University of Saint Thomas, Maputo, Mozambique  
Computer Science 3rd year  
Parallel Computing   
Lecturer: Lars Lamos

Assignment 1

Supermarket cashier Parallelism Documentation

Group Members : Erick Wasonga , Kainza Scovia, Daniel Darsamo

Contents

[Documentation of Supermarket Cashier code showcasing parallelism 2](#_Toc169105476)

[Code Structure 2](#_Toc169105477)

[Headers and Definitions 2](#_Toc169105478)

[Data Structures 3](#_Toc169105479)

[Helper Function 3](#_Toc169105480)

[Checkout Function 3](#_Toc169105481)

[Utility Functions 4](#_Toc169105482)

[Main function 6](#_Toc169105483)

[Execution flow 7](#_Toc169105484)

[Key Points 7](#_Toc169105485)

[Output example 8](#_Toc169105486)

# Documentation of Supermarket Cashier code showcasing parallelism

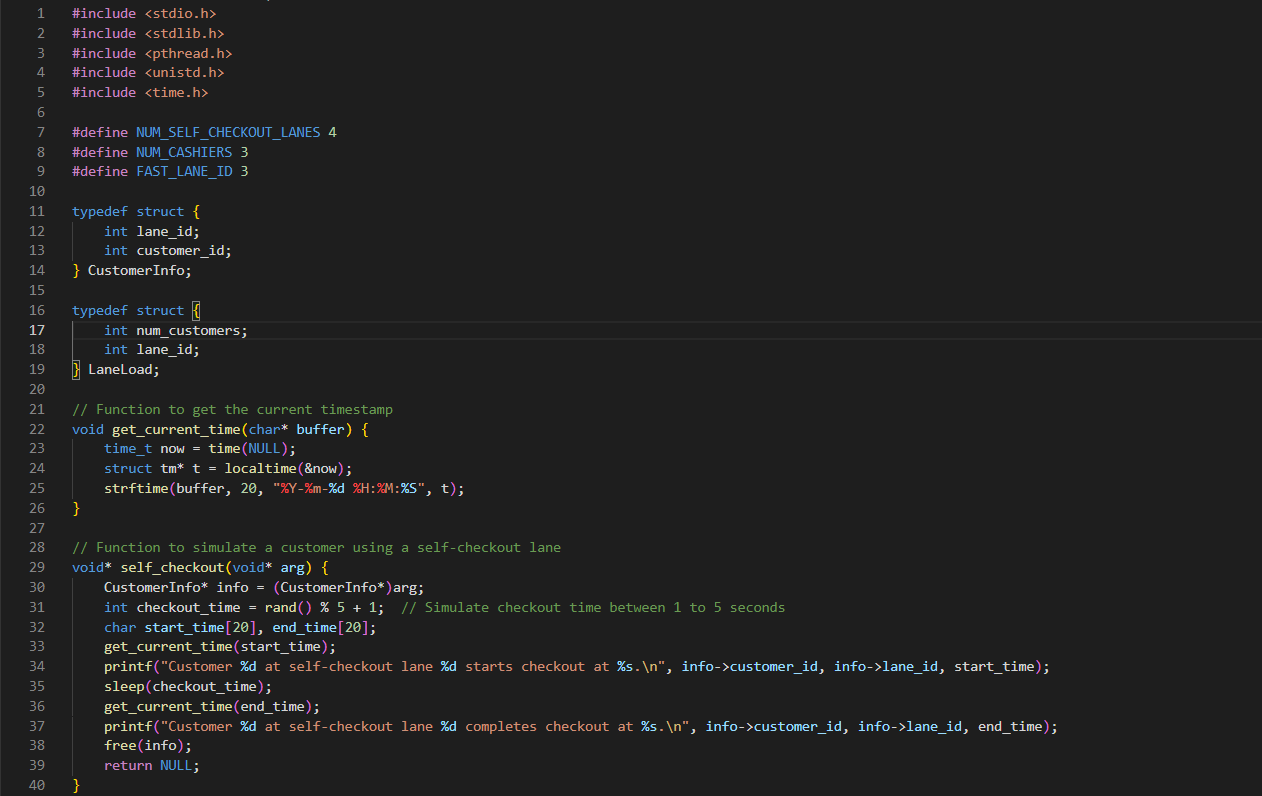
This program simulates parallel processing of customers in a supermarket checkout scenario using C and POSIX threads (pthreads). The supermarket has two types of lanes:

* Self-checkout lanes: managed by a single cashier who oversees multiple self-checkout lanes.
* Cashier lanes: traditional lanes managed by individual cashiers.

The program takes the number of customers for each type of lane as input, processes them in parallel, and prints the start and end times of each customer's checkout process, along with a summary of the number of customers per lane.

# Code Structure

## Headers and Definitions

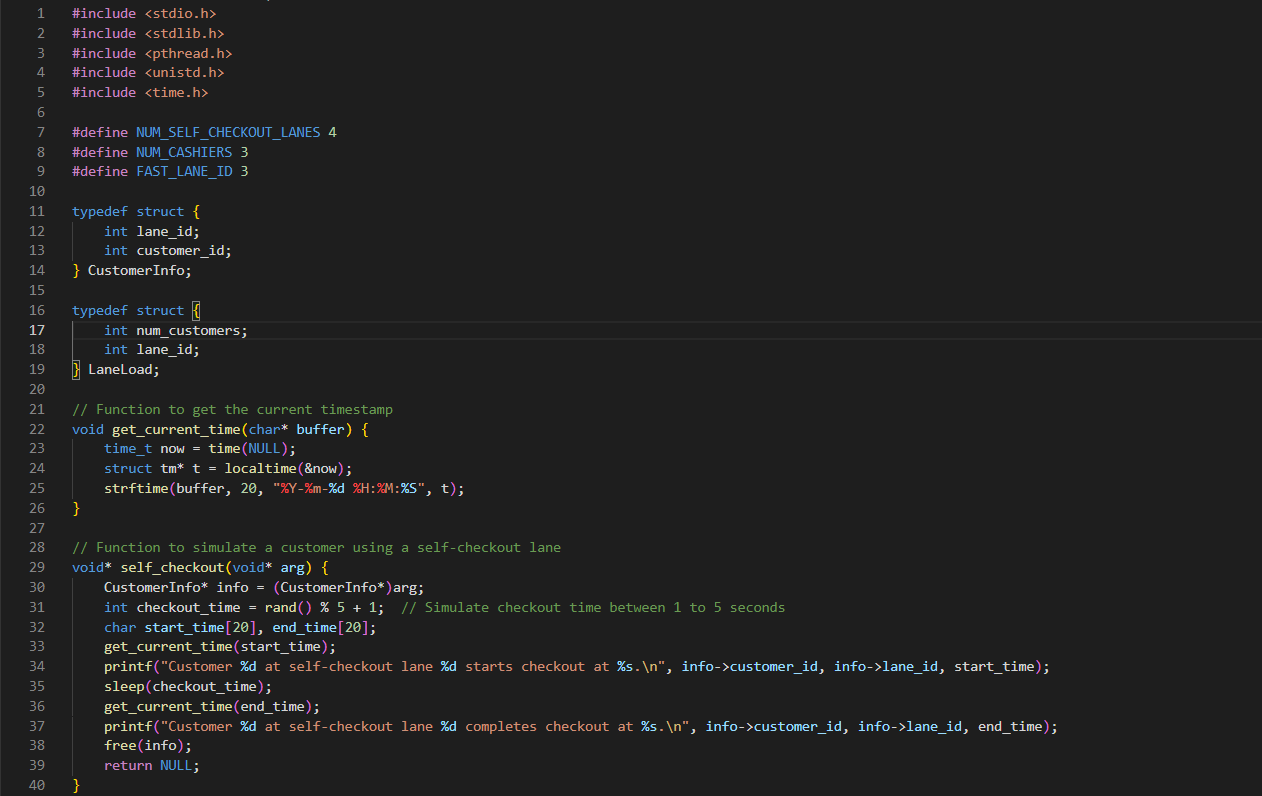
  
**Headers**

* **stdio.h**: Standard input and output library.
* **stdlib.h**: Standard library for memory allocation, random number generation, etc.
* **pthread.h**: POSIX thread library for multithreading.
* **unistd.h**: Provides access to the POSIX operating system API.
* **time.h**: Time library for timestamp functions

**Definitions**

* **NUM\_SELF\_CHECKOUT\_LANES**: Number of self-checkout lanes.
* **NUM\_CASHIERS:** Number of cashier lanes, including the fast lane.
* **FAST\_LANE\_ID:** Identifier for the fast lane.

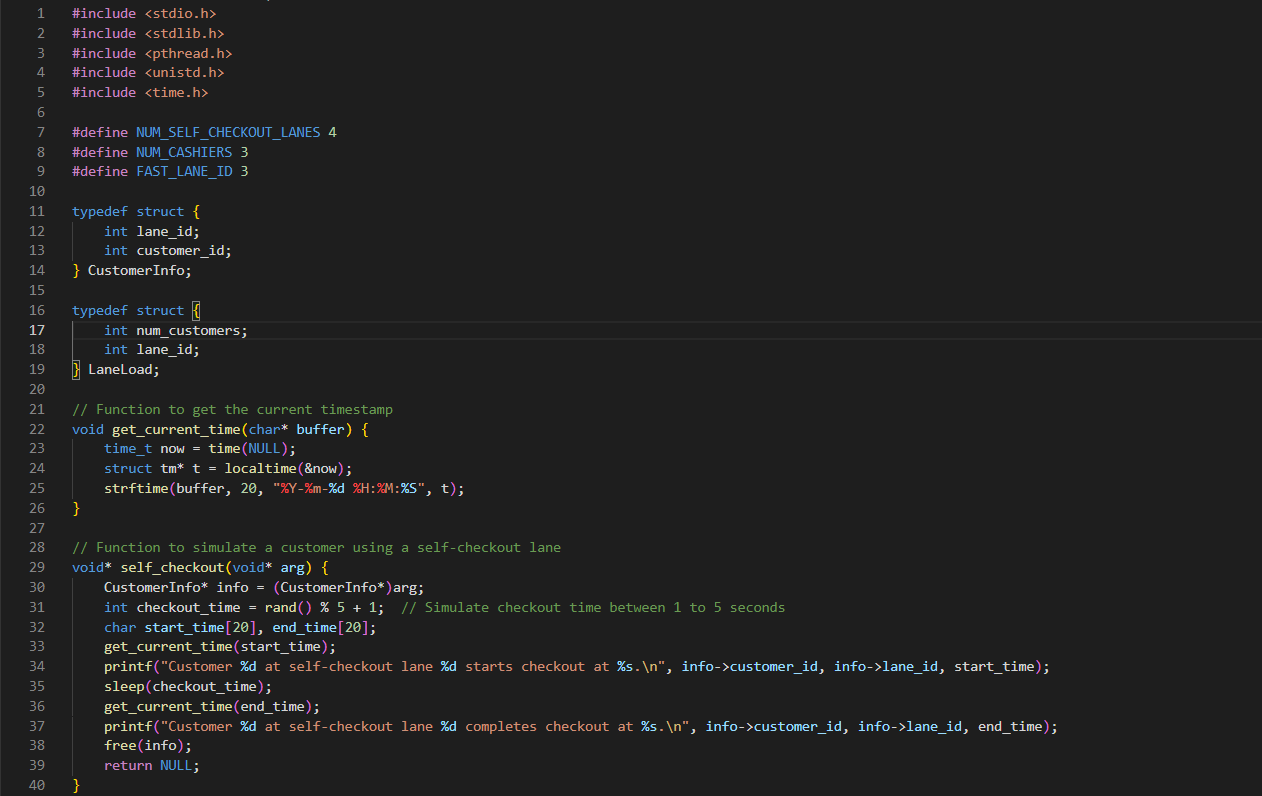
## Data Structures

* **CustomerInfo**: Holds the lane ID and customer ID for each customer.
* **LaneLoad**: Tracks the number of customers in each lane.

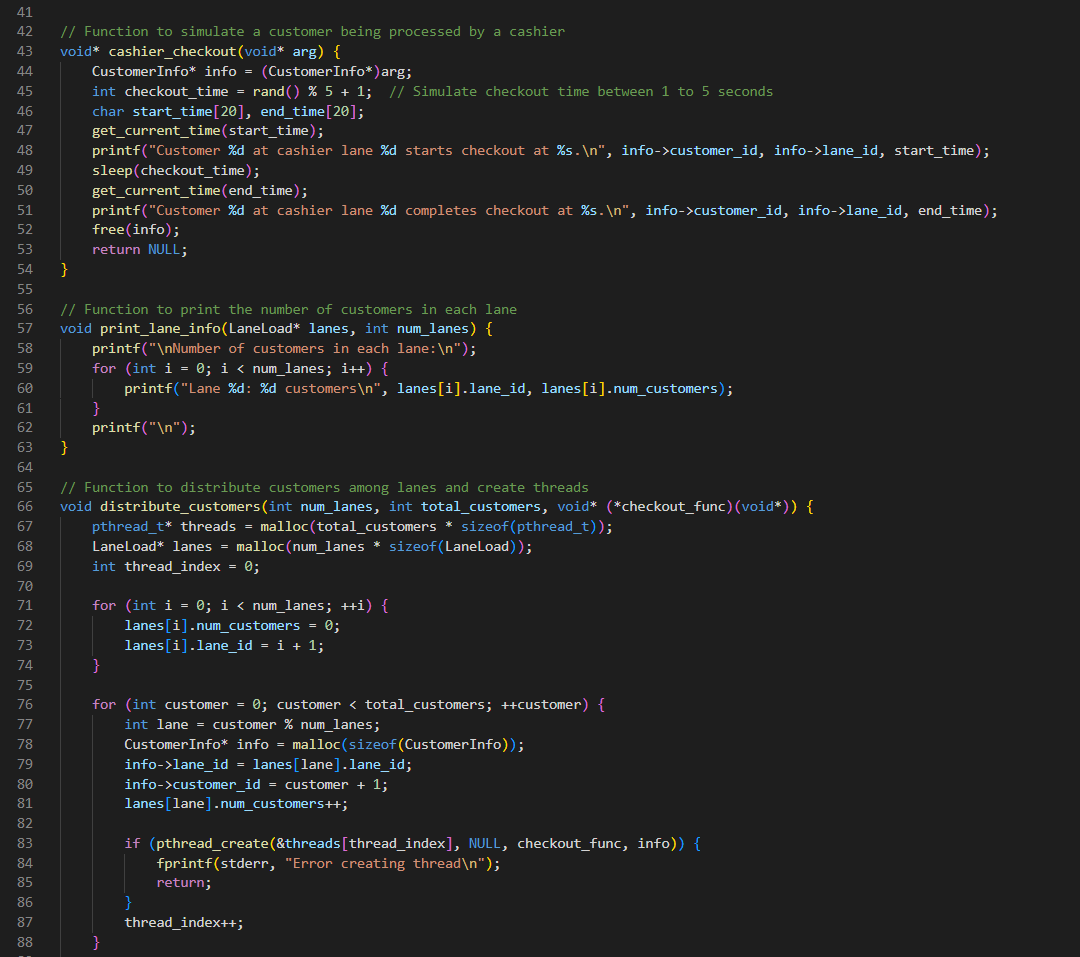
## Helper Function

* **get\_current\_time :** Gets the current timestamp and formats it as a string.

## Checkout Function

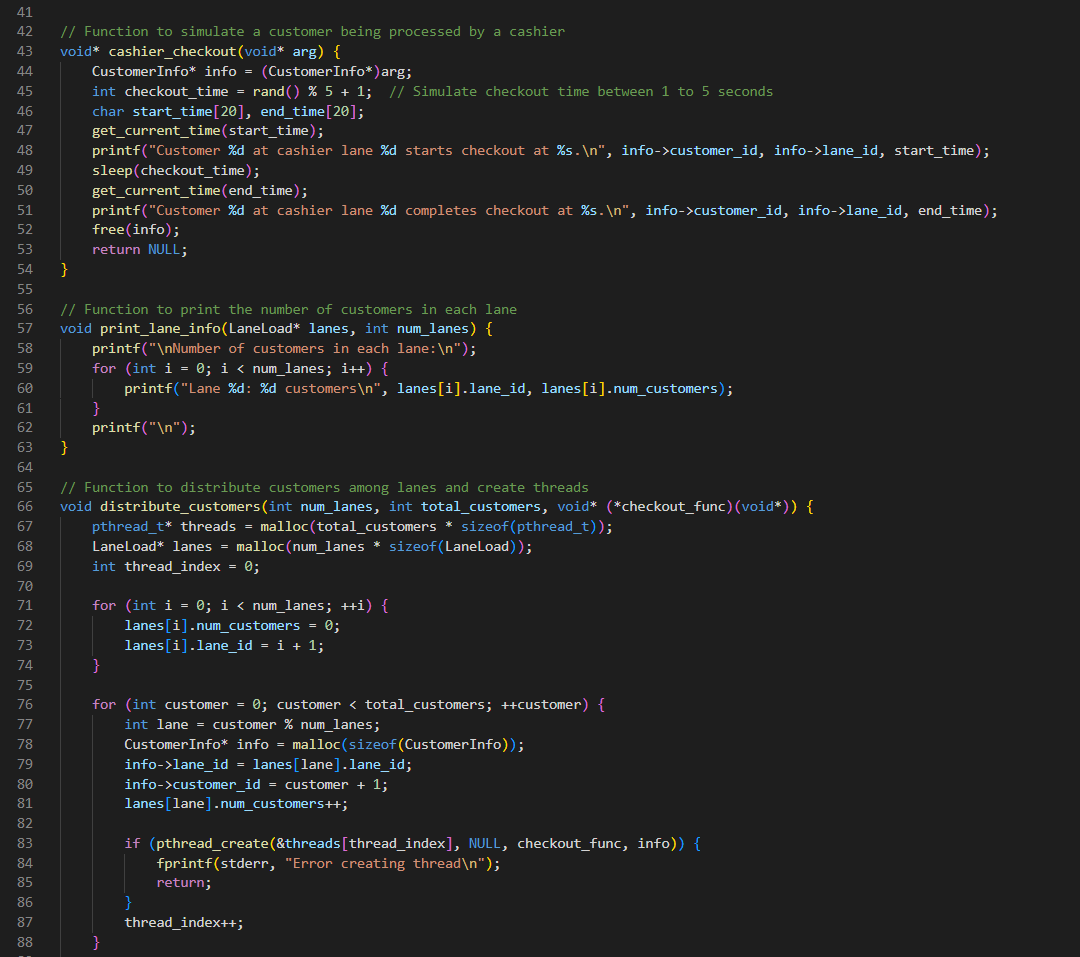
**‘self\_checkout’ :**

* Simulates a customer using a self-checkout lane.
* Randomly determines the checkout time (1-5 seconds).
* Prints the start and end times of the checkout process.

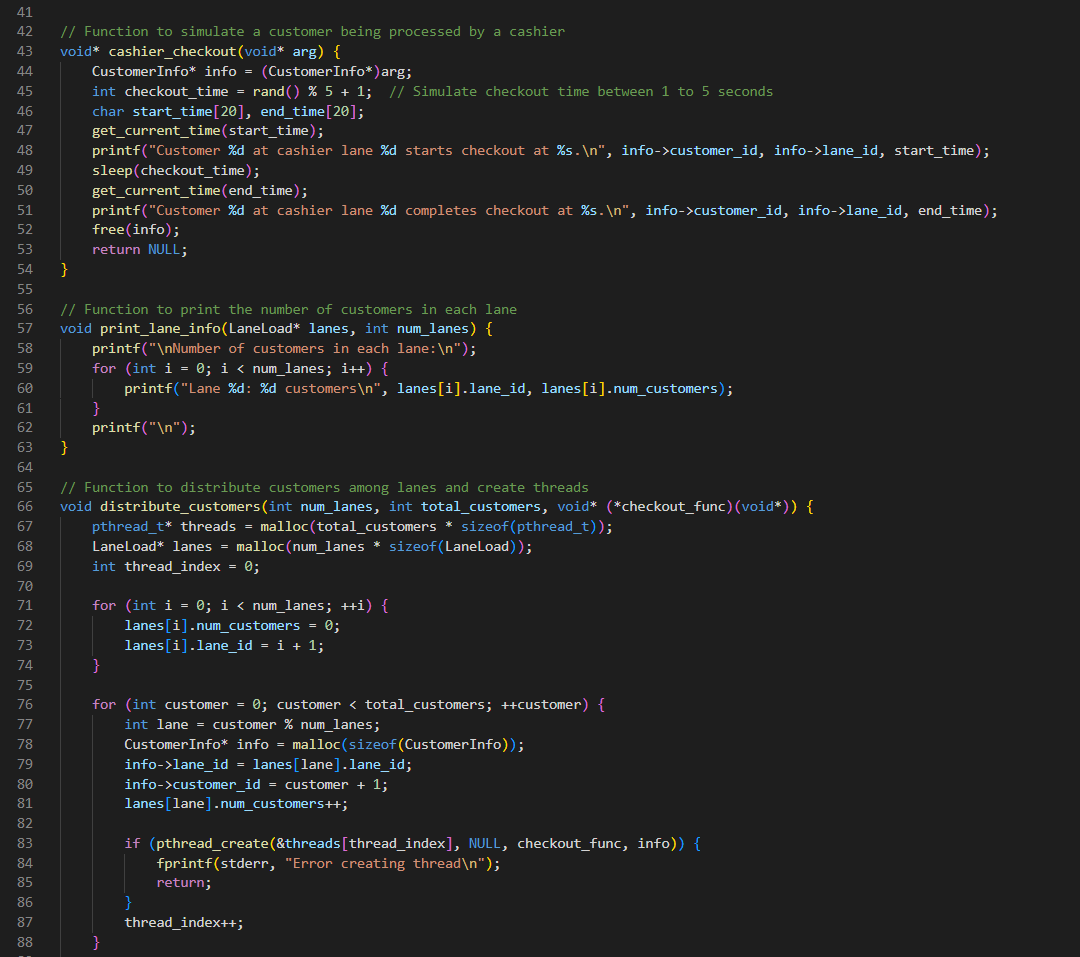
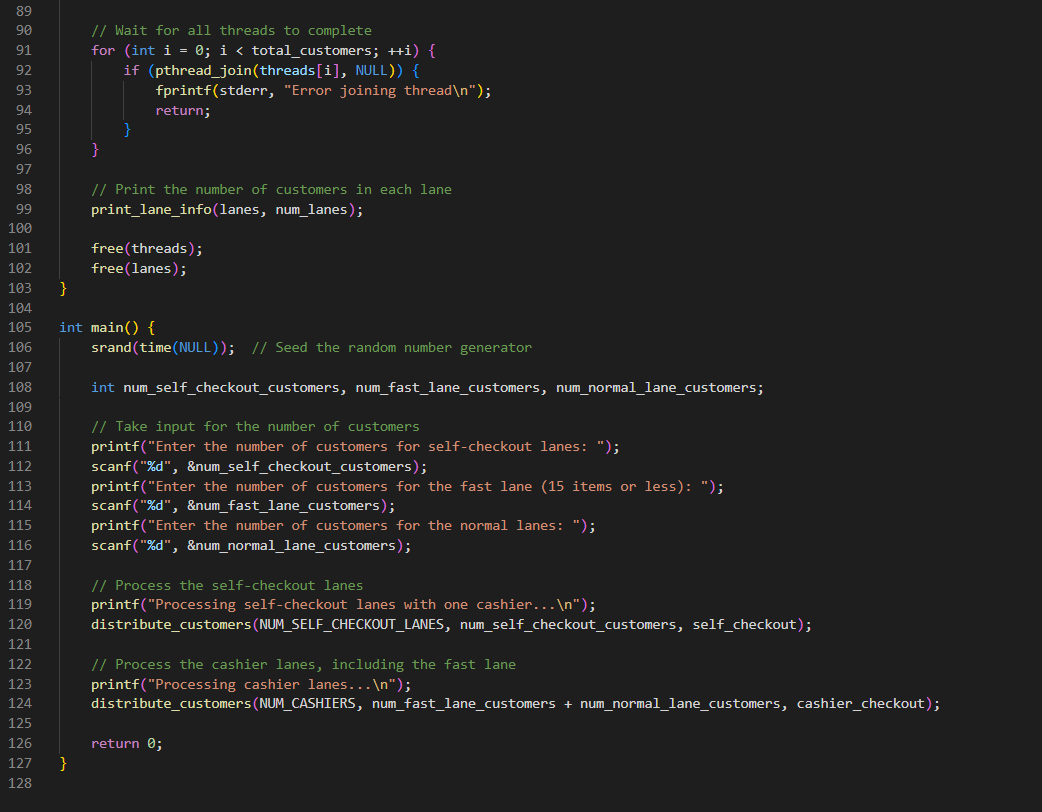
**‘cashier\_checkout’:**

* Simulates a customer being processed by a cashier.
* Randomly determines the checkout time (1-5 seconds).
* Prints the start and end times of the checkout process.

## Utility Functions

**‘print\_lane\_info’ :**

* Prints the number of customers assigned to each lane.

**distribute\_customers:**

* Distributes customers among lanes and creates threads for each customer.
* Ensures each customer is processed in parallel.
* Prints the number of customers in each lane after processing.

## Main function

* Seeds the random number generator.
* Takes input for the number of customers in each type of lane.
* Distributes customers to self-checkout lanes and cashier lanes.
* Processes the customers in parallel using threads.

# Execution flow

1. **Initialization**: Seed the random number generator and take user input for the number of customers in each type of lane.
2. **Self-Checkout Processing**: Distribute customers among self-checkout lanes and process them in parallel.
3. **Cashier Processing**: Distribute customers among cashier lanes (including the fast lane) and process them in parallel.
4. **Output**: Print the number of customers in each lane and the start and end times for each customer's checkout process.

## Key Points

* The program uses the POSIX thread library (pthread.h) for parallel processing.
* Each customer is processed in parallel by creating a separate thread for them.
* Mutexes are used to ensure correct synchronization within each lane.
* The program prints the start and end times of the checkout process for each customer to demonstrate parallelism.

# Output example

