To Run Program:

Make compile -> for compile

Make server => runs server program

Make client => runs client program

Make clean -> log and exe files will be cleaned

Connection between Server and Client

In this project, the communication between the server and client is established using sockets, a fundamental concept in network programming. Sockets provide a way for programs to communicate with each other, either on the same machine or across different machines over a network.

Overview of Socket Communication

A socket is an endpoint for sending or receiving data across a computer network. It serves as an interface for network communication, providing a way for two devices to establish a connection and exchange data. There are two main types of sockets:

1. **Stream Sockets (TCP):** These provide reliable, two-way, connection-based byte streams. They ensure that data is delivered in the same order it was sent and without duplication.
2. **Datagram Sockets (UDP):** These provide connectionless, unreliable communication. Data is sent as discrete packets, and there is no guarantee of order or delivery.

Implementation in the Project

In this project, we use stream sockets (TCP) for communication between the server and client. TCP is chosen because it provides reliable, ordered, and error-checked delivery of a stream of bytes, making it suitable for applications where data integrity and order are crucial.

Server-Side Implementation

Socket Creation:

```
int server_fd = socket(AF_INET, SOCK_STREAM, 0);
if (server_fd == 0) {
    perror("Socket failed");
    exit(EXIT_FAILURE);
}
```

Binding:

```
struct sockaddr_in address;
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons(portnum);

if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
    perror("Bind failed");
    close(server_fd);
    exit(EXIT_FAILURE);
}
```

Listening:

```
if (listen(server_fd, 3) < 0) {
    perror("Listen failed");
    close(server_fd);
    exit(EXIT_FAILURE);
}
```

Accepting Connections:

```
int new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t*)&addrlen);
if (new_socket < 0) {
    perror("Accept failed");
    continue;
}
```

Client-Side Implementation

Socket Creation:

```
int client_socket = socket(AF_INET, SOCK_STREAM, 0);
if (client_socket < 0) {
    printf("Socket creation error\n");
    return -1;
}
```

Connecting to the Server:

```c
struct sockaddr_in server_addr;
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(portnum);

if (inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr) <= 0) {
    printf("Invalid address/ Address not supported\n");
    return -1;
}

if (connect(client_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
    printf("Connection Failed\n");
    return -1;
}
```

SYNC

In this code, mutex (mutual exclusion) and condition variables are used to ensure data consistency and synchronization between chef and deliverer threads. Several mutex and condition variables are defined to prevent data races (race conditions) when accessing resources to be shared between threads.

Mutexler:

1. ordersMutex: Used to synchronize access and updates to orders. This mutex is used when chefs access the order list to prepare a new order or delivery people to retrieve an order waiting to be delivered.

2. ovenSlotsMutex: Used to control the availability of oven slots. This mutex is used when chefs are looking for a slot to put the order into the oven or when the slot is released after cooking is complete.

3. ovenPeelsMutex: Used to check the usability of the oven spatula. This mutex is used when chefs use the spatula to place or remove the order from the oven.

4. ovenMutex: Used to manage oven capacity. This mutex is used when checking how many orders are cooked in the oven.

5. logMutex: Used to synchronize write operations to the log file. This mutex is used to ensure data integrity when multiple threads try to write to the log file simultaneously.

Condition Variables:

1. orderAvailable: Used to wake up the chefs when a new order arrives. When orders decrease or new orders arrive, chef threads are informed with this condition variable.

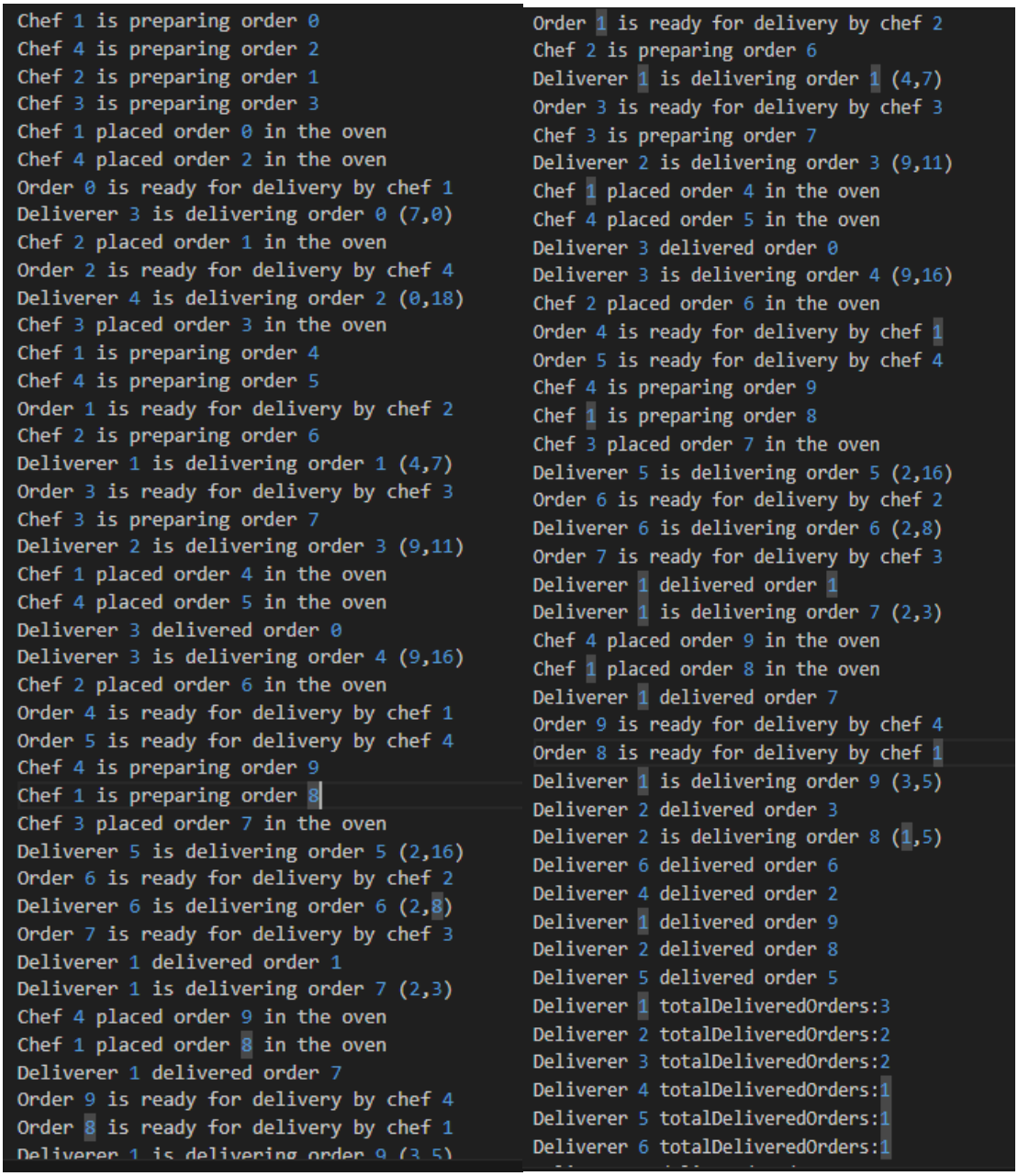
2. ovenPeelAvailable: Used to wake up waiting chefs when the spatula is released.

3. ovenAvailable: Used to wake up the waiting chefs when space becomes available in the oven.

4. ovenSlotAvailable: Used to wake up waiting chefs when the oven slot is released.

Thanks to these synchronization mechanisms, the preparation, cooking and delivery of orders between chefs and delivery people are carried out securely and data consistency is maintained. This also ensures that threads in the system run efficiently and prevents potential deadlocks.

Output logfile:



```
Chef 1 is preparing order 0
Chef 4 is preparing order 2
Chef 2 is preparing order 1
Chef 3 is preparing order 3
Chef 1 placed order 0 in the oven
Chef 4 placed order 2 in the oven
Order 0 is ready for delivery by chef 1
Deliverer 3 is delivering order 0 (7,0)
Chef 2 placed order 1 in the oven
Order 2 is ready for delivery by chef 4
Deliverer 4 is delivering order 2 (0,18)
Chef 3 placed order 3 in the oven
Chef 1 is preparing order 4
Chef 4 is preparing order 5
Order 1 is ready for delivery by chef 2
Chef 2 is preparing order 6
Deliverer 1 is delivering order 1 (4,7)
Order 3 is ready for delivery by chef 3
Chef 3 is preparing order 7
Deliverer 2 is delivering order 3 (9,11)
Chef 1 placed order 4 in the oven
Chef 4 placed order 5 in the oven
Deliverer 3 delivered order 0
Deliverer 3 is delivering order 4 (9,16)
Chef 2 placed order 6 in the oven
Order 4 is ready for delivery by chef 1
Order 5 is ready for delivery by chef 4
Chef 4 is preparing order 9
Chef 1 is preparing order 8
Chef 3 placed order 7 in the oven
Deliverer 5 is delivering order 5 (2,16)
Order 6 is ready for delivery by chef 2
Deliverer 6 is delivering order 6 (2,8)
Order 7 is ready for delivery by chef 3
Deliverer 1 delivered order 1
Deliverer 1 is delivering order 7 (2,3)
Chef 4 placed order 9 in the oven
Chef 1 placed order 8 in the oven
Deliverer 1 delivered order 7
Order 9 is ready for delivery by chef 4
Order 8 is ready for delivery by chef 1
Deliverer 1 is delivering order 9 (3,5)
Deliverer 2 delivered order 3
Deliverer 2 is delivering order 8 (1,5)
Deliverer 6 delivered order 6
Deliverer 4 delivered order 2
Deliverer 1 delivered order 9
Deliverer 2 delivered order 8
Deliverer 5 delivered order 5
Deliverer 1 totalDeliveredOrders:3
Deliverer 2 totalDeliveredOrders:2
Deliverer 3 totalDeliveredOrders:2
Deliverer 4 totalDeliveredOrders:1
Deliverer 5 totalDeliveredOrders:1
Deliverer 6 totalDeliveredOrders:1
```