UNIVERSIDADE SAO TOMAS DE MOCAMBIQUE

6/13/2024

Parallel Computing

**Group Members:**

Camira Chissico

Fernando Chikanda

Helder Alisio

# Supermarket Checkout Simulation

Analyzing and Optimizing Checkout Lanes Using Python

## Imports and Libraries

import threading  
import random  
import time  
  
- \*\*`threading`\*\*: Allows the creation and management of multiple threads, enabling concurrent execution.  
- \*\*`random`\*\*: Used to generate random numbers, simulating random processing times for customers.  
- \*\*`time`\*\*: Provides functions to handle time-related tasks, such as sleeping for a specific duration.

## Customer and Processing Functions

### self\_checkout\_cashier(lane\_id, customers)

def self\_checkout\_cashier(lane\_id, customers):  
 print(f"Self-checkout lane {lane\_id} starts processing.")  
 for customer in customers:  
 process\_customer(lane\_id, customer)  
 print(f"Self-checkout lane {lane\_id} has finished processing.")

Simulates a self-checkout lane processing customers. Iterates over each customer in the list and processes them.

### traditional\_cashier(lane\_id, customers)

def traditional\_cashier(lane\_id, customers):  
 print(f"Cashier at lane {lane\_id} starts processing.")  
 for customer in customers:  
 process\_customer(lane\_id, customer)  
 print(f"Cashier at lane {lane\_id} has finished processing.")

Simulates a traditional manned checkout lane processing customers. Similar to the self-checkout function but represents a traditional cashier.

### process\_customer(lane\_id, customer)

def process\_customer(lane\_id, customer):  
 processing\_time = random.uniform(0.5, 2.0)  
 print(f"Lane {lane\_id} processing customer {customer}...")  
 time.sleep(processing\_time)  
 print(f"Lane {lane\_id} finished processing customer {customer}.")

Simulates the processing of a customer. Generates a random processing time between 0.5 to 2.0 seconds. Pauses execution to simulate the processing duration.

## Main Function

### Main Function Definition

def main():  
 num\_customers = 20 # Total number of customers  
 num\_self\_checkout\_lanes = 4 # Number of self-checkout lanes  
 num\_traditional\_lanes = 3 # Number of traditional checkout lanes

Initializes the number of customers, self-checkout lanes, and traditional lanes.

### Customer Distribution

customers = list(range(1, num\_customers + 1))  
 random.shuffle(customers)  
  
 self\_checkout\_customers = [customers[i::num\_self\_checkout\_lanes] for i in range(num\_self\_checkout\_lanes)]  
 traditional\_checkout\_customers = [customers[i::num\_traditional\_lanes] for i in range(num\_traditional\_lanes)]

Generates a list of customer IDs and shuffles them to simulate random arrival patterns. Divides customers among self-checkout and traditional lanes using list slicing.

### Thread Creation and Execution

threads = []  
  
 for i in range(num\_self\_checkout\_lanes):  
 thread = threading.Thread(target=self\_checkout\_cashier, args=(i + 1, self\_checkout\_customers[i]))  
 threads.append(thread)  
 thread.start()  
  
 for i in range(num\_traditional\_lanes):  
 thread = threading.Thread(target=traditional\_cashier, args=(i + 1, traditional\_checkout\_customers[i]))  
 threads.append(thread)  
 thread.start()

Creates and starts threads for each lane, passing the lane ID and assigned customers as arguments.

### Thread Synchronization

for thread in threads:  
 thread.join()  
  
 print("All customers have been processed.")

Waits for all threads to finish using the `join()` method, ensuring the main program waits for thread completion.

### Execution Control

if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Ensures the main function runs only when the script is executed directly.