Questionário 12

✂--- Problemas Clássicos de Sincronização* ---✂

1. Usando semáforos, escreva o pseudocódigo de um sistema produtor/consumidor com dois buffers limitados organizado na forma X => B_1 => Y => B_2 => Z, onde X, Y e Z são tipos de processos e B_1 e B_2 são buffers independentes com capacidades N_1 e N_2, respectivamente, inicialmente vazios. Os buffers são acessados unicamente através das operações insere(B_i, item) e retira(B_i , item) (que não precisam ser detalhadas). O número de processos X, Y e Z é desconhecido. Devem ser definidos os códigos dos processos X, Y e Z e os semáforos necessários, com seus significados e valores iniciais.

2. O trecho de código a seguir apresenta uma solução para o problema do jantar dos filósofos, mas ele contém um erro. Explique o código e explique onde está o erro e porque ele ocorre. A seguir, modifique o código para que ele funcione corretamente.

```
#define N 5
sem_t garfo[N] ; // 5 semaforos iniciados em 1
void filosofo (int i) {
    while (1) {
        medita () ;
        sem_down (garfo [i]) ;
        sem_down (garfo [(i+1) % N]) ;
        come () ;
        sem_up (garfo [i]) ;
        sem_up (garfo [(i+1) % N]) ;
    }
}
```

3. Suponha três robôs (Bart, Lisa, Maggie), cada um controlado por sua própria thread. Você deve escrever o código das threads de controle, usando semáforos para garantir que os robôs se movam sempre na sequência Bart → Lisa → Maggie → Lisa → Bart → Lisa → Maggie → ⋯, um robô de cada vez. Use a chamada move() para indicar um movimento do robô. Não esqueça de definir os valores iniciais das variáveis e/ou dos semáforos utilizados. Soluções envolvendo espera ocupada (busy wait) não devem ser usadas.

4. O *Rendez-Vous* é um operador de sincronização forte entre dois processos ou threads, no qual um deles espera até que ambos cheguem ao ponto de encontro (rendez-vous, em francês). O exemplo a seguir ilustra seu uso:

```
Processo A
    A1 () ;
    rv_wait (rv) ;
    A2 () ;
    rv_wait (rv) ;
    A3 () ;
```

```
Processo B
    B1 () ;
    rv_wait (rv) ;
    B2 () ;
    rv_wait (rv) ;
    B3 () ;
```

*Baseado no conteúdo do livro "Sistemas Operacionais: Conceitos e Mecanismos" do Prof. Carlos A. Maziero (UFPR).

Considerando a relação a → b como "a ocorre antes de b" e a relação a || b como "a e b ocorrem sem uma ordem definida", temos as seguintes restrições de sincronização:

- $\forall(i,j)$, $A_i \rightarrow B_{j>i}$ e $B_i \rightarrow A_{j>i}$ (imposto pelo *Rendez-Vous*)
- $\forall(i,j)$, $A_i \rightarrow A_{j>i}$ e $B_i \rightarrow B_{j>i}$ (imposto pela execução sequencial)
- $\forall(i,j)$, $A_i || B_{j=i}$ (possibilidade de execução concorrente)

Escreva o pseudo-código necessário para implementar *Rendez-Vous*, usando semáforos ou mutexes. Não esqueça de inicializar as variáveis e semáforos utilizados. Soluções que incorrem em espera ocupada (busy wait) não devem ser usadas.

```
// estrutura que representa um RV
typedef struct rv_t {
    ... // completar
} rv_t ;
// operador de espera no RV
void rv_wait (rv_t *rv) {
    ... // completar
}
// inicializacao do RV
void rv_init (rv_t *rv) {
    ... // completar
}
```

5. Uma Barreira é um operador de sincronização forte entre N processos ou threads, no qual eles esperam até que todos cheguem à barreira. O exemplo a seguir ilustra seu uso:

```
Processo A
    A1 () ;
    barrier_wait (b) ;
    A2 () ;
    barrier_wait (b) ;
    A3 () ;
```

```
Processo B
    B1 () ;
    barrier_wait (b) ;
    B2 () ;
    barrier_wait (b) ;
    B3 () ;
```

```
Processo C
    C1 () ;
    barrier_wait (b) ;
    C2 () ;
    barrier_wait (b) ;
    C3 () ;
```

```
Processo D
    D1 () ;
    barrier_wait (b) ;
    D2 () ;
    barrier_wait (b) ;
    D3 () ;
```

Considerando a relação $a \rightarrow b$ como "a ocorre antes de b" e a relação $a||b$ como "a e b ocorrem sem uma ordem definida", temos as seguintes restrições de sincronização:

- $\forall(i,j)$, $X \neq Y$, $X_i \rightarrow Y_{j>i}$ (imposto pela barreira)
- $\forall(i,j)$, $X_i \rightarrow X_{j>i}$ (imposto pela execução sequencial)
- $\forall(i,j)$, $X \neq Y$, $X_i||Y_{j=i}$ (possibilidade de execução concorrente)

Escreva o pseudocódigo necessário para implementar barreiras para N processos, usando semáforos ou mutexes. Não esqueça de inicializar as variáveis e semáforos utilizados. Soluções que incorrem em espera ocupada (busy wait) não devem ser usadas.

```
// estrutura que representa uma barreira
typedef struct barrier_t {
    ... // completar
} barrier_t ;
// operador de espera na barreira
void barrier_wait (barrier_t *rv) {
    ... // completar
}
// inicializacao da barreira para N processos
void barrier_init (barrier_t *rv) {
    ... // completar
}
```