of 4
==13287== at 0x483DD99: calloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==13287== by 0x40149DA: allocate_dtv (dl-tls.c:286)
==13287== by 0x40149DA: _dl_allocate_tls (dl-tls.c:532)
==13287== by 0x49BC322: allocate_stack (allocatestack.c:622)
==13287== by 0x49BC322: pthread_create@@GLIBC_2.2.5 (pthread_create.c:660)
==13287== by 0x109890: main (in /home/alper/MasaüstÜ/server)
==13287==
==13287== 1,632 bytes in 6 blocks are possibly lost in loss record 4 of 4
==13287== at 0x483DD99: calloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==13287== by 0x40149DA: allocate_dtv (dl-tls.c:286)
==13287== by 0x40149DA: _dl_allocate_tls (dl-tls.c:532)
==13287== by 0x49BC322: allocate_stack (allocatestack.c:622)
==13287== by 0x49BC322: pthread_create@@GLIBC_2.2.5 (pthread_create.c:660)
==13287== by 0x109965: main (in /home/alper/MasaüstÜ/server)
==13287==
==13287== LEAK SUMMARY:
==13287== definitely lost: 0 bytes in 0 blocks
==13287== indirectly lost: 0 bytes in 0 blocks
==13287== possibly lost: 2,720 bytes in 10 blocks
==13287== still reachable: 416 bytes in 2 blocks
==13287== suppressed: 0 bytes in 0 blocks
==13287== Reachable blocks (those to which a pointer was found) are not shown.
==13287== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==13287==
==13287== Use --track-origins=yes to see where uninitialised values come from
==13287== For lists of detected and suppressed errors, rerun with: -s
==13287== ERROR SUMMARY: 28 errors from 14 contexts (suppressed: 0 from 0)
alper@alper-VirtualBox:~/MasaüstÜ$
alper@alper-VirtualBox: ~/Masaüstü
or client 284 has been delivered
or client 285 has been delivered
or client 286 has been delivered
or client 287 has been delivered
or client 288 has been delivered
or client 289 has been delivered
or client 290 has been delivered
or client 291 has been delivered
or client 292 has been delivered
or client 293 has been delivered
or client 294 has been delivered
or client 295 has been delivered
or client 298 is being prepared
or client 299 is being prepared
or client 299 is get order into apparatus
or client 299 is being cooked
or client 297 is get order into apparatus
or client 297 is being cooked
or client 297 is get order into apparatus
or client 297 is ready for delivery
or client 297 is out for delivery
or client 296 is get order into apparatus
or client 296 is ready for delivery
or client 300 is being prepared
or client 297 has been delivered
or client 296 is out for delivery
or client 299 is get order into apparatus
or client 299 is ready for delivery
or client 298 is get order into apparatus
or client 298 is being cooked
or client 298 is get order into apparatus
or client 298 is ready for delivery
or client 300 is get order into apparatus
or client 300 is being cooked
or client 300 is get order into apparatus
or client 300 is ready for delivery
or client 296 has been delivered
or client 299 is out for delivery
or client 298 is out for delivery
or client 300 is out for delivery
or client 298 has been delivered
or client 299 has been delivered
or client 300 has been delivered
ers processed. Shutting down client.
shutting down...
==
== HEAP SUMMARY:
== in use at exit: 0 bytes in 0 blocks
== total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==
== All heap blocks were freed -- no leaks are possible
==
== For lists of detected and suppressed errors, rerun with: -s
== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
lper-VirtualBox:~/MasaüstÜ$
bytes allocated
==13081== All heap blocks were freed -- no leaks are possible
==13081==
==13081== For lists of detected and suppressed errors, rerun with: -s
==13081== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
alper@alper-VirtualBox:~/MasaüstÜ$
Restricted Mode
Ln 294, Col 19
Spaces: 4
UTF-8
LF
C


Etkinlikler
Metin Düzenleyici
15 Haz 05:25
Aç
pide_shop.log
~/Masaüstü
Kaydet
2340 Order 295 at (8, 6): Cooking by thread 0 at Sat Jun 15 05:23:45 2024
2341 Order for client 292 is get order into aparatus
2342 Order 292 at (3, 0): Ready for dellvery by thread 3 at Sat Jun 15 05:23:45 2024
2343 Order for client 294 is get order into aparatus
2344 Order 294 at (4, 3): Ready for dellvery by thread 2 at Sat Jun 15 05:23:45 2024
2345 Order for client 293 is get order into aparatus
2346 Order 293 at (3, 8): Ready for dellvery by thread 1 at Sat Jun 15 05:23:45 2024
2347 Order for client 295 is get order into aparatus
2348 Order 295 at (8, 6): Ready for dellvery by thread 0 at Sat Jun 15 05:23:45 2024
2349 Order 290 at (5, 9): Out for dellvery by thread 3 at Sat Jun 15 05:23:45 2024
2350 Order 291 at (3, 0): Out for dellvery by thread 3 at Sat Jun 15 05:23:45 2024
2351 Order 292 at (3, 0): Out for dellvery by thread 3 at Sat Jun 15 05:23:45 2024
2352 Order 294 at (4, 3): Out for dellvery by thread 4 at Sat Jun 15 05:23:45 2024
2353 Order 293 at (3, 8): Out for dellvery by thread 4 at Sat Jun 15 05:23:45 2024
2354 Order 295 at (8, 6): Out for dellvery by thread 4 at Sat Jun 15 05:23:45 2024
2355 Order 294 at (4, 3): Dellvered by thread 4 at Sat Jun 15 05:23:45 2024
2356 Order 290 at (5, 9): Dellvered by thread 3 at Sat Jun 15 05:23:45 2024
2357 Order 293 at (3, 8): Dellvered by thread 4 at Sat Jun 15 05:23:45 2024
2358 Order 291 at (3, 0): Dellvered by thread 3 at Sat Jun 15 05:23:45 2024
2359 Order 295 at (8, 6): Dellvered by thread 4 at Sat Jun 15 05:23:45 2024
2360 Order 292 at (3, 0): Dellvered by thread 3 at Sat Jun 15 05:23:45 2024
2361 Order 296 at (6, 8): Order received by thread -1 at Sat Jun 15 05:23:46 2024
2362 Order 297 at (5, 6): Order received by thread -1 at Sat Jun 15 05:23:46 2024
2363 Order 298 at (0, 2): Order received by thread -1 at Sat Jun 15 05:23:46 2024
2364 Order 299 at (4, 9): Order received by thread -1 at Sat Jun 15 05:23:46 2024
2365 Order 300 at (8, 8): Order received by thread -1 at Sat Jun 15 05:23:46 2024
2366 Order 296 at (6, 8): Preparing by thread 3 at Sat Jun 15 05:23:46 2024
2367 Order for client 296 is get order into aparatus
2368 Order 296 at (6, 8): Cooking by thread 3 at Sat Jun 15 05:23:46 2024
2369 Order 297 at (5, 6): Preparing by thread 2 at Sat Jun 15 05:23:46 2024
2370 Order 298 at (0, 2): Preparing by thread 1 at Sat Jun 15 05:23:46 2024
2371 Order 299 at (4, 9): Preparing by thread 0 at Sat Jun 15 05:23:46 2024
2372 Order for client 299 is get order into aparatus
2373 Order 299 at (4, 9): Cooking by thread 0 at Sat Jun 15 05:23:46 2024
2374 Order for client 297 is get order into aparatus
2375 Order 297 at (5, 6): Cooking by thread 2 at Sat Jun 15 05:23:46 2024
2376 Order for client 297 is get order into aparatus
2377 Order 297 at (5, 6): Ready for delivery by thread 2 at Sat Jun 15 05:23:46 2024
2378 Order 297 at (5, 6): Out for delivery by thread 3 at Sat Jun 15 05:23:46 2024
2379 Order for client 296 is get order into aparatus
2380 Order 296 at (6, 8): Ready for delivery by thread 3 at Sat Jun 15 05:23:46 2024
2381 Order 300 at (8, 8): Preparing by thread 3 at Sat Jun 15 05:23:46 2024
2382 Order 297 at (5, 6): Delivered by thread 3 at Sat Jun 15 05:23:46 2024
2383 Order 296 at (6, 8): Out for delivery by thread 3 at Sat Jun 15 05:23:46 2024
2384 Order for client 299 is get order into aparatus
2385 Order 299 at (4, 9): Ready for delivery by thread 0 at Sat Jun 15 05:23:46 2024
2386 Order for client 298 is get order into aparatus
2387 Order 298 at (0, 2): Cooking by thread 1 at Sat Jun 15 05:23:46 2024
2388 Order for client 298 is get order into aparatus
2389 Order 298 at (0, 2): Ready for delivery by thread 1 at Sat Jun 15 05:23:46 2024
2390 Order for client 300 is get order into aparatus
2391 Order 300 at (8, 8): Cooking by thread 3 at Sat Jun 15 05:23:46 2024
2392 Order for client 300 is get order into aparatus
2393 Order 300 at (8, 8): Ready for delivery by thread 3 at Sat Jun 15 05:23:46 2024
2394 Order 296 at (6, 8): Dellvered by thread 3 at Sat Jun 15 05:23:46 2024
2395 Order 299 at (4, 9): Out for delivery by thread 3 at Sat Jun 15 05:23:46 2024
2396 Order 298 at (0, 2): Out for delivery by thread 3 at Sat Jun 15 05:23:46 2024
2397 Order 300 at (8, 8): Out for delivery by thread 3 at Sat Jun 15 05:23:46 2024
2398 Order 299 at (4, 9): Delivered by thread 3 at Sat Jun 15 05:23:46 2024
2399 Order 298 at (0, 2): Delivered by thread 3 at Sat Jun 15 05:23:46 2024
2400 Order 300 at (8, 8): Delivered by thread 3 at Sat Jun 15 05:23:46 2024
Düz Metin
Girinti Genişliği: 8
Sat 224, Süt 54
ARY

MAKEFILE :


makefile
compile:
gcc -Wall -Wextra -pedantic-errors pideShop.c -lpthread -lrt -o PideShop -lm
gcc -Wall -Wextra -pedantic-errors hungryVeryMuch.c -lpthread -lrt -o HungryVeryMuch -lm
clean:
rm PideShop && rm HungryVeryMuch
alper@alper-VirtualBox: ~/Masaüstü
alper@alper-VirtualBox:~/Masaüstü$ make
alper@alper-VirtualBox:~/Masaüstü$ ./pideShop 192.168.1.32 9050 4 6 100
Shop server listening on 192.168.1.32 address and 9050 port
New customer connected
Order for client 25 is being prepared
Order for client 26 is being prepared
Order for client 27 is being prepared
Order for client 28 is being prepared
Order for client 29 is being prepared
Order for client 30 is being prepared
Order for client 25 is get order into apparatus
Order for client 25 is being cooked
Order for client 26 is get order into apparatus
Order for client 26 is being cooked
Order for client 27 is get order into apparatus
Order for client 27 is being cooked
Order for client 28 is get order into apparatus
Order for client 28 is being cooked
Order for client 29 is get order into apparatus
Order for client 29 is being cooked
Order for client 30 is get order into apparatus
Order for client 30 is being cooked
Order for client 25 is get order into apparatus
Order for client 25 is ready for delivery
Order for client 26 is get order into apparatus
Order for client 26 is ready for delivery
Order for client 27 is get order into apparatus
Order for client 27 is ready for delivery
Order for client 28 is get order into apparatus
Order for client 28 is ready for delivery
Order for client 29 is get order into apparatus
Order for client 29 is ready for delivery
Order for client 30 is get order into apparatus
Order for client 30 is ready for delivery
Order for client 25 is out for delivery
Order for client 26 is out for delivery
Order for client 27 is out for delivery
Order for client 28 is out for delivery
Order for client 29 is out for delivery
Order for client 30 is out for delivery
Order for client 25 has been delivered
Order for client 28 has been delivered
Order for client 26 has been delivered
Order for client 29 has been delivered
Order for client 27 has been delivered
Order for client 30 has been delivered
All orders processed. Shutting down client.
Client shutting down...
alper@alper-VirtualBox:~/Masaüstü$


makefile
compile:
gcc -Wall -Wextra -pedantic-errors pideShop.c -lpthread -lrt -o PideShop -lm
gcc -Wall -Wextra -pedantic-errors hungryVeryMuch.c -lpthread -lrt -o HungryVeryMuch -lm
clean:
rm PideShop && rm HungryVeryMuch
alper@alper-VirtualBox: ~/Masaüstü
alper@alper-VirtualBox:~/Masaüstü$ make
alper@alper-VirtualBox:~/Masaüstü$ ./pideShop 192.168.1.32 9050 4 6 100
Shop server listening on 192.168.1.32 address and 9050 port
New customer connected
^C
RIP PIDE SHOP...
Most efficient cook: Cook 2 with 12 orders prepared
Most efficient delivery person: Delivery Person 0 with 6 orders delivered
alper@alper-VirtualBox:~/Masaüstü$

NOTE!!

- There is a issue at order cancelation but it is safe because it is about the signal received but it is already in process .Some of them in out of delivery ones are delivered but ı write again about Canceled information. If the order canceled before Delivered status , ı mark them and write them to terminal and log file also .
- You need to write local host(it is dynamic) or static IP address.
- If you want to do it slower or harder ı was doing sleep(calc..) ,usleep(calc..) . You can change it it Works both of them but usleep is sof ast to see the differences . sleep with 100 speed is great to test .I am gonna send it with usleep because it is going to be so long for 200 250 or more.
- I use SO_REUSEPORT because of the port availability . I prefer SO_REUSEPORT but some of students said it is not working on Prof. Erkan Hoca's computer .. 😊
- I hope you are going to like my Reports .