CSE-344 SYSTEM PROGRAMING FINAL PROJECT REPORT
MUHAMMED ALPEREN KARAÇETE 171044052

Usage:
make compiles client.c and server.c
make clean deletes .log file .o files.

For server → ./PideShop 8080 4 6 10000: 8080 is port, 4 cook count and 6 delivery count. 10000 is speed of deliverers.

For client → ./HungryVeryMuch 8080 100 15 15: 8080 is port, 100 client count, 15 x coordinate and 15 y coordinate.

System Architecture:
Firstly I used functions for finding pseudo inverse of a 30 by 40 matrix having complex elements. These are:

```
typedef struct {
   double real;
   double imag;
} complex;

typedef struct {
   complex **data;
   int rows;
   int cols;
} matrix;
```

`allocateMatrix`

- **Purpose**: Allocates memory for a matrix structure and initializes its dimensions (`rows` and `cols`).

`freeMatrix`

- **Purpose**: Frees the memory allocated for a matrix structure.

`conjugateTranspose`

- **Purpose**: Computes the conjugate transpose of matrix `A` and stores it in matrix `A_conj_trans`.

`matrixMultiply`

- **Purpose**: Computes the matrix multiplication of matrices `A` and `B` and stores the result in matrix `result`.

`luDecomp`

- **Purpose**: Performs LU decomposition of matrix `A` and stores the lower triangular matrix `L` and upper triangular matrix `U`.

`luSolve`

- **Purpose**: Solves a linear system using LU decomposition.

`pseudoInverse`

- **Purpose**: Computes the pseudo-inverse of matrix `A` and stores it in matrix `A_pinv`.

`fillMatrixRandom`

- **Purpose**: Fills a matrix with random complex numbers.

My pide structure:

```
typedef struct {
    int id;
    int clientSocketFd;
    char status[20];
    int x;
    int y;
} Pide;
```

This is my main structure. Hole project is depend on this structure. I hide every client's fd,id,status,x and y coordinates on this structure.

My server information structure:

```
struct serverClientStruct {
    int serverFd;
    struct sockaddr_in serverInfo;
    struct sockaddr_in clientInfo;
} information;
```

This structure for letting manager thread to use server socket for communicating with clients.

My queues

```
Pide *pide_queue;
Pide *cooked_queue;
Pide *delivery_queue;
```

These are my queues . Manager thread uese pide queue. Fills required informations about pide like clientSocketFd for communicating, x and y coordinates for delivering and id.Status information changes on every stage.

Global Variables:

int totalPides; keeps total pide count.
int availableCook; keeps total cook count.
int availableDeliveryPersonal; keeps total delivery count.
Int totalDelivered; keeps total delivered pide count.
int townx; keeps x limit coordinate of town center
int towny; keeps y limit coordinate of town center
int deliverySpeed; keeps speed of delivery
int logFd; keeps file descriptor of log file
int *cookPlace; keep which cook cooked how many pide.
int *deliveryPlace; keep which delivery delivered how many pide.
int oven_spots = 6;
int oven_aparatus = 3;
int oven_openings = 2;

Signal Handlers:

```
void sigIntHandler(int sig) {
    exitFlag = 1;
    pthread_cond_broadcast(&delivery_ready);
    pthread_cond_broadcast(&pide_ready);
    pthread_cond_broadcast(&finish);
}
```

exitFlag for ending all ongoing loops.
Broadcasts for let exiting all waiting threads on cond variables.

My threads:

Manager Thread:

This thread function is responsible for coordinating the entire process of taking orders from clients, assigning cooks and delivery personnel, managing queues, logging operations, and communicating with customers.

Reads the initial customerCount from the first client, which indicates how many customers (and therefore, how many pides) are expected to be served.
Reads clientPid (client process ID)
And x, y coordinates representing the town's geographical location from the client.

Then it reads orders and places of customers. It assign them to pide queue and deliver it to cook thread for letting them cook.
Communicates with the client by sending acknowledgment messages like "Order Placed" after successfully placing an order.

When all customers have been served (customerCount == 0), it waits for all pending operations (cooking and delivery) to finish (pthread_cond_wait(&finish, &cooked_queue_lock)).

Calculates and logs the best-performing cook and delivery personnel based on the number of tasks completed.

Resets internal counters and frees memory allocated for queues (pide_queue, cooked_queue, delivery_queue) to prepare for the next batch of orders.

When exit flag received from ctrl + c it breaks the loop and exits.

Cook Thread:

Extracts cook_id from the passed argument and immediately frees the allocated memory for arg for other cook threads.

Allocates two matrices (mat and matPinv) using a custom function allocateMatrix and fills mat with random data using fillMatrixRandom. This for calculating pseudo inverse.

Enters a loop that continues until exitFlag becomes true (indicating the program should exit when user press ctrl + c).

Acquires the pide_queue_lock mutex to safely access and modify the pide_queue.

Waiting for Pides:

Uses a condition variable pide_ready to wait until there is at least one pide order (pide_count > 0) in the pide_queue. Manager thread insert them in pide queue and send signal when it insert it one.

If exitFlag becomes true during this waiting period, it releases the lock and breaks out of the loop to exit the thread.

Processing Pide Order:

Once a pide order is available, dequeues (--pide_count) the pide from pide_queue.

Then it add it to cooked queue for usage on delivery thread.

Calculates the pseudo-inverse of matrix mat using pseudoInverse function and stores it in matPinv.Additionally hides the time passed on executing pseudoInverse function for sleeping on cooking half of the that time.

Updates the status of the pide to "Prepared", logs this action, and notifies the client that their order is prepared.

Updates statistics (cookPlace[cook_id]) to track how many pides each cook has prepared for promoting hardworking cook.

Handling Oven Capacity,aparatus and openings:

Acquires the cooked_queue_lock mutex to manage access to the cooked_queue and oven_spots (available oven spots).

Uses a condition variable delivery_ready to wait until there is an available spot in the oven (oven_spots > 0), there is a oven aparatus available or one of the oven openning is available.

If exitFlag becomes true during this waiting period, it releases the lock and breaks out of the loop to exit the thread.

Cooking:

Simulates the cooking process by decrementing oven_spots (assuming each cook uses one spot).

On cooking decreases oven_spots,oven_aparat,oven_openings.

Sleeps for half of the time it took to compute the pseudo-inverse (executionTime / 2) to simulate the time needed to cook the pide.

After cooking, increments oven_spots,oven_aparat,oven_openings, updates the pide status to "Cooked", logs this action, and notifies the client that their order is cooked.

Signals the delivery_ready condition variable to indicate that a cooked pide is ready for delivery.

Delivery Thread:

Extracts delivery_id from the passed argument and immediately frees the allocated memory for other delivery threads

Initializes a temporary buffer temp to store log messages.

Execution Loop:

Enters a loop that continues until exitFlag becomes true (indicating the program should exit on ctrl + c).

Waiting for Cooked Pides:

Acquires the cooked_queue_lock mutex to safely access and modify the cooked_queue.

Uses a condition variable delivery_ready to wait until there are cooked pides available for delivery (cooked_count > 0).

If exitFlag becomes true during this waiting period, it releases the lock and breaks out of the loop to exit the thread.

Processing Delivery Orders:

When there are cooked pides in the cooked_queue, dequeues (--cooked_count) up to 3 pides for delivery.

Updates the status of each pide to "Delivered", logs this action, sends a delivery confirmation message to the client, and closes the client socket.

Updates statistics (deliveryPlace[delivery_id]) to track how many deliveries each delivery thread has completed.

If fewer than 3 pides are available for delivery but the total number of pides minus those delivered is less than 3 (max_pides - totalDelivered < 3), signals delivery_ready to continue cooking more pides.

Simulating Delivery:

Calculates the distance between the delivery location (p.x, p.y) and the town center (townx/2, towny/2) using the Euclidean distance formula.

Sleeps for a simulated delivery time based on the distance and deliverySpeed.

Handling Remaining Pides:

If there are still cooked pides left in cooked_queue after delivering up to 3 pides, immediately signaling delivery_ready for delivering this orders too. Because there will be no more orders and deliverers should not be waiting for their pides to become 3.

I have thread pool for cook and delivery threads. This threads do not exit until program exits or interrupted by a signal. They always wait for manager’s orders. In main I am getting them from user as argument.

I have used mutexes for preventing race conditions and data corruptions.

max_pides - totalDelivered == 0 This means all deliviries has been delivered and order can be done by manager.

Main Function:

Opens or creates a log file named "server.log" with appropriate permissions (O_CREAT | O_WRONLY | O_TRUNC, S_IRUSR | S_IWUSR).

Checks if the file descriptor (logFileFd) is valid and exits with an error message if not.

Sets up a signal handler (sigIntHandler) for handling the SIGINT signal.

Gets command line arguments from user for fullfilling arguments.

Validates that each of the arguments (argv[1] to argv[4]) can be converted to integers (portnumber, CookthreadPoolSize, DeliveryPoolSize, k). If any conversion fails, prints an error message and exits.

Uses socket() to create a TCP socket (AF_INET, SOCK_STREAM).

Configures the server socket (serverSocketFd) with address (serverInfo) and port number.

Binds the socket to the server address. If binding fails, prints an error, closes the socket, and exits.

Sets the socket to listen for incoming connections (listen()).

Thread Initialization:

Manager Thread:

Creates a manager thread (pthread_create) to handle incoming client connections. Passes serverSocketFd as an argument (managerThreadFunc).

Cook Threads:

Creates an array of cook threads (cookThreads) based on CookthreadPoolSize.

Each cook thread is created with a unique cookId, which identifies its position in the thread pool (pthread_create).

Delivery Threads:

Creates an array of delivery threads (deliveryThreads) based on DeliveryPoolSize.

Each delivery thread is created with a unique deliveryId (pthread_create).

Waits for the manager thread and all cook and delivery threads to finish execution (pthread_join).

Frees dynamically allocated memory used for thread IDs (cookId, deliveryId), thread arrays (cookThreads, deliveryThreads), and other data structures (pide_queue, cooked_queue, delivery_queue, deliveryPlace, cookPlace).

Closes the log file descriptor (logFileFd) and the server socket (serverSocketFd).

Client Generator:

Client Struct (client_info_t):

Holds client information: clientId, hostname, port, x, y, clientCount, townx, towny. And pass them to handle_client thread.

handle_client Function:

Thread function for each client:

Creates a socket and connects to the server.

Sends client details to the server (for the first client).

Sends client position to the server.

Then receives and prints status messages from the server for 4 times because of 4 status type.

Main Function:

Parses command-line arguments (port, number of clients, town dimensions).

Spawns threads for each client using pthread_create.

Waits for all client threads to finish with pthread_join.

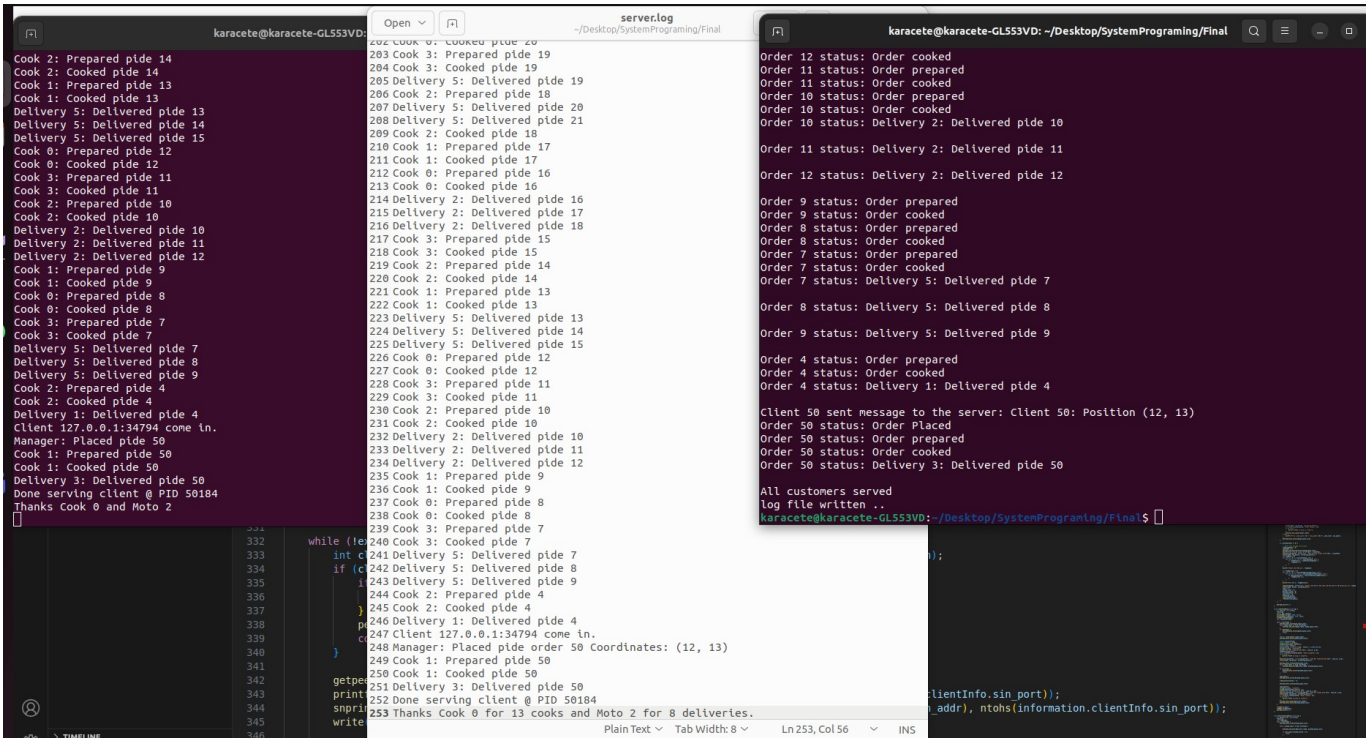
Mutexes: Ensures thread-safe access to shared variables (clientNo, deliverPide).

Command-Line Arguments: Validates and processes input parameters.

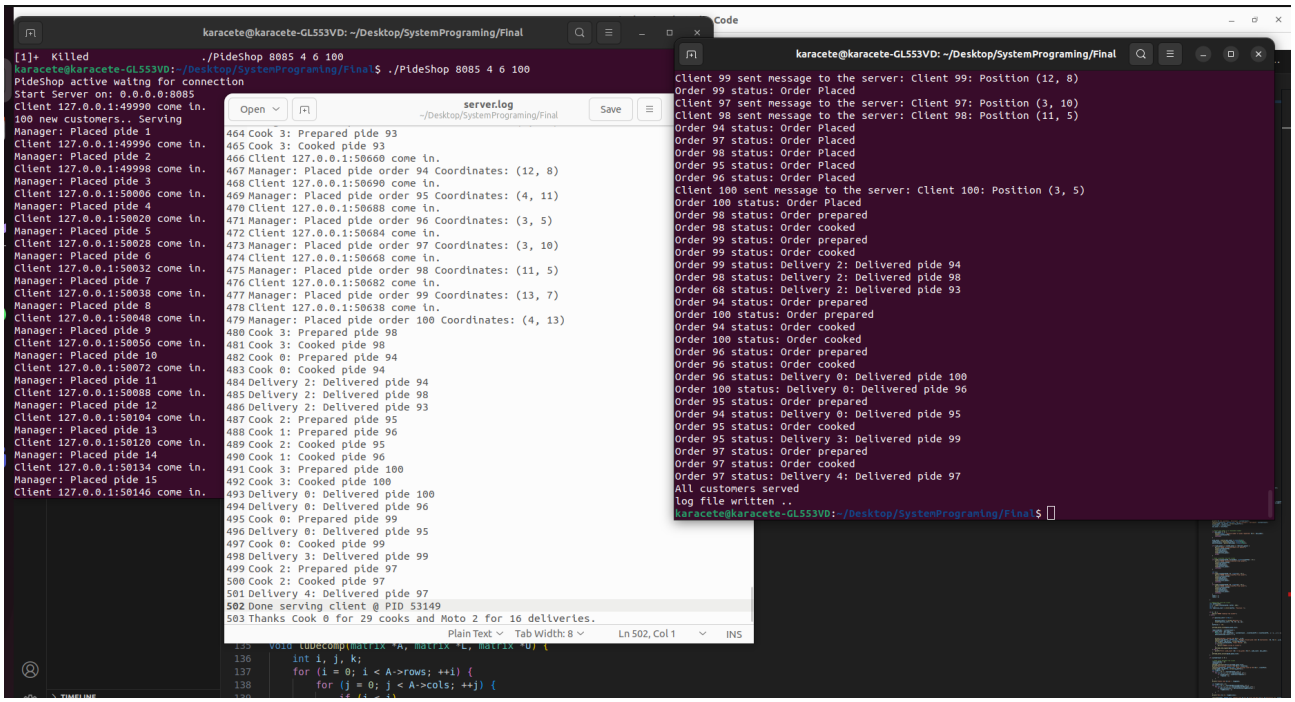
Randomization: Generates random client positions within specified town dimensions and spawn new clients..

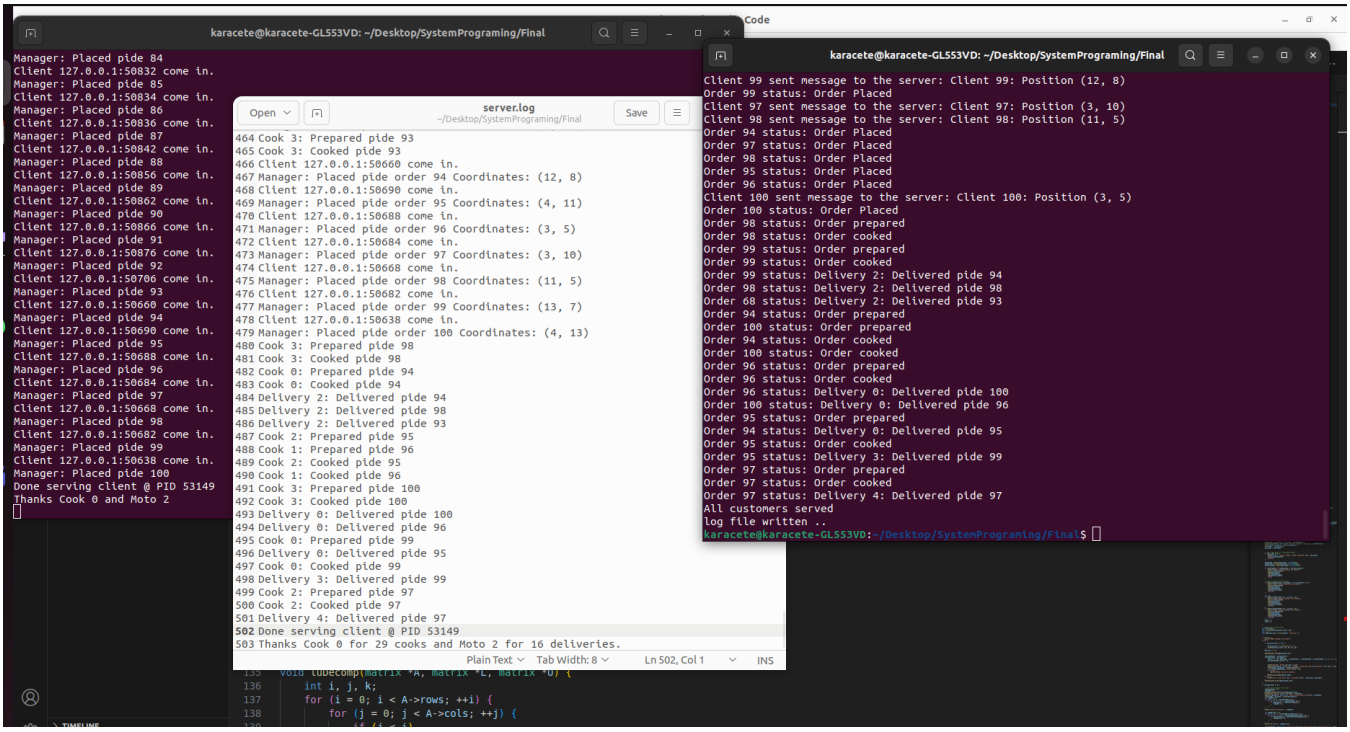
Test Cases:

For 50 Orders:

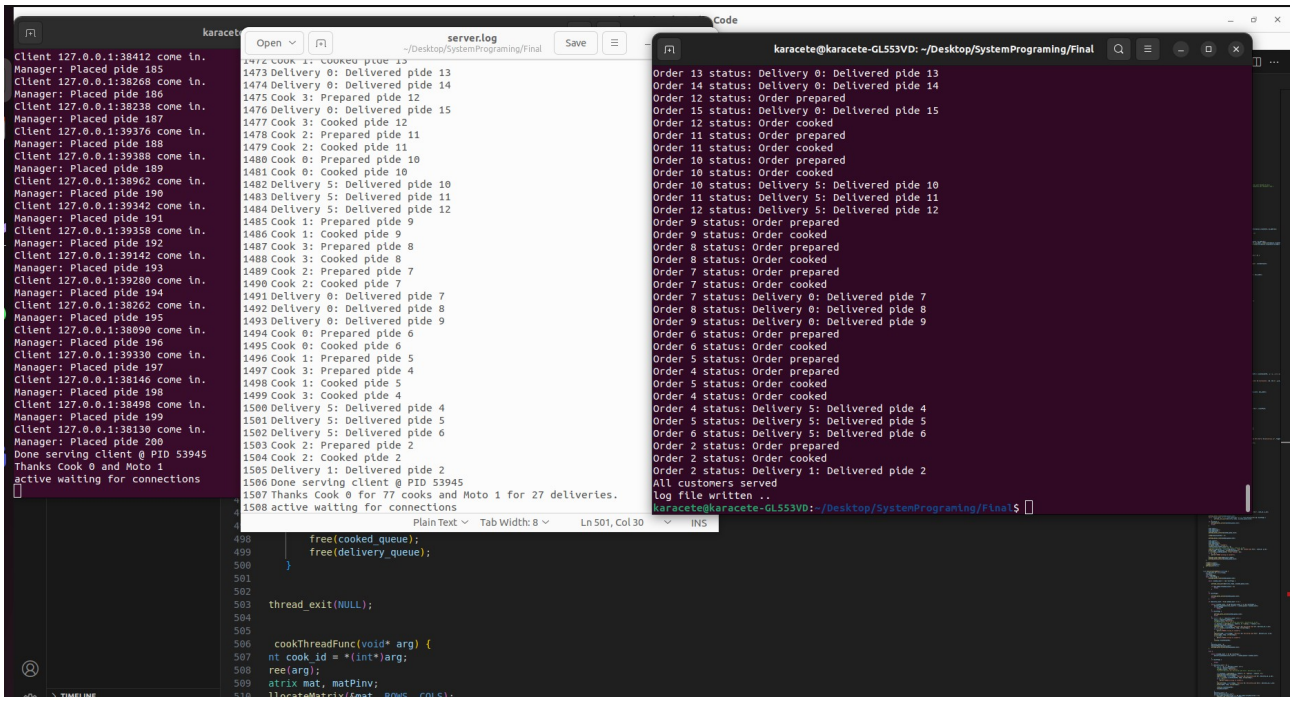


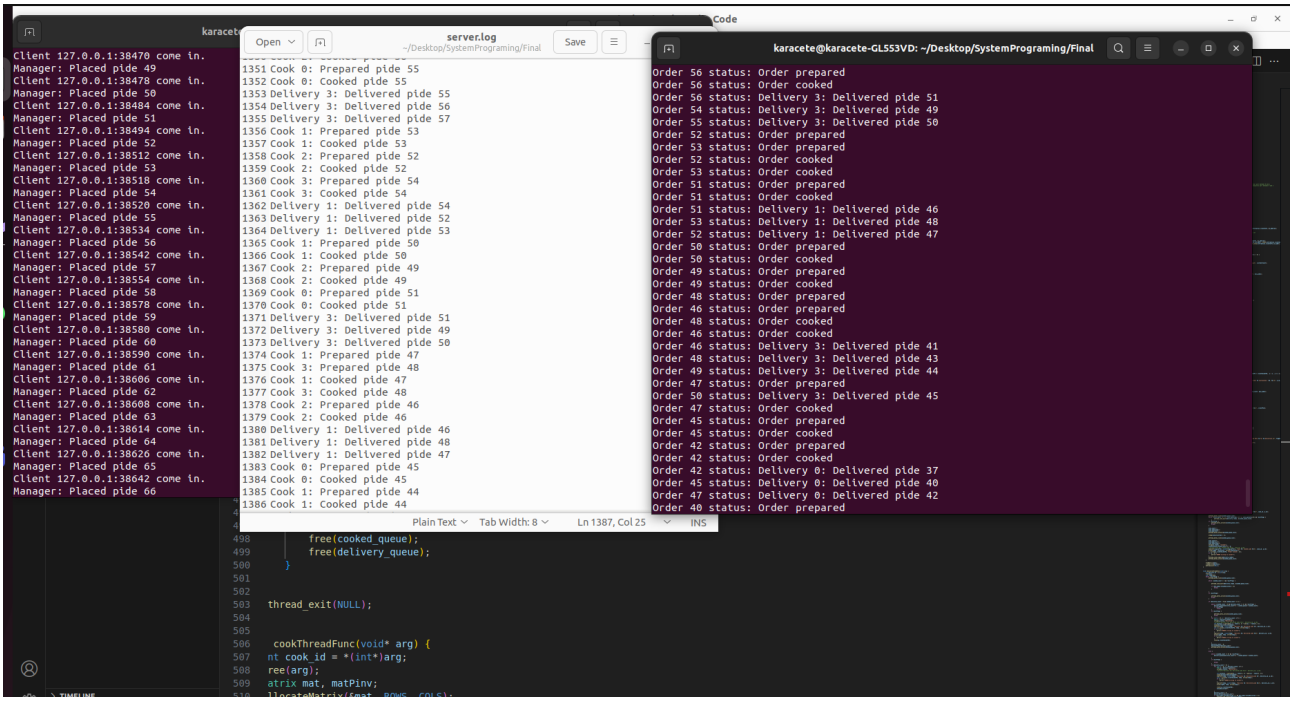
For 100 Order



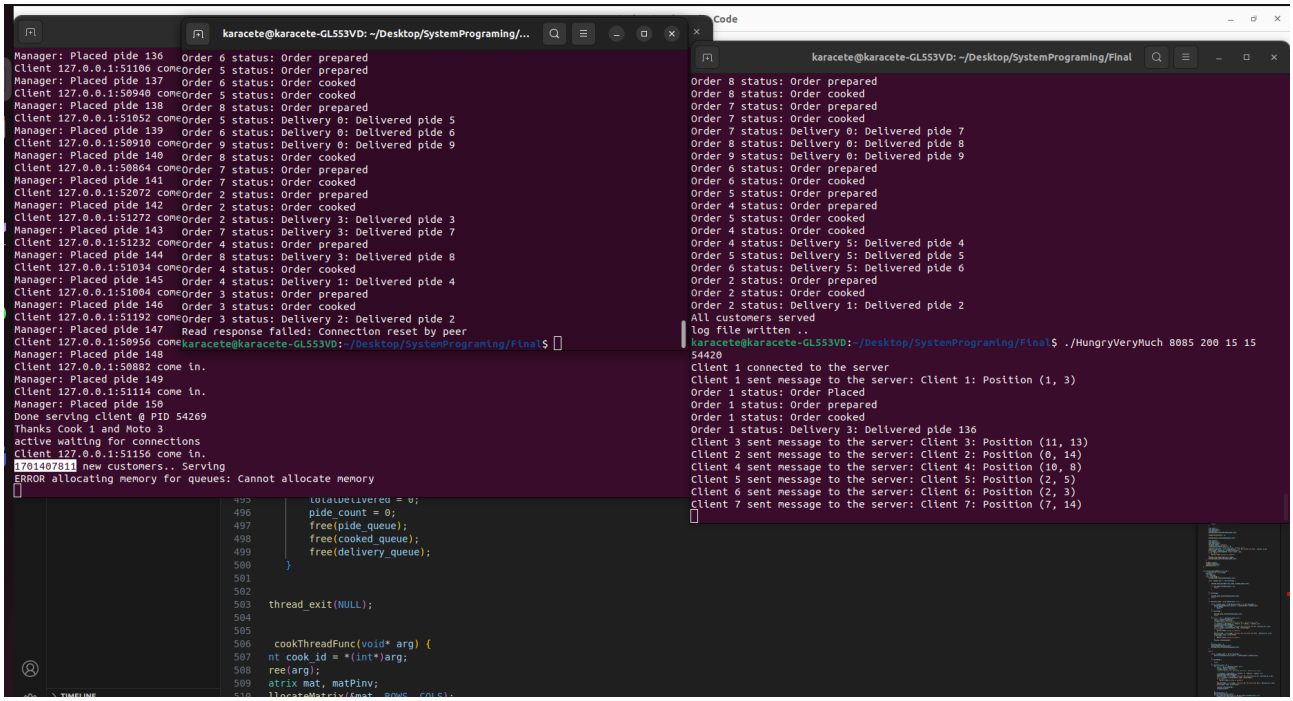


For 200 Order after Calling 100 order



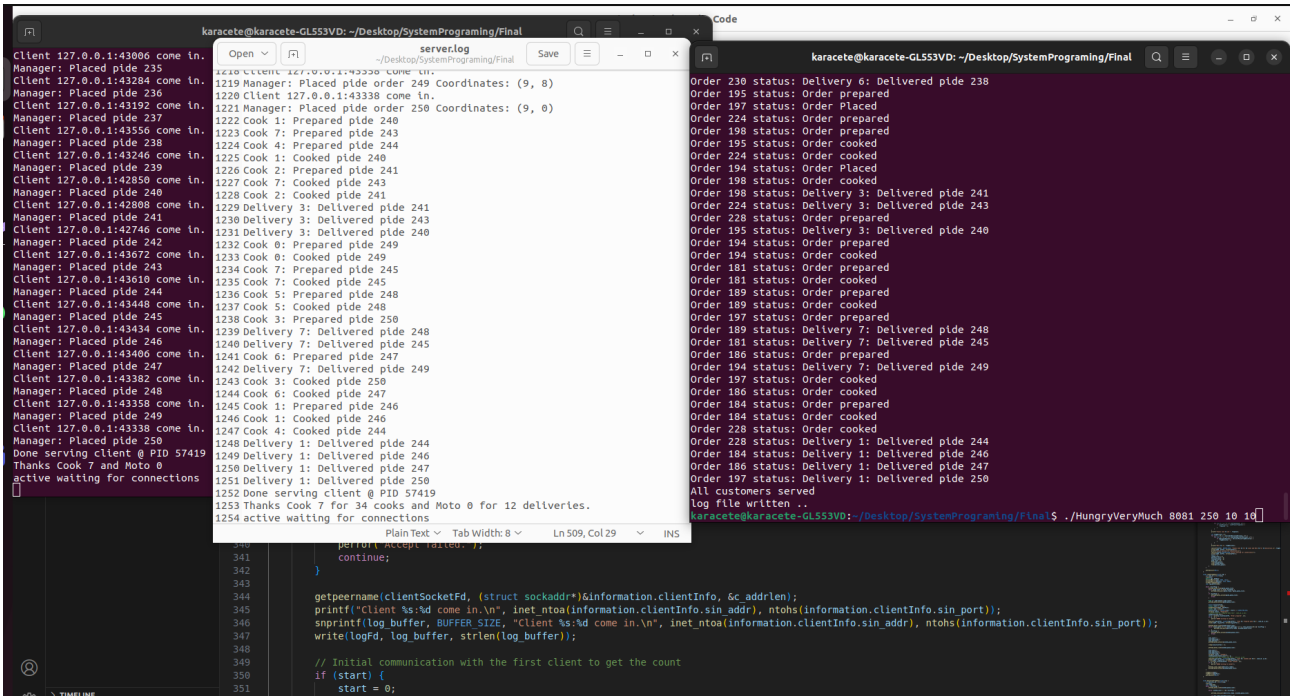


Connecting another terminal while server communicating with a terminal



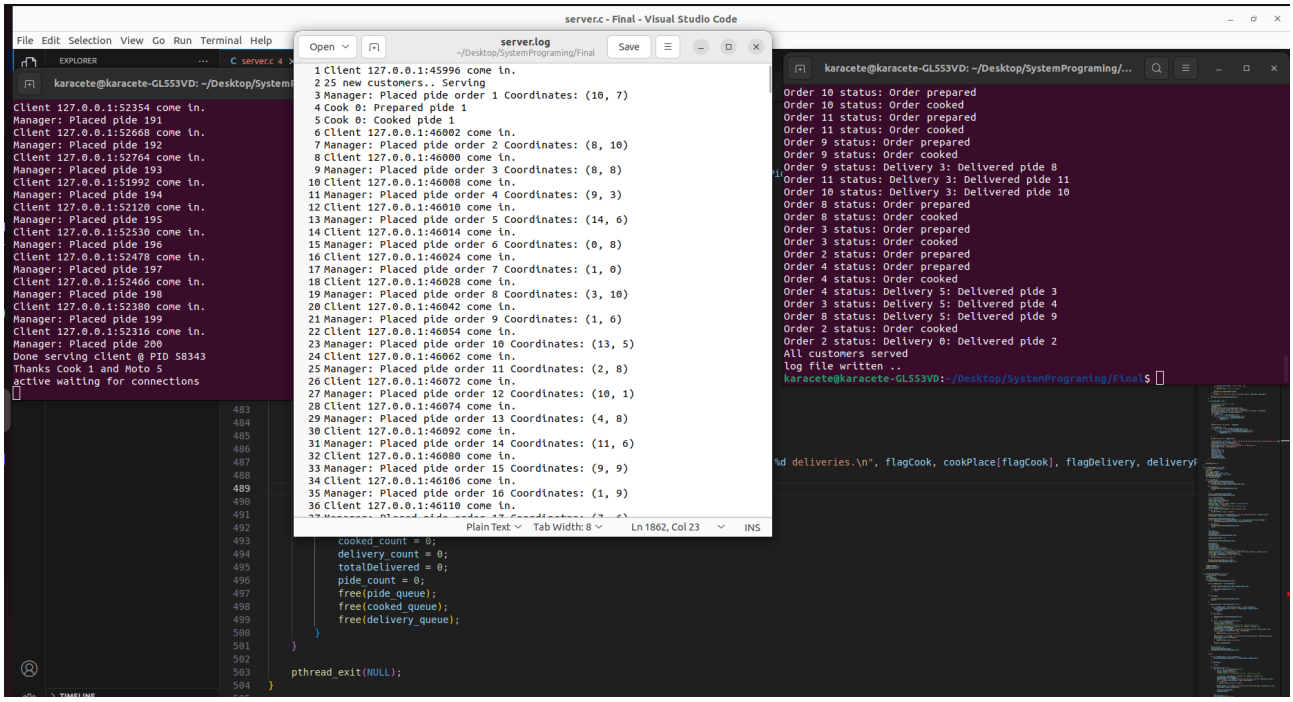
Server could not get customer count right. So go down.

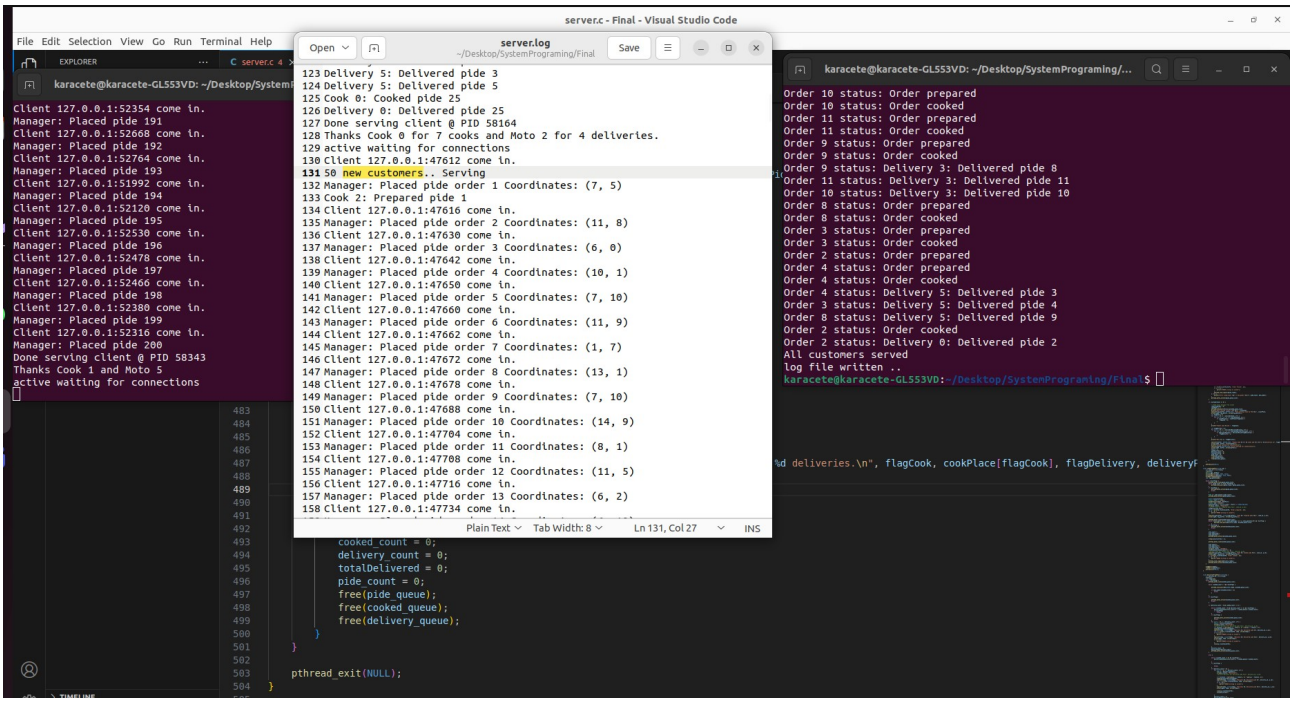
Eight cooks eight deliverer and 250 customer:

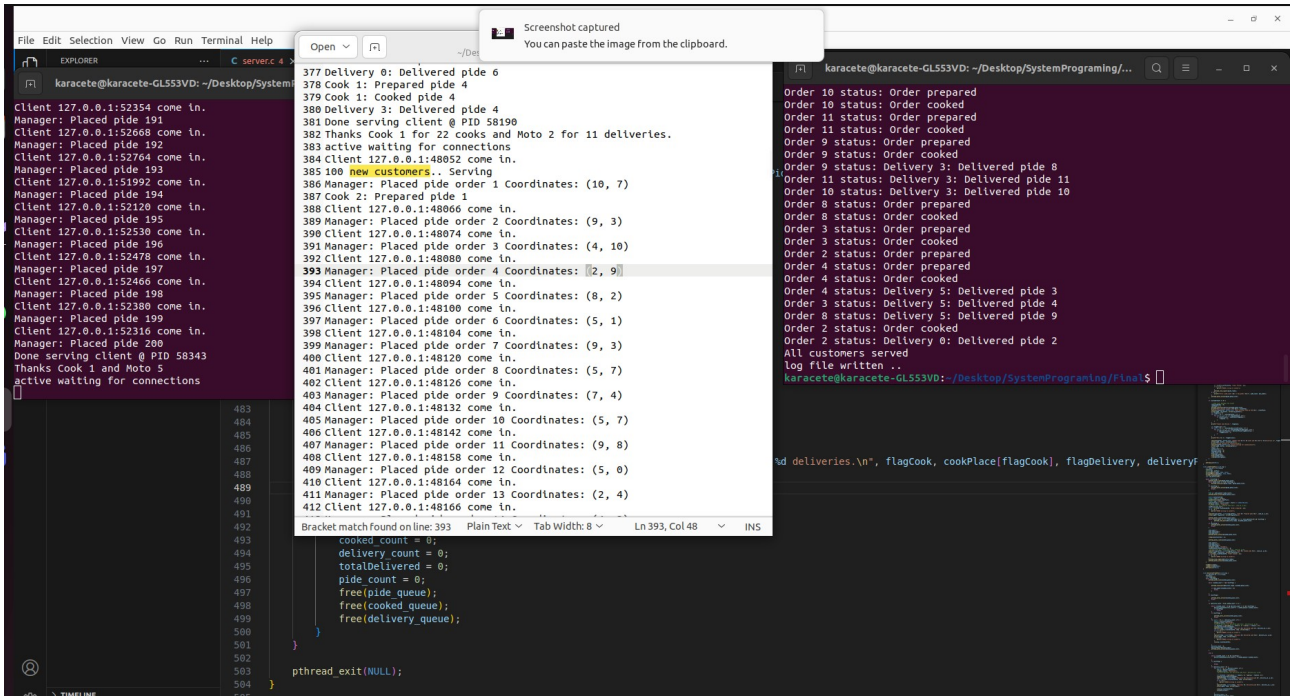


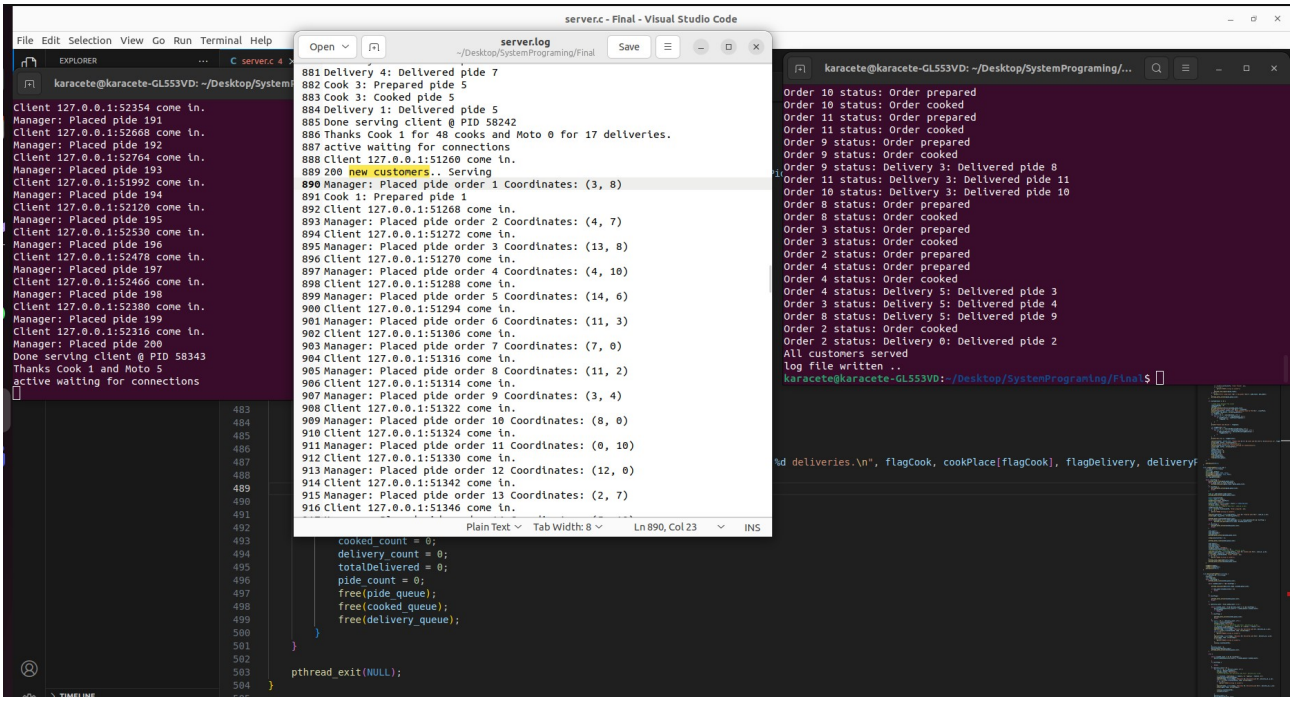
Program does not get any problem when new customers send from terminal.

Serverlog after 25,50,100,200 customers came in on same runnings.






Screenshot captured
You can paste the image from the clipboard.
Client 127.0.0.1:52354 come in.
Manager: Placed pide 191
Client 127.0.0.1:52668 come in.
Manager: Placed pide 192
Client 127.0.0.1:52764 come in.
Manager: Placed pide 193
Client 127.0.0.1:51992 come in.
Manager: Placed pide 194
Client 127.0.0.1:52120 come in.
Manager: Placed pide 195
Client 127.0.0.1:52530 come in.
Manager: Placed pide 196
Client 127.0.0.1:52478 come in.
Manager: Placed pide 197
Client 127.0.0.1:52466 come in.
Manager: Placed pide 198
Client 127.0.0.1:52380 come in.
Manager: Placed pide 199
Client 127.0.0.1:52316 come in.
Manager: Placed pide 200
Done serving client @ PID 58343
Thanks Cook 1 and Moto 5
active waiting for connections
377 Delivery 0: Delivered pide 6
378 Cook 1: Prepared pide 4
379 Cook 1: Cooked pide 4
380 Delivery 3: Delivered pide 4
381 Done serving client @ PID 58190
382 Thanks Cook 1 for 22 cooks and Moto 2 for 11 deliveries.
383 active waiting for connections
384 Client 127.0.0.1:48052 come in.
385 100 new customers.. Serving
386 Manager: Placed pide order 1 Coordinates: (10, 7)
387 Cook 2: Prepared pide 1
388 Client 127.0.0.1:48066 come in.
389 Manager: Placed pide order 2 Coordinates: (9, 3)
390 Client 127.0.0.1:48074 come in.
391 Manager: Placed pide order 3 Coordinates: (4, 10)
392 Client 127.0.0.1:48080 come in.
393 Manager: Placed pide order 4 Coordinates: (2, 9)
394 Client 127.0.0.1:48094 come in.
395 Manager: Placed pide order 5 Coordinates: (8, 2)
396 Client 127.0.0.1:48100 come in.
397 Manager: Placed pide order 6 Coordinates: (5, 1)
398 Client 127.0.0.1:48104 come in.
399 Manager: Placed pide order 7 Coordinates: (9, 3)
400 Client 127.0.0.1:48120 come in.
401 Manager: Placed pide order 8 Coordinates: (5, 7)
402 Client 127.0.0.1:48126 come in.
403 Manager: Placed pide order 9 Coordinates: (7, 4)
404 Client 127.0.0.1:48132 come in.
405 Manager: Placed pide order 10 Coordinates: (5, 7)
406 Client 127.0.0.1:48142 come in.
407 Manager: Placed pide order 11 Coordinates: (9, 8)
408 Client 127.0.0.1:48158 come in.
409 Manager: Placed pide order 12 Coordinates: (5, 0)
410 Client 127.0.0.1:48164 come in.
411 Manager: Placed pide order 13 Coordinates: (2, 4)
412 Client 127.0.0.1:48166 come in.
Order 10 status: Order prepared
Order 10 status: Order cooked
Order 11 status: Order prepared
Order 11 status: Order cooked
Order 9 status: Order prepared
Order 9 status: Order cooked
Order 9 status: Delivery 3: Delivered pide 8
Order 11 status: Delivery 3: Delivered pide 11
Order 10 status: Delivery 3: Delivered pide 10
Order 8 status: Order prepared
Order 8 status: Order cooked
Order 3 status: Order prepared
Order 3 status: Order cooked
Order 2 status: Order prepared
Order 4 status: Order prepared
Order 4 status: Order cooked
Order 4 status: Delivery 5: Delivered pide 3
Order 3 status: Delivery 5: Delivered pide 4
Order 8 status: Delivery 5: Delivered pide 9
Order 2 status: Order cooked
Order 2 status: Delivery 0: Delivered pide 2
All customers served
log file written ..
karacete@karacete-GL553VD:~/Desktop/SystemPrograming/Final$


server.c - Final - Visual Studio Code
server.log
~/Desktop/SystemPrograming/Final
Save
Client 127.0.0.1:52354 come in.
Manager: Placed pide 191
Client 127.0.0.1:52668 come in.
Manager: Placed pide 192
Client 127.0.0.1:52764 come in.
Manager: Placed pide 193
Client 127.0.0.1:51992 come in.
Manager: Placed pide 194
Client 127.0.0.1:52120 come in.
Manager: Placed pide 195
Client 127.0.0.1:52530 come in.
Manager: Placed pide 196
Client 127.0.0.1:52478 come in.
Manager: Placed pide 197
Client 127.0.0.1:52466 come in.
Manager: Placed pide 198
Client 127.0.0.1:52380 come in.
Manager: Placed pide 199
Client 127.0.0.1:52316 come in.
Manager: Placed pide 200
Done serving client @ PID 58343
Thanks Cook 1 and Moto 5
active waiting for connections
881 Delivery 4: Delivered pide 7
882 Cook 3: Prepared pide 5
883 Cook 3: Cooked pide 5
884 Delivery 1: Delivered pide 5
885 Done serving client @ PID 58242
886 Thanks Cook 1 for 48 cooks and Moto 0 for 17 deliveries.
887 active waiting for connections
888 Client 127.0.0.1:51260 come in.
889 200 new customers.. Serving
890 Manager: Placed pide order 1 Coordinates: (3, 8)
891 Cook 1: Prepared pide 1
892 Client 127.0.0.1:51268 come in.
893 Manager: Placed pide order 2 Coordinates: (4, 7)
894 Client 127.0.0.1:51272 come in.
895 Manager: Placed pide order 3 Coordinates: (13, 8)
896 Client 127.0.0.1:51270 come in.
897 Manager: Placed pide order 4 Coordinates: (4, 10)
898 Client 127.0.0.1:51288 come in.
899 Manager: Placed pide order 5 Coordinates: (14, 6)
900 Client 127.0.0.1:51294 come in.
901 Manager: Placed pide order 6 Coordinates: (11, 3)
902 Client 127.0.0.1:51306 come in.
903 Manager: Placed pide order 7 Coordinates: (7, 0)
904 Client 127.0.0.1:51316 come in.
905 Manager: Placed pide order 8 Coordinates: (11, 2)
906 Client 127.0.0.1:51314 come in.
907 Manager: Placed pide order 9 Coordinates: (3, 4)
908 Client 127.0.0.1:51322 come in.
909 Manager: Placed pide order 10 Coordinates: (8, 0)
910 Client 127.0.0.1:51324 come in.
911 Manager: Placed pide order 11 Coordinates: (0, 10)
912 Client 127.0.0.1:51330 come in.
913 Manager: Placed pide order 12 Coordinates: (12, 0)
914 Client 127.0.0.1:51342 come in.
915 Manager: Placed pide order 13 Coordinates: (2, 7)
916 Client 127.0.0.1:51346 come in.
Order 10 status: Order prepared
Order 10 status: Order cooked
Order 11 status: Order prepared
Order 11 status: Order cooked
Order 9 status: Order prepared
Order 9 status: Order cooked
Order 9 status: Delivery 3: Delivered pide 8
Order 11 status: Delivery 3: Delivered pide 11
Order 10 status: Delivery 3: Delivered pide 10
Order 8 status: Order prepared
Order 8 status: Order cooked
Order 3 status: Order prepared
Order 3 status: Order cooked
Order 2 status: Order prepared
Order 4 status: Order prepared
Order 4 status: Order cooked
Order 4 status: Delivery 5: Delivered pide 3
Order 3 status: Delivery 5: Delivered pide 4
Order 8 status: Delivery 5: Delivered pide 9
Order 2 status: Order cooked
Order 2 status: Delivery 0: Delivered pide 2
All customers served
log file written ..
karacete@karacete-GL553VD:~/Desktop/SystemPrograming/Final$