
# Gebze Technical University

## Department of Computer Engineering

## CSE344 System Programming

---

# Final Report

---

Burak Yıldırım
1901042609

# Contents

# 1 Introduction

## 1.1 Project Description

The task of this project is to simulate a food production and delivery system. The overall system is composed of three main components: (i) the pide house containing a special pide oven (ii) delivery personel waiting on their motorsycles infront of the store and (iii) lots of hungry costumers.

## 1.2 Compilation

```
CC = gcc
CFLAGS = -Wall -pedantic-errors
LIBFLAGS = -pthread -lrt -lm
DFLAGS = -g
INCDIR = include
SRCDIR = src
LOG_EXT = log
SERVER_TARGET = PideShop
CLIENT_TARGET = HungryVeryMuch
UTIL_SOURCES = $(wildcard $(SRCDIR)/utils/*.c)
SHARED_SOURCE = $(SRCDIR)/shared.c
SERVER_SOURCES = $(SRCDIR)/server.c $(SHARED_SOURCE) $(UTIL_SOURCES)
CLIENT_SOURCES = $(SRCDIR)/client.c $(SHARED_SOURCE) $(UTIL_SOURCES)
HEADERS = $(wildcard $(INCDIR)/*.h) $(wildcard $(INCDIR)/utils/*.h)
VALGRIND_OPTIONS = --leak-check=full --show-leak-kinds=all --track-origins=yes
V_PORT =
V_COOK =
V_DELIVERY =
V_SPEED =
V_IP =
V_CLIENT =
V_P =
V_Q =

export V_PORT
export V_COOK
export V_DELIVERY
export V_SPEED
export V_IP
export V_CLIENT
export V_P
export V_Q

all: server client

server: $(SERVER_SOURCES) $(HEADERS)
	$(CC) $(CFLAGS) -o $(SERVER_TARGET) $(SERVER_SOURCES) -I$(INCDIR) $(LIBFLAGS)

client: $(CLIENT_SOURCES) $(HEADERS)
	$(CC) $(CFLAGS) -o $(CLIENT_TARGET) $(CLIENT_SOURCES) -I$(INCDIR) $(LIBFLAGS)

debug_server:
	$(CC) $(CFLAGS) $(DFLAGS) -o $(SERVER_TARGET) $(SERVER_SOURCES) -I$(INCDIR) $(LIBFLAGS)

debug_client:
	$(CC) $(CFLAGS) $(DFLAGS) -o $(CLIENT_TARGET) $(CLIENT_SOURCES) -I$(INCDIR) $(LIBFLAGS)

clean:
	rm -f $(SERVER_TARGET) $(CLIENT_TARGET) *.$(LOG_EXT)

valgrind_server: debug_server
	valgrind $(VALGRIND_OPTIONS) ./$(SERVER_TARGET) $(V_PORT) $(V_COOK) $(V_DELIVERY) $(V_SPEED)

valgrind_client: debug_client
	valgrind $(VALGRIND_OPTIONS) ./$(CLIENT_TARGET) $(V_IP) $(V_PORT) $(V_CLIENT) $(V_P) $(V_Q)
```

The provided Makefile automates the build process: executing **make** compiles the project, **make clean** removes executables, and logs, and **make valgrind_server** and **make valgrind_client** launches the server and client programs in Valgrind for memory leak checks.

# 2 General Structure

```
Final
├── include
│   ├── utils
│   │   ├── array.h
│   │   ├── io.h
│   │   ├── log.h
│   │   ├── matrix.h
│   │   ├── string.h
│   │   └── type.h
│   ├── client.h
│   ├── server.h
│   └── shared.h
├── src
│   ├── utils
│   │   ├── array.c
│   │   ├── io.c
│   │   ├── log.c
│   │   ├── matrix.c
│   │   └── string.c
│   ├── client.c
│   ├── server.c
│   └── shared.c
└── Makefile
```

This is the folder structure of the project.

- utils: utility functions used by multiple files.
- client: client-specific functions and macros.
- server: server-specific functions and macros.
- shared: common functions and macros used by client and server.
- Makefile: compile project, and clean executables and logs.

# 3 Structs and Enums

```
typedef enum {
    PLACED,
    PREPARING,
    PREPARED,
    COOKING,
    COOKED,
    DELIVERING,
    DELIVERED,
    DONE
} OrderStatus;

typedef struct {
    int x;
    int y;
} Coordinate;

typedef struct {
    Coordinate destination;
    Coordinate area_end;
    OrderStatus status;
    int id;
    int cook_id;
    int delivery_id;
    pid_t pid;
} Order;

typedef struct {
    Order *entries;
    int size;
    int capacity;
} OrderQueue;
```

# 4 Server Implementation

First all the necessary checks for the arguments are done. Then `SIGCHLD` and `SIGINT` are handled. `cleanup` function is registered with `atexit`. Queues, threads, and statistics arrays are dynamically allocated. Socket is created. Log file is created. Manager timer is set. This timer periodically checks if there is any order in the cooked queue. If there is at least one, these orders are then taken from the cooked queue and put into the delivery queue and condition variable for delivery is signaled. After the timer setup, server runs in an double infinite loop. In the outer loop, it accepts the connection, and in the inner loop it reads orders from the socket. Read orders are put in the preparation queue for cooks to get from. If the status of the read order is `DONE`, it breaks the inner loop and begins signalling cook threads. After that it waits a condition variable which is only signaled when the delivery is done. After it's signaled it send an order with status `DONE` to the client to indicate all customers are server, it finds the most efficient cook and delivery person and prints them, and finally resets all the queues, arrays and counters. Cook thread runs in an infinite loop. When signaled it dequeues an order from the preparation queue, and starts doing the pseudo inverse calculations for preparation. Time it took to do these calculations are stored and used in the cooking. After preparation it sets up a timer for the half of the time of the calculations. This timer removes the order from the oven and puts it in the cooked queue when expired. At the end of the iteration it increase the number of orders it done. Delivery thread runs in an infinite loop. When signaled it dequeues an order from the delivery queue and delivers it to the destination of the order by calculating the time required and sleeping for that amount. Time is calculated as distance/k. Then from the destination it calculates the distance to go back to the pide oven and sleeps for that amount. After arriving back at pide oven it increases the number of orders it delivered. If this delivery was the last delivery it set the `finished` flag to `TRUE` and signals the condition variable the manager is waiting on.

# 5 Client Implementation

First all the necessary checks for the arguments are done. Then `SIGCHLD` and `SIGINT` are handled. `cleanup` function is registered with `atexit`. Socket is created and connected. First client runs in a for loop for client number times. It generates and sends order to the server. After for loop, it sends an order with its status `DONE` to indicate the end of orders. Then it runs in an infinite loop. In the loop it reads the orders sent from the server and prints their status. If the status of an order is `DONE`, it means all the customers are server so it breaks the loops and exits.

# 6 Signal Handling

In the server it prints a message, finishes all the threads and does the cleanup by closing the socket and destroying all the mutexes, condition variables, and freeing all the dynamically allocated variables. In the client, it prints a message, and does the cleanup by closing the socket.

# 7 Testing

## 7.1 50 Customers

### 7.1.1 Client

```
[burak@fedora Final]$ ./HungryVeryMuch 127.0.0.1 8080 50 10 20
> PID 22356
> Connected to the server at 127.0.0.1:8080
> Status of Order #1 with destination (9, 13): Placed
> Status of Order #2 with destination (9, 13): Placed
> Status of Order #49 with destination (0, 6): Delivered
> Status of Order #50 with destination (6, 2): Delivered
> All customers served
> Log file written
[burak@fedora Final]$
```

### 7.1.2 Server

```
[burak@fedora Final]$ ./PideShop 8080 4 6 1
> PideShop active waiting for connection
> 50 new customers... Serving
> Done serving client @ XXX PID 22356
> Thanks Cook 1 and Moto 2
> Active waiting for connections
```

### 7.1.3 Log

```
[2024-06-15 04:02:41] [INFO] All orders for PID 22356 are delivered.
[2024-06-15 04:02:41] [INFO] Delivery personel #2 is the most productive with 12 orders.
[2024-06-15 04:02:41] [INFO] Cook #1 is the most productive with 13 orders.
```

## 7.2 100 Customers

### 7.2.1 Client

```
[burak@fedora Final]$ ./HungryVeryMuch 127.0.0.1 8080 100 10 20
> PID 23203
> Connected to the server at 127.0.0.1:8080
> Status of Order #1 with destination (2, 4): Placed
> Status of Order #2 with destination (5, 14): Placed
> Status of Order #100 with destination (2, 14): Delivered
> Status of Order #99 with destination (7, 3): Delivered
> All customers served
> Log file written
[burak@fedora Final]$
```

### 7.2.2 Server

```
> 100 new customers... Serving
> Done serving client @ XXX PID 23203
> Thanks Cook 4 and Moto 3
> Active waiting for connections
```

### 7.2.3 Log

```
[2024-06-15 04:15:58] [INFO] All orders for PID 23203 are delivered.
[2024-06-15 04:15:58] [INFO] Delivery personel #3 is the most productive with 20 orders.
[2024-06-15 04:15:58] [INFO] Cook #4 is the most productive with 26 orders.
```

## 7.3 200 Customers

### 7.3.1 Client

```
[burak@fedora Final]$ ./HungryVeryMuch 127.0.0.1 8080 200 10 20
> PID 25214
> Connected to the server at 127.0.0.1:8080
> Status of Order #1 with destination (4, 0): Placed
> Status of Order #2 with destination (9, 16): Placed
> Status of Order #199 with destination (9, 15): Delivered
> Status of Order #200 with destination (5, 16): Delivered
> All customers served
> Log file written
[burak@fedora Final]$
```

### 7.3.2 Server

```
> 200 new customers... Serving
> Done serving client @ XXX PID 25214
> Thanks Cook 1 and Moto 5
> Active waiting for connections
```

### 7.3.3 Log

```
[2024-06-15 04:43:21] [INFO] All orders for PID 25214 are delivered.
[2024-06-15 04:43:21] [INFO] Delivery personel #5 is the most productive with 36 orders.
[2024-06-15 04:43:21] [INFO] Cook #1 is the most productive with 51 orders.
```

## 7.4 SIGINT On Server

### 7.4.1 Client

```
> Status of Order #10 with destination (7, 4): Preparing
> Status of Order #10 with destination (7, 4): Preparing
> Status of Order #10 with destination (7, 4): Prepared
> Status of Order #10 with destination (7, 4): Prepared
> Status of Order #10 with destination (7, 4): Prepared
> Pide shop burnt down. Exiting...
[burak@fedora Final]$
```

### 7.4.2 Server

```
[burak@fedora Final]$ ./PideShop 8080 4 6 1
> PideShop active waiting for connection
> 10 new customers... Serving
^C
> Upps quitting...
Writing log file...
> Cook 1 is joining
> Cook 2 is joining
> Cook 3 is joining
> Cook 4 is joining
> Delivery 1 is joining
> Delivery 2 is joining
> Delivery 3 is joining
> Delivery 4 is joining
> Delivery 5 is joining
> Delivery 6 is joining
[burak@fedora Final]$
```

## 7.5 SIGINT On Client

### 7.5.1 Client

```
> Status of Order #9 with destination (5, 19): Delivering
> Status of Order #1 with destination (5, 10): Delivered
> Status of Order #4 with destination (5, 2): Delivering
^C> Signal...
> Cancelling orders...
> Editing log...
[burak@fedora Final]$
```

### 7.5.2 Server

```
[burak@fedora Final]$ ./PideShop 8080 4 6 1
> PideShop active waiting for connection
> 10 new customers... Serving
> Pide shop bankrupted. Exiting...
> Cook 1 is joining
> Cook 2 is joining
> Cook 3 is joining
> Cook 4 is joining
> Delivery 1 is joining
> Delivery 2 is joining
> Delivery 3 is joining
> Delivery 4 is joining
> Delivery 5 is joining
> Delivery 6 is joining
[burak@fedora Final]$
```