ASSIGNMENT 2

Course: HY-345 Operating Systems
Winter Semester 2025

In this exercise you will become familiar with:

- The creation and management of threads (POSIX threads)
- And the use of POSIX semaphores for synchronizing multiple threads that cooperate in a shared environment.

Description — Restaurant and Customer Groups

We consider a restaurant where customers arrive in groups.
Each group wishes to sit together at the same table — splitting members is **not allowed**.

- The restaurant has **N tables**, each with a capacity of up to **X people**.
- **G** customer groups (threads) are created.
- Each group consists of **1 to X people** (random value).

The **waiter** (a separate thread) is responsible for monitoring the table status and assigning groups to available tables.

Program Logic

Group Behavior (Group Thread)

Each group is an independent thread and performs the following steps:

1. Arrives at the restaurant and reports its size (random from 1–X)
    - Group arrivals should occur in random order
2. Waits until a table becomes available that can fit all its members.
    - Waiting is implemented using a **semaphore** that is “woken up” by the waiter.
3. Sits at the assigned table.
4. Eats for a random amount of time (5–15 seconds).
5. Leaves and frees up the seats at the table.
6. Notifies the waiter that the table is now available.

Waiter Behavior (Waiter Thread)

The waiter is also a thread and continuously runs the following loop:

1. Monitors the waiting queue of groups that have not yet been seated.
2. For each waiting group, checks if there is a table with enough free seats.
3. If a suitable table is found:
    - "Reserves" the seats (updates the occupied variables for the table).
    - "Wakes up" the specific group.
4. If no group can be seated, waits until something changes (when a group leaves).
5. The program terminates when **all groups have eaten and left**.

Synchronization — Semaphores and Mutexes

The solution must use **both semaphores and mutexes**.

Execution Parameters

The program must accept three parameters from the command line:

./restaurant <num_tables> <num_groups> <table_capacity>

Example execution for *2 tables, 4 groups, 6 each table capacity*

```
./restaurant 2 4 6

[GROUP 3] Arrived with 3 people (after 2 sec)
[WAITER] Assigned group 3 (size=3) to Table 0 (3/6 occupied)
[WAITER] Groups waiting: None
[GROUP 3] Seated at Table 0 with 3 people
[GROUP 3] Eating (3 people) for 8 seconds...
[GROUP 1] Arrived with 2 people (after 6 sec)
[GROUP 0] Arrived with 6 people (after 6 sec)
[WAITER] Assigned group 1 (size=2) to Table 0 (5/6 occupied)
[WAITER] Groups waiting: 0
[GROUP 1] Seated at Table 0 with 2 people
[GROUP 1] Eating (2 people) for 14 seconds...
[WAITER] Assigned group 0 (size=6) to Table 1 (6/6 occupied)
[WAITER] Groups waiting: None
[GROUP 0] Seated at Table 1 with 6 people
[GROUP 0] Eating (6 people) for 13 seconds...
[GROUP 2] Arrived with 3 people (after 8 sec)
[GROUP 3] Left Table 0 (2/6 occupied)
[WAITER] Assigned group 2 (size=3) to Table 0 (5/6 occupied)
[WAITER] Groups waiting: None
[GROUP 2] Seated at Table 0 with 3 people
[GROUP 2] Eating (3 people) for 7 seconds...
[GROUP 2] Left Table 0 (2/6 occupied)
[GROUP 0] Left Table 1 (0/6 occupied)
[GROUP 1] Left Table 0 (0/6 occupied)

All groups have left. Closing restaurant.
```

Evaluation Criteria

1. Correct synchronization (no race conditions).
2. Proper use of mutexes and semaphores.
3. Realistic simulation (correct synchronization between waiter and groups).
4. Console output should be clear and provide all necessary information.
5. Completeness of comments and README.

Submission

The submission will be done through **E-Learn**.

You must create a **zip file** that contains:

- Your **code**, including explanatory comments.
- A **Makefile** that supports the following commands:
  - make all → compiles and produces the executable
  - make clean → deletes all generated files
- A **README** describing your implementation and any notes you consider important.

A suggested structure for your deliverables:

- main.c
- restaurant.c
- restaurant.h
- Makefile
- README.md

Notes

- This exercise is **strictly individual**. Submitted files will be checked by an **anti-plagiarism system**. Any detected plagiarism will result in **automatic zero**.
- At the beginning of your code, include in comments your **full name** and **student ID number**.
- Keep your code tidy and include comments explaining its functionality where needed.
- The use of code **generated by AI is strictly prohibited**.