CSE325: Operating Systems
Section: 02

Sweet Harmony

Group: 8

Submitted By

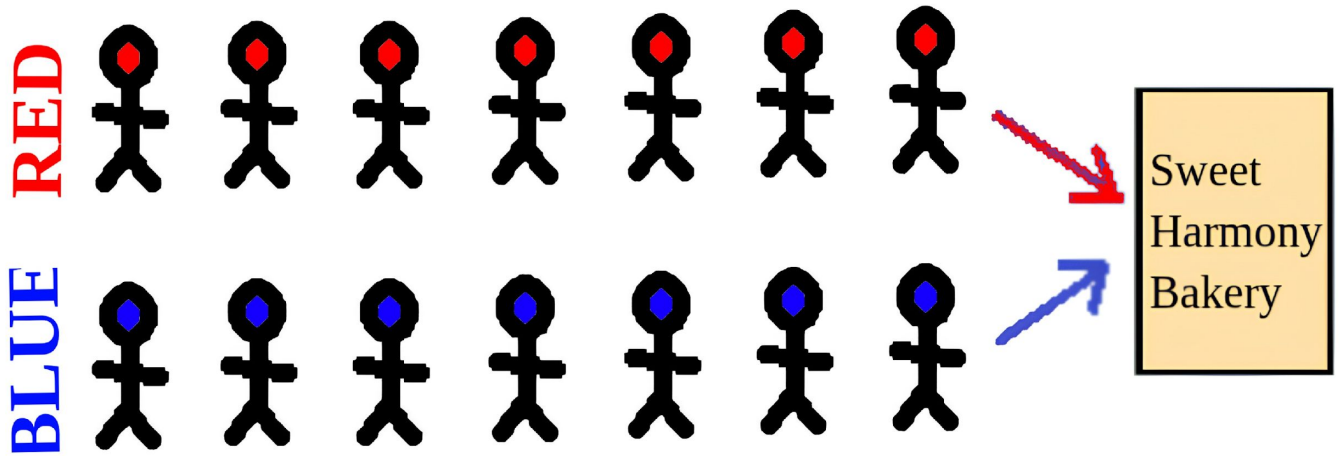
Muhammad Abdul Kayum
ID: 2023-2-60-082

Afia Mahpara
ID: 2023-2-60-011

Md. Hasib Ali
ID: 2023-3-60-186

Introduction & Problem Statement

- **Sweet Harmony:** A bakery with limited number of seats
- **Unique rule:** inside must always have equal numbers of red- and blue-clothed customers



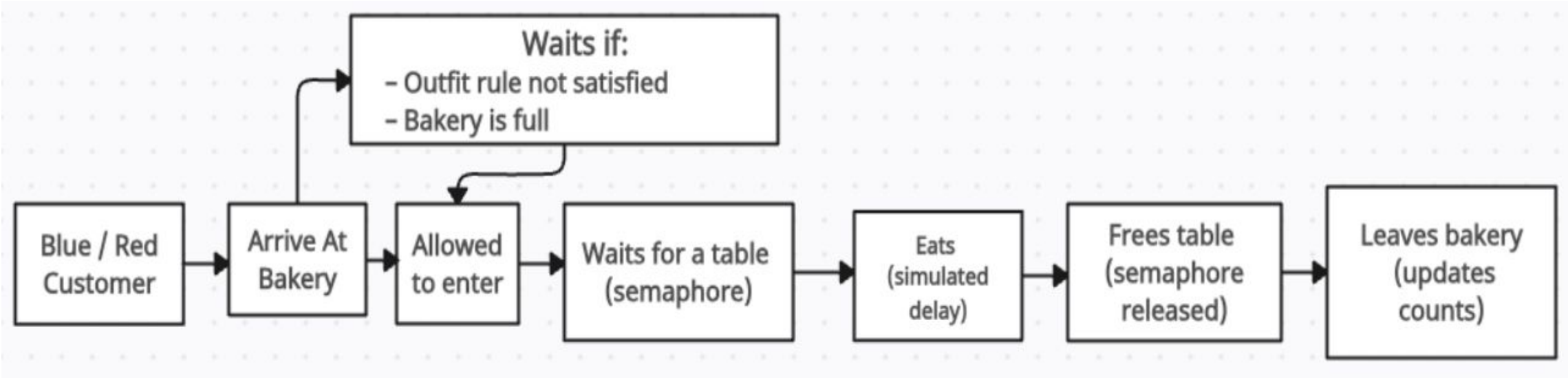
The core problem:

- Let customers enter **only if** outfit balance + table space allow
- Keep others waiting politely (queue) until the rule is satisfied
- Update counts (red, blue, free tables) **safely in parallel** — many customers act at once

Key Concepts Used

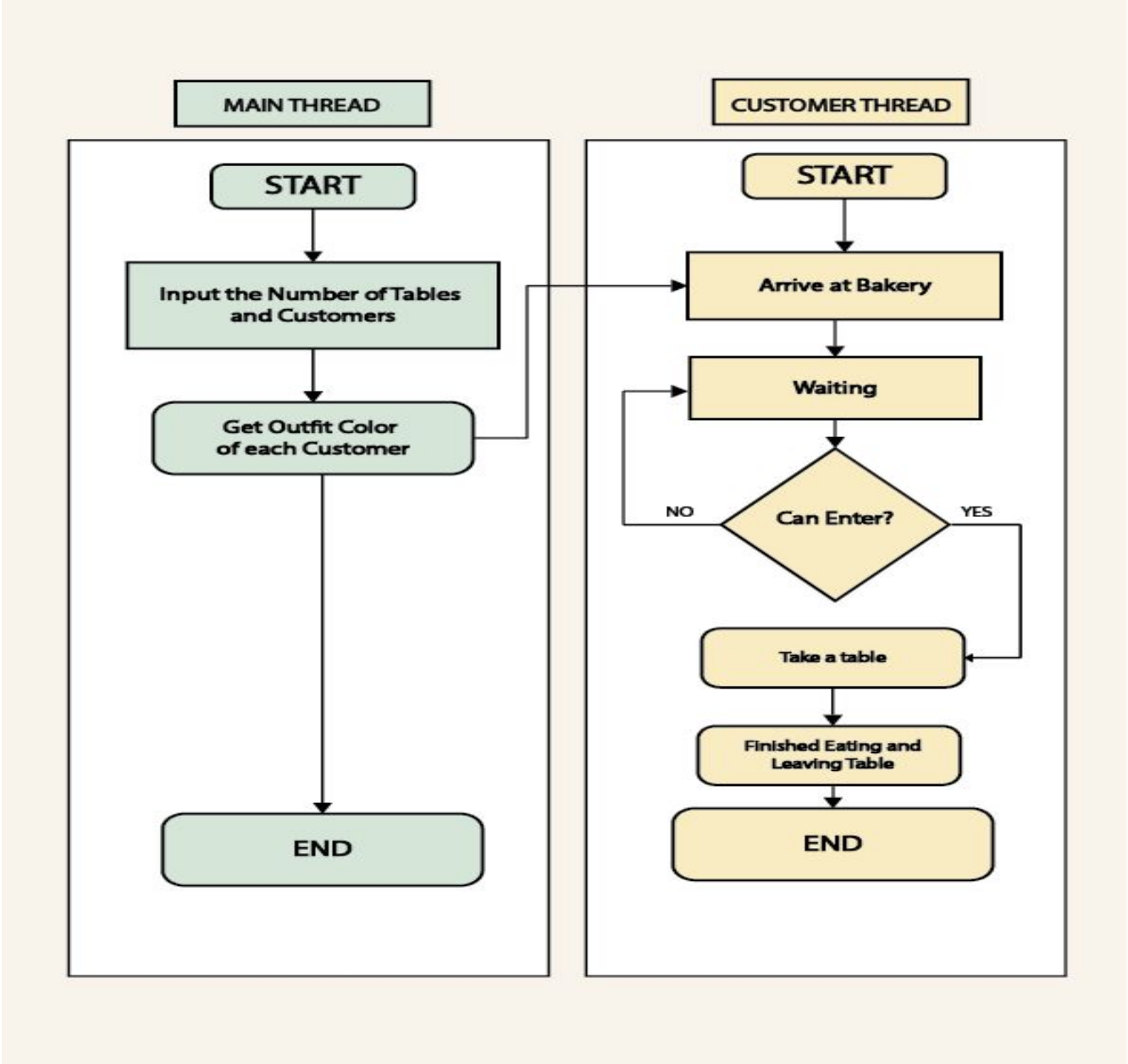
- **Threads:** Each customer is simulated as a separate thread to act independently, like real people arriving, waiting, eating, and leaving.
- **Mutex:** Used to safely update shared data like the number of red and blue customers inside, prevents race conditions. (pthread_mutex_t)
- **Semaphores:** Control the limited number of tables. A customer must wait if no table is free. (table_sem, etc)
- **Condition Variables:** Handle the outfit rule, customers wait if entering would break the red-blue balance. (pthread_cond_t)
- **Critical Sections:** Any part of the code that reads or modifies shared variables (like red count, blue count, or table access) is protected.

System Design & Visualization



- **Queue Style Leaving:** The customers (red/blue) follow FIFO style when leaving the bakery
- **Balanced entry logic:** A customer enters only if their outfit keeps red/blue counts equal and there's space inside.
- **Efficient resource control:** Uses a semaphore for tables and mutex + condition variable to manage entry rules.

Control Flow of the Code



Code: Entry and Waiting Logic

```
//Customer Arrives and Tries to Enter Bakery
pthread_mutex_lock(&count_mutex);
printf("[Queue] %s customer %d arrived, waiting to enter.\n",
       cust->color, cust->id);
// Wait until entry conditions are met
while (1) {
    int current_inside = red_count + blue_count;
    int can_enter = 0;
    if (current_inside < bakery_capacity) {
        if (strcmp(cust->color, "red") == 0)
            if (red_count <= blue_count) can_enter = 1;
        else
            if (blue_count <= red_count) can_enter = 1;
    }
    if (can_enter) break;
    printf("[Wait ] %s customer %d waiting outside...\n",
           cust->color, cust->id);
    pthread_cond_wait(&cond_equal, &count_mutex);
}
// Update counts after entry
if (strcmp(cust->color, "red") == 0) red_count++;
else blue_count++;
pthread_mutex_unlock(&count_mutex);
```

Code: Table Management and Leaving

```
// Wait for a free table
sem_wait(&table_sem);

//Customer is now seated
int eating_time = rand() % 3 + 1;
sleep(eating_time);

//Finished eating
printf("[Leave] %s customer %d done in %d s.\n",
       cust->color, cust->id, eating_time);
sem_post(&table_sem);    //frees the table

//Update counts & exit
pthread_mutex_lock(&count_mutex);
if (strcmp(cust->color, "red") == 0) red_count--;
else blue_count--;
pthread_cond_broadcast(&cond_equal);
pthread_mutex_unlock(&count_mutex);
```

Taking Input

```
=== Sweet Harmony Bakery Simulation ===
Enter the number of tables in the bakery (1-20): 1
Enter the total number of customers arriving (1-100): 4
Enter outfit color for each customer:
Customer 1:
  Enter outfit color (red/blue): red
Customer 2:
  Enter outfit color (red/blue): red
Customer 3:
  Enter outfit color (red/blue): blue
Customer 4:
  Enter outfit color (red/blue): blue
```

Problem - 1

```
=== Sweet Harmony Bakery Simulation ===
Enter the number of tables in the bakery (1-20): 2
Enter the total number of customers arriving (1-100): 5
Enter outfit color for each customer:
Customer 1:
  Enter outfit color (red/blue): blue
Customer 2:
  Enter outfit color (red/blue): blue
Customer 3:
  Enter outfit color (red/blue): red
Customer 4:
  Enter outfit color (red/blue): blue
Customer 5:
  Enter outfit color (red/blue): red
```

Problem - 2

Output and Simulation – problem 1

```
--- Starting Customer Arrivals (Bakery Capacity: 2, Tables: 1) ---
[Queue] red customer 1 arrived, waiting to enter.
[Enter] red customer 1 ENTERED bakery. (Red: 1, Blue: 0, Inside: 1/2 capacity).
[Queue] red customer 2 arrived, waiting to enter.
[Wait ] red customer 2 waiting outside (Red: 1, Blue: 0, Inside: 1/2 capacity).
[Queue] blue customer 3 arrived, waiting to enter.
[Enter] blue customer 3 ENTERED bakery. (Red: 1, Blue: 1, Inside: 2/2 capacity).
[Queue] blue customer 4 arrived, waiting to enter.
[Wait ] blue customer 4 waiting outside (Red: 1, Blue: 1, Inside: 2/2 capacity).
[Leave] red customer 1 finished eating, leaving table (2 sec).
[Exit ] red customer 1 LEFT bakery. (Red: 0, Blue: 1, Inside: 1/2 capacity).
[Wait ] blue customer 4 waiting outside (Red: 0, Blue: 1, Inside: 1/2 capacity).
[Enter] red customer 2 ENTERED bakery. (Red: 1, Blue: 1, Inside: 2/2 capacity).
[Leave] blue customer 3 finished eating, leaving table (3 sec).
[Exit ] blue customer 3 LEFT bakery. (Red: 1, Blue: 0, Inside: 1/2 capacity).
[Enter] blue customer 4 ENTERED bakery. (Red: 1, Blue: 1, Inside: 2/2 capacity).
[Leave] red customer 2 finished eating, leaving table (3 sec).
[Exit ] red customer 2 LEFT bakery. (Red: 0, Blue: 1, Inside: 1/2 capacity).
[Leave] blue customer 4 finished eating, leaving table (3 sec).
[Exit ] blue customer 4 LEFT bakery. (Red: 0, Blue: 0, Inside: 0/2 capacity).

=== Simulation Complete! All customers have visited Sweet Harmony. ===
```

Output and Simulation – problem 2

```
--- Starting Customer Arrivals (Bakery Capacity: 4, Tables: 2) ---
[Queue] blue customer 1 arrived, waiting to enter.
[Enter] blue customer 1 ENTERED bakery. (Red: 0, Blue: 1, Inside: 1/4 capacity).
[Queue] blue customer 2 arrived, waiting to enter.
[Wait ] blue customer 2 waiting outside (Red: 0, Blue: 1, Inside: 1/4 capacity).
[Queue] red customer 3 arrived, waiting to enter.
[Enter] red customer 3 ENTERED bakery. (Red: 1, Blue: 1, Inside: 2/4 capacity).
[Queue] blue customer 4 arrived, waiting to enter.
[Enter] blue customer 4 ENTERED bakery. (Red: 1, Blue: 2, Inside: 3/4 capacity).
[Queue] red customer 5 arrived, waiting to enter.
[Enter] red customer 5 ENTERED bakery. (Red: 2, Blue: 2, Inside: 4/4 capacity).
[Leave] blue customer 1 finished eating, leaving table (1 sec).
[Exit ] blue customer 1 LEFT bakery. (Red: 2, Blue: 1, Inside: 3/4 capacity).
[Enter] blue customer 2 ENTERED bakery. (Red: 2, Blue: 2, Inside: 4/4 capacity).
[Leave] blue customer 4 finished eating, leaving table (1 sec).
[Exit ] blue customer 4 LEFT bakery. (Red: 2, Blue: 1, Inside: 3/4 capacity).
[Leave] red customer 3 finished eating, leaving table (2 sec).
[Exit ] red customer 3 LEFT bakery. (Red: 1, Blue: 1, Inside: 2/4 capacity).
[Leave] red customer 5 finished eating, leaving table (1 sec).
[Exit ] red customer 5 LEFT bakery. (Red: 0, Blue: 1, Inside: 1/4 capacity).
[Leave] blue customer 2 finished eating, leaving table (1 sec).
[Exit ] blue customer 2 LEFT bakery. (Red: 0, Blue: 0, Inside: 0/4 capacity).

=== Simulation Complete! All customers have visited Sweet Harmony. ===
```

Conclusion

- Our simulation consistently keeps red = blue inside while respecting table limits.
- We have successfully implemented Semaphores, mutexes, and condition variables work together to solve a real concurrency puzzle.
- Properly Enter and Exit of Customer is handled with Queue.

Future work: Scale to larger crowds, introduce more outfit categories, or add a visual dashboard for live monitoring.

Real-world relevance: Similar patterns apply to load-balanced servers, ticket counters, or any shared-resource system.