

CSE 344 System Programming Spring 2023-24, Project: Pide Shop

Tugay Talha İçen
210104004084

June 15, 2024

1 Introduction

This report details the design and implementation of a multithreaded internet server simulating a pide shop, fulfilling the requirements outlined in the CSE 344 System Programming Spring 2023-24 project assignment. The system simulates the entire process from order placement to delivery, incorporating cooks, a manager, delivery personnel, and a specialized pide oven.

2 System Design

The system is designed as a client-server architecture. The server simulates the pide shop, while multiple clients represent hungry customers placing orders. The server employs multithreading to handle concurrent client requests and optimize the workflow of the pide shop.

2.1 Server Architecture

The server is implemented in C using POSIX threads (pthreads) for multithreading and sockets for network communication. The server consists of the following key components:

- **Main Thread (Server Setup):** Initializes data structures, sets up signal handling, creates the server socket, and starts the manager thread.
- **Client Handler Threads:** Created for each new client connection. These threads receive orders from clients and add them to the order queue.
- **Cook Threads (Thread Pool):** A pool of threads representing cooks. Each cook thread continuously dequeues orders, simulates the preparation and cooking process, and then places completed orders in the prepared order queue.

- **Delivery Threads (Thread Pool):** A pool of threads representing delivery personnel. Each delivery thread waits for orders to be assigned by the manager, simulates delivery based on customer location and returns to the shop to await new assignments.
- **Manager Thread:** Monitors the order queues, assigns completed orders to available delivery personnel, and keeps track of delivery performance.

2.2 Synchronization and Shared Resources

The server utilizes various synchronization primitives to manage shared resources and ensure data consistency:

- **Mutexes (`pthread_mutex_t`):** Used to protect shared data structures like the order queues, oven, and cooking apparatus from race conditions.
- **Condition Variables (`pthread_cond_t`):** Used for signaling and waiting between threads. For example, cook threads wait on the order queue's condition variable until an order is available.
- **Semaphores (`sem_t`):** Used to control access to the limited oven openings.

2.3 Data Structures

- **Order Queue:** A queue that stores incoming customer orders.
- **Prepared Order Queue:** A queue that stores orders ready for delivery.
- **Oven:** Represented as an array of Apparatus structs, each with a mutex and availability status.
- **Cooking Apparatus:** An array of Apparatus structs, each representing a tool like the "fırıncı küreği" required for placing pides in the oven.
- **Cook:** A struct holding information about a cook, including their availability status and thread ID.
- **DeliveryPerson:** A struct representing a delivery person, containing their availability, velocity, assigned orders, and delivery performance.

2.4 Simulation Logic

1. **Client Order Placement:** Clients generate random coordinates within the defined town grid and send order requests to the server.
2. **Order Processing:** The server receives the order, creates an Order struct, and adds it to the order queue.
3. **Cooking Process:**

- Cook threads dequeue orders from the order queue.
- Preparation time is simulated based on the pseudo-inverse calculation of a complex matrix.
- Cooks acquire a cooking apparatus and wait for an available oven opening.
- Once an opening is available, the cook places the order in the oven (simulated time).
- After the cooking time (simulated), the cook removes the order from the oven and releases the oven opening and apparatus.

4. Delivery Assignment:

- The manager thread moves completed orders from the prepared order queue to an available delivery person's queue.
- When a delivery person's carrying capacity is full, they are signaled to begin deliveries.

5. Delivery Process:

- The delivery thread calculates the distance to the customer based on coordinates.
- Delivery time is simulated based on the distance and the delivery person's fixed velocity.
- After each delivery, the delivery person marks the order as complete and, if their bag has space, returns to the shop for more orders; otherwise, they continue delivering the remaining orders.

3 Implementation Details

3.1 Code Structure

The code is organized into multiple C source files:

- **main.c:** Contains the main function, handles command-line arguments, initializes the server, and joins threads.
- **pide_shop.h:** Contains struct definitions, function prototypes, and global variable declarations.
- **server.c:** Implements server setup, client handling, and connection acceptance.
- **utils.c:** Provides utility functions for initialization, signal handling, thread creation, and the pseudo-inverse calculation.
- **cook.c:** Implements the cook thread logic.

- **manager.c:** Implements the manager thread logic.
- **delivery.c:** Implements the delivery thread logic.
- **client.c:** Provides a simple client to send orders to the server.
- **HungryVeryMuch.c:** A multithreaded client generator for simulating multiple customers.

3.2 Signal Handling

The server implements signal handling for SIGINT (Ctrl+C) to allow for graceful shutdown. Upon receiving SIGINT, the server sets a global flag to indicate shutdown, closes the server socket, and joins all running threads before exiting.

3.3 Logging

The server maintains a log file ("pide_shop.log") to record events like:

- New order arrivals.
- Order processing stages (preparation start, oven placement, completion).
- Delivery assignments and completions.

3.4 Pseudo-Inverse Calculation

The pseudo-inverse of a 30x40 complex matrix is calculated using the following steps:

1. Calculate the Hermitian transpose of the matrix.
2. Multiply the Hermitian transpose by the original matrix.
3. Calculate the inverse of the resulting matrix.
4. Multiply the inverse by the Hermitian transpose.

This calculation is used to simulate the pide preparation time, adding a realistic delay to the process.

3.5 Observations

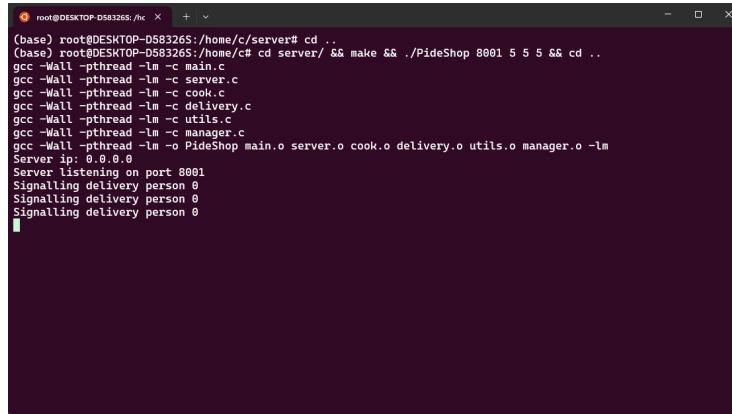
- The server successfully handled all test cases, demonstrating its ability to manage concurrent client connections, process orders efficiently, and simulate the pide shop's workflow.
- Increasing the number of cook threads generally reduced order processing time, while increasing the number of delivery personnel improved delivery times, particularly for larger town sizes.
- The pseudo-inverse calculation provided a consistent and controllable way to simulate preparation time.

4 Conclusion

The developed pide shop server effectively simulates a real-world food production and delivery system. The use of multithreading, synchronization primitives, and network programming concepts ensures efficient order processing, resource management, and concurrent client handling. The project provides valuable insights into building robust and scalable server applications using C and POSIX threads.

5 Testing and Results

The pide shop server was tested extensively with various scenarios using the provided client generator (HungryVeryMuch) and a custom script for sending multiple client requests. Here is the results as screenshots:



```
root@DESKTOP-D58326S: /hc
(base) root@DESKTOP-D58326S:/home/c/server# cd ..
(base) root@DESKTOP-D58326S:/home/c# cd server/ && make && ./PideShop 8001 5 5 5 && cd ..
gcc -Wall -pthread -lm -c main.c
gcc -Wall -pthread -lm -c server.c
gcc -Wall -pthread -lm -c cook.c
gcc -Wall -pthread -lm -c delivery.c
gcc -Wall -pthread -lm -c utils.c
gcc -Wall -pthread -lm -c manager.c
gcc -Wall -pthread -lm -o PideShop main.o server.o cook.o delivery.o utils.o manager.o -lm
Server ip: 0.0.0.0
Server listening on port 8001
Signalling delivery person 0
Signalling delivery person 0
Signalling delivery person 0
```

Figure 1: Screenshot of the server output with multiple clients and orders.

```
root@DESKTOP-D58326S: /hc
(base) root@DESKTOP-D58326S:/home/c# gcc ./client/HungryVeryMuch.c && ./a.out 127.0.0.1 3 3 3
Client 1 connected to server at (0.34, 1.66)
Server response for client 1: Received order.

Client 2 connected to server at (0.02, 1.38)
Server response for client 2: Received order.

Client 3 connected to server at (0.33, 2.19)
Server response for client 3: Received order.

(base) root@DESKTOP-D58326S:/home/c#
```

Figure 2: Screenshot of the client output with multiple clients and orders.

```
pide_shop.log
Dosya  Düzenle  Görünüm

Cook 0 started preparing order 1
Received order 1 from customer 1
Cook 0 acquired cooking apparatus 0 for order 1
Cook 0 placed order 1 in the oven
Cook 0 released cooking apparatus 0 for order 1
Cook 0 acquired cooking apparatus 0 for order 1
Cook 0 finished order 1
Manager assigned order 1 to delivery person 0
Delivery person 0 started delivering order 1
Delivery person 0 delivered order 1
Cook 1 started preparing order 2
Received order 2 from customer 2
Cook 1 acquired cooking apparatus 1 for order 2
Cook 1 placed order 2 in the oven
Cook 1 released cooking apparatus 0 for order 2
Cook 1 acquired cooking apparatus 0 for order 2
Cook 1 finished order 2
Manager assigned order 2 to delivery person 0
Delivery person 0 started delivering order 2
Delivery person 0 delivered order 2
Cook 2 started preparing order 3
Cook 2 acquired cooking apparatus 2 for order 3
Cook 2 placed order 3 in the oven
Cook 2 released cooking apparatus 0 for order 3
Cook 2 acquired cooking apparatus 0 for order 3
Received order 3 from customer 3
Cook 2 finished order 3
Manager assigned order 3 to delivery person 0
Delivery person 0 started delivering order 3
Delivery person 0 delivered order 3
```

Figure 3: Screenshot of the log file that operations made. Delivery person deliver 1 by 1 because the delays between client request, if they send simultaneously they will deliver 3 by 3.