Exercise: Lamport's Bakery Algorithm (Software Synchronization) [1em] Operating Systems / Concurrency Module

Objective

In this exercise, students will implement and test the **Lamport's Bakery Algorithm** for mutual exclusion. This algorithm ensures that only one process enters its critical section at a time while maintaining fairness and progress among all competing processes.

Learning Outcomes

By the end of this exercise, students should be able to:

- Explain how Lamport's Bakery Algorithm guarantees **Mutual Exclusion**, **Progress**, and **Bounded Waiting**.
- Implement synchronization logic for multiple concurrent processes using shared variables.
- Understand the concept of distributed numbering (ticket system) for fair process ordering.

Algorithm Summary

The Bakery Algorithm works like taking a ticket at a bakery shop:

1. Each process takes a number that is one greater than all existing numbers.
2. The process waits until its number is the smallest (if equal, the lower ID wins).
3. When done, it sets its number back to zero to allow others to proceed.

This approach ensures fairness and prevents starvation.

Exercise Tasks

1. Study the given Go code skeleton.
2. Complete the missing parts marked as `TODO`.
3. Run the code using different values of `N` (number of tasks) and observe the output.
4. Verify that at no time do two tasks enter the critical section simultaneously.

Go Code Skeleton

```
// Skeleton code for Lamport's Bakery Algorithm (Exercise Template)
// Students should complete the missing parts marked with TODO comments.

package main

import (
"fmt"
"runtime"
"time"
)

const N = 3 // number of processes

var choosing [N]bool
var number [N]int

// helper function to get the maximum ticket number
func maxNumber() int {
max := 0
for i := 0; i < N; i++ {
if number[i] > max {
max = number[i]
}
}
return max
}

func Task(id int) {
for {
fmt.Printf("Task-%d: Begin Section\n", id)

```
    // Entry Section
    // TODO: 1. Set choosing[id]
    // TODO: 2. Assign number[id]
    // TODO: 3. Set choosing[id]

    // TODO: 4. Wait for all other processes (follow Bakery algorithm conditions)

    // Critical Section
    fmt.Printf(">>> Task-%d: Critical Section <<<\\n", id)
    time.Sleep(time.Second)

    // Exit Section
    // TODO: 5. Set number[id]

    fmt.Printf("Task-%d: Remainder Section\\n", id)
    time.Sleep(2 * time.Second)
}
```

}

func main() {
```

```
for i := 0; i < N; i++ {
go Task(i)
}
select {} // keeps the program running
}
```

Listing 1: Lamport's Bakery Algorithm Skeleton in Go

Deliverables

- Completed Go code with all TODO parts implemented.
- A short written explanation (5–10 lines) describing how the algorithm ensures fairness.
- Screenshot or terminal log showing sample output where each task enters its critical section in order.

Note: This exercise should be performed in pairs. Students are encouraged to test edge cases (e.g., `N=5`) and observe scheduling behavior.