Atividades

5.2) Porque ocorre deadlock?

Ocorre deadlock porque os semáforos foram usados incorretamente:

```
sem_wait(&full); //Espera o buffer estar cheio
buffer[index++] = i;
sem_post(&empty); //Sinaliza que há um espaço vazio
```

Esse código faz o produto esperar (sem_wait(&full)) por um item cheio no buffer, o que é responsável do consumidor, não do produtor. Isso inverte a lógica correta. O correta seria o produto esperar por espaço vazio e sinalizar que o buffer tem um item cheio.

A correção:

```
sem_wait(&empty); //Espera espaço vazio
//insere item no buffer
sem_post(&full); //Sinaliza item disponível
```

5.4) Em um cenário com múltiplos consumidores, por que alguns threads podem ficar em starvation?

No código do arquivo:

```
public synchronized void adicionar(int item) {
    fila.add(item);
    notify();
}
```

Se houver vários consumidores esperando, o uso de notify() acorda somente **um thread**, sem garantir ordem justa. Se o thread acordado não for o que está esperando há mais tempo, ele pode consumir repetidamente, e outro podem nunca ser acordados, resultando em **startion**.

Solução:

Substituir notify() por notifyAll().

```
    fila.add(item);
    notifyAll(); //Evita starvation para múltiplos consumidores
}
```

Assim, todos os consumidores são acordados e concorrem justamente.

5.5) Por que ocorre deadlock?

No problema do Jantar dos Filósofos, cada filósofo pega um garfo (semáforo) e espera pelo próximo. Se todos fizerem isso ao mesmo tempo, todos pegam um garfo e ficam esperando o outro, resultando em deadlock circular.

Solução Clássica: Ordenar a aquisição dos garfos para evitar ciclos.

Exemplo:

```
if(id % 2 == 0) {
    sem_wait(&garfo[(id + 1) % NUM_FILOSOFOS]); //NUM_FILOSOFOS 5
    sem_wait(&garfo[id]);
}

else {
    sem_wait(&garfo[id]);
    sem_wait(&garfo[(id + 1) % NUM_FILOSOFOS]);
}
```

Assim, a ordem de aquisição varia e evita o bloqueio circular.

6.1) O que acontece se o semáforo mutex for removido no código do Barbeiro Sonolento?

Sem o `mutex`, várias threads poderiam acessar e modificar a variável cadeiras_livres **simultaneamente**, causando **condições de corrida**. Isso resultaria em contagem incorreta de cadeiras disponíveis, podendo aceitar mais clientes do que o limite, ou perder clientes por erro de contagem.

6.3) Por que usar `notify()` em vez `notifyAll()` no código da Fila Java pode causar starvation?

notify() acorda **somente uma thread**, e essa não pode não ser a que pode prosseguir. Em filas com múltiplos consumidores, isso pode levar a **threads sendo ignoradas repetidamente**, enquanto outras continuam sendo ativadas.

Com notifyAll(), **todas as threads são acordadas** e podem competir de forma justa para acessar a fila.