Ice Cream Parlor Simulation Report

1. Code Overview:

The provided C code simulates the operation of an ice cream parlor, where customers arrive, place orders, machines prepare ice creams, and an exit handler manages the closing of the parlor. The simulation involves multiple components, including machines, customers, toppings, flavours, and ingredient management.

2. Code Components:

2.1. Map Data Structure:

- A map data structure is implemented using a `Map` structure to manage ingredients, flavours, and toppings. This allows for efficient retrieval and updating of ingredient quantities.

2.2. Machine and Customer Structures:

- The code defines structures for `machine` and `customer` to store information about machine working times and customer orders.

2.3. Ingredient, flavours , and Topping Structures:

Separate structures (`flavour` and `topping`) are defined to represent flavors and toppings, each with its quantity or preparation time.

2.4. Order Structure:

- An `order` structure is used to represent the details of a customer's ice cream order, including flav flavours or, toppings, and the total number of toppings.

2.5. Thread Synchronization:

- The code employs thread synchronization mechanisms, such as mutexes (`mutex`, `mutex_s`, `mutex_q`, `mutex_map`, `mutex_swapn`) and condition variables (`cond_sec`, `cond_q`, `fastswap`, `verfastswap`, `cond_swapn`), to manage shared resources and coordinate the execution of threads.

3. Threading:

- Multiple threads are used to simulate concurrent processes, including threads for the counter (`counter_sec`), machines (`ma`), customer arrivals (`arrival`), the queue (`qu`), and an exit handler (`exit`).

4. Functionality:

4.1. counter Function:

- The `counter` function increments the simulation time (`sec`) every second and broadcasts the updated time using `cond_sec`.

4.2. machine_handler Function:

- The `machine_handler` function simulates machine operation, waiting for its assigned start time and then printing messages when it starts and stops working.

4.3. queue_hand Function:

--Queue Management: Utilizes `pthread_mutex_t mutex_q`, `pthread_cond_t cond_q`, and `pthread_cond_t fastswap` for synchronized queue operations.

- Machine Allocation: Waits for ingredients to be allocated to the ice cream before entering the queue. Uses `cond_sec` for signaling and waiting.

- Index-Wise Processing: Ensures orderly ice cream processing based on index using `fastswap` and `scheduleind`.

- Machine Selection: Waits for an available machine based on the machine's start and stop times. Uses mutex locks to prevent race conditions.

- Ice Cream Preparation: Initiates ice cream preparation, waits for the required time, and releases the machine (`marr[got]=0;`).

- Order Completion: Updates counters (`done[queu[cur_ind].a-1]++`) and broadcasts the completion of customer current ice-cream order using `cond_swapn`.

4.4. arrival_handler Function

- **Synchronization Variables:** Utilizes `pthread_cond_t cond_swapn` and `pthread_mutex_t mutex_swapn` for thread coordination.

-Index-Wise Processing: Ensures orderly customer processing based on index using `verfastswap` and `scheduler`.

- Arrival Time Check: Waits for the simulation time to reach or exceed the customer's arrival time (`arr[ind].t_arr`).

- Order Placement: Prints customer details and ice cream orders with color-coded output.

- Ingredient Allocation: Manages ingredient allocation using a thread-safe map (`MapT toppingsMap`).

- Simultaneous Processing: Broadcasts customer arrival and waits for machines. Processes customers simultaneously if capacity allows (`cind < spawned`).

- Order Completion: Waits for the `cond_swapn` condition variable, indicating completion of customer orders.

4.5. exit_handler Function:

- Simulation Time Check: Waits until the simulation time exceeds `maxtime+1` using `cond_sec`.

- Unserved Customer Check: Iterates through customers and prints a message for those who remain unserved due to machine unavailability.

-Parlor Closure: Prints a closing message and exits the simulation.

5. Conclusion:

The simulation adapts its customer spawning mechanism based on the availability of machines. While the default behavior is index-wise spawning when customers arrive at the same time, the program dynamically adjusts to machine availability. If a machine becomes available after customers have arrived, the allocation might not follow the index-wise order. This ensures flexibility and efficient resource utilization, balancing realism and computational efficiency in the simulation.

This adjustment clarifies that if machines become available after customers have already arrived, the allocation may not strictly adhere to index-wise order, providing a more accurate representation of a dynamic operational scenario.

QUESTION/ ANSWERS:

1. Bestfit machine -| 20 sec | 30 sec | 15 sec | 25 sec | 10 sec | and customer orders ice cream requires 15, 10 ,28 secs respectively.
There is case where 15-> 20 sec machine ,10-> 30sec machine ,28 -> waits for machine.
At the end 28-> isn't given to any machine and customer is not serviced.
So here, there is need to find Best-fit machine resulting in Minimizing Incomplete Order.

2. Assuming supplier takes finite time t_s to supply ingredients. We can add this t_s paramerter while consdering Best-fit machine ,and time particular flavour takes.
These parameters are to be used for acceptance/rejection process based on ingredient availability.

3. Even if there are sufficiently ingredients available,we must also consider Machine time and allocate it to each icecream(Best-fit) of individual customer.Keep updating Machine left time (Basically forecasting future Scheduling) as we assign Machine to particular ice-cream.
But if we know,at end Machine won't become available.I would simply reject order ->reducing Unserviced Orders.