Mini EP8 - Simulando Sistemas Distribuídos

Tarefa 1 - Introdução ao Maelstrom

No Maelstrom, um nó é representado pela classe `Node` e abstrai o nó de sistemas distribuídos (uma unidade computacional que faz parte de um sistema maior - como um cluster):

```
type Node struct {
    mu sync.Mutex
    wg sync.WaitGroup

    id        string
    nodeIDs   []string
    nextMsgID int

    handlers  map[string]HandlerFunc
    callbacks map[int]HandlerFunc

    // Stdin is for reading messages in from the Maelstrom network.
    Stdin io.Reader

    // Stdout is for writing messages out to the Maelstrom network.
    Stdout io.Writer
}
```

O método `Node.Reply()` configura a `source` e o `destination` automaticamente para o envio de mensagens, evitando assim, que o usuário tenha que configurar a rede do "sistema distribuído".

Por fim, o método `Node.Run()` simula a execução da rotina principal que processa continuamente as mensagens recebidas.

Tarefa 2 - Concorrência, Paralelismo e o Teorema CAP

A nova versão do código resolve o problema de múltiplos acessos simultâneos (que acarreta em IDs duplicados) a um recurso compartilhado do nó (`counter`) ao criar uma região crítica:

```
mu.Lock()
body["id"] = getCounter()
incrementCounter()
mu.Unlock()
```

No entanto, não funciona para múltiplos nós pois cada nó possui uma variável `counter`.

Para garantir as propriedades de disponibilidade e tolerância a partições (AP), podemos usar um gerador global de IDs que combina o ID do nó atual e o UNIX

timestamp atual, que dispensa que os nós comuniquem entre si:

```
id := fmt.Sprintf("%s-%d", n.ID(), time.Now().UnixNano())
body["id"] = id
```