Baker Project Design Doc

Overview

This project simulates a group of bakers working in a shared kitchen to prepare various recipes using limited resources. Each baker operates as a separate thread, competing for access to shared resources like the pantry, refrigerator, utensils, and oven. The program ensures fairness and synchronization through semaphores, allowing bakers to complete their tasks efficiently.

Baker Logic

Random Recipe Selection

Each baker is assigned a random recipe to prepare. This ensures a fair distribution of workload and avoids all bakers working on the same recipe simultaneously.

1. **Recipe Assignment**:
   - A predefined array lists all available recipes.
2. **Randomization**:
   - The random number generator is seeded uniquely for each baker using `srand(time(NULL) + baker_id)`.
   - The recipe is chosen with `rand() % num_recipes`, ensuring randomness.
3. **Feedback**:
   - The baker prints the selected recipe to the terminal.

Acquiring Ingredients from Pantry and Refrigerator

Bakers collect the ingredients for their assigned recipes by accessing the pantry and refrigerator simultaneously. Semaphores are used to control access to these shared resources.

1. **Ingredient Collection**:

   - Bakers track their required ingredients using an array (`needed_ingredients`).
   - A status array (`is_acquired`) ensures no duplication of already-acquired ingredients.

2. **Simultaneous Access**:

- Bakers alternate between attempting to access the pantry and refrigerator, acquiring any available ingredient.
- Semaphores (`semop`) ensure that only one baker can use a resource at a time.

3. **Efficient Resource Usage**:

    - If the pantry is busy, a baker can check and grab ingredients from the refrigerator (or vice versa), avoiding idle waiting.

4. **Resource Release**:

    - Once an ingredient is acquired, the baker releases the semaphore, making the resource available to others.

Mixing Ingredients with Utensils

Bakers must acquire **all three utensils**—a bowl, spoon, and mixer—at the same time to start mixing their ingredients. If any utensil is unavailable, the baker waits until all three can be acquired together.

1. **Utensil Operations**:

    - Semaphore operations for the bowl, spoon, and mixer are combined into a single `semop` call, ensuring atomic acquisition.

2. **Mixing Process**:

    - After acquiring all utensils, the baker mixes the ingredients, simulating the process with a `printf` statement.

3. **Releasing Utensils**:

    - Once mixing is complete, the baker releases all three utensils simultaneously, allowing other bakers to proceed.

Baking with the Oven

Once the ingredients are mixed, the baker uses the oven to bake the recipe. The oven is a shared resource, and only one baker can use it at a time.

1. **Acquiring the Oven**:
    - The `use_resource` function ensures exclusive access to the oven via semaphore.
2. **Simulating Baking**:
    - The baker simulates baking by printing a message to the terminal, indicating progress.
3. **Releasing the Oven**:
    - After baking, the `release_resource` function unlocks the oven, allowing the next baker to use it.

Ramsied

During a randomly selected baker loop, a randomly selected baker will be "Ramsied" once every time the program runs.

1. **Selecting the Baker and Loop**
    - In the `main` function, the `rand()` function is called twice. The first call randomly selects a baker from the available bakers, and the second call picks a loop iteration between 0 and 5, corresponding to the total number of recipes
2. **Catching the Ramsied Baker**
    - During each recipe loop, after the baker gathers their ingredients, an if statement checks if the current baker and loop iteration match the randomly selected ones. If they do, the Ramsied function is triggered.
3. **Ramsied Logic**
    - When a baker is "Ramsied", they must restart their recipe from the `grab_ingredients` function. This simulates losing all their ingredients and releasing any held semaphores, without affecting other recipes.

Key Features

- **Concurrency**:

  - Each baker runs in a separate thread, simulating a real-world kitchen scenario.
- **Synchronization**:
  - Semaphores ensure fair and efficient access to shared resources.
- **Randomization**:
  - Recipes are assigned randomly to balance workload.
- **Real-Time Feedback**:
  - Bakers provide live updates in the terminal, showing progress through the recipe preparation steps.

How to Run

1. Compile the program using `gcc`:

   ```
   gcc -pthread -o bakers bakers.c
   ```

2. Run the executable:

   ```
   ./bakers
   ```

3. Enter the number of bakers when prompted:

   ```
   Enter the number of bakers: 3
   ```

4. Watch the terminal for live updates as the bakers complete their tasks!