

4) (3,0) O código abaixo tenta resolver o problema do “Produtor-Consumidor” em um buffer compartilhado, usando as primitivas sleep/wakeup.

a) Qual o problema com essa solução?

```
#define N 100                /* number of slots in the buffer */
int count = 0;              /* number of items in the buffer */

void producer(void) {
    while (true){
        produce_item();      /* generate next item */
        if (count == N)
            sleep();          /* if buffer is full, go to sleep */
        enter_item();         /* put item in buffer */
        count = count + 1;    /* increment count of items in buffer*/
        if (count >= 1){      /* was buffer empty? */
            wakeup(consumer);
        }
    }
}

void consumer(void){
    while (true){
        if (count == 0)
            sleep();          /* if buffer is empty, got to sleep */
        remove_item();        /* take item out of buffer */
        count = count - 1;    /* decrement count of items in buffer*/
        if (count <= N-1){
            wakeup(producer); /* was buffer full? */
        }
        consume_item();       /* print item */
    }
}
```

- Não garante exclusão mútua no acesso à variável compartilhada “count” ... condição de corrida!

- Deadlock: Buffer está vazio. Consumidor testa o valor de count, que é zero, mas não tem tempo de executar sleep, pois o escalonador selecionou agora produtor. Este produz um item, insere-o no buffer e incrementa count. Como count = 1, produtor chama wakeup para acordar consumidor. O sinal não tem efeito (é perdido), pois o consumidor ainda não está logicamente adormecido. Consumidor ganha a CPU, executa sleep e vai dormir. Produtor ganha a CPU e, cedo ou tarde, encherá o buffer, indo também dormir. Ambos dormirão eternamente.

b) Corrija a solução alterando o código apresentado no quadro abaixo, usando os semáforos s1, s2 e s3 (foram retiradas as chamadas sleep/wakeup). Considere que as funções enter_item() e remove_item(), ambas acessam todos os índices de inserção e remoção do buffer (compartilhados), ou seja, esses índices são acessados de forma concorrente.

<pre>#define N 100 int count = 0; semaphore s1 = <u>1</u> ; semaphore s2 = <u>N</u> ; semaphore s3 = <u>0</u> ; void producer(void) { while (true){ produce_item(); down(s2); //if (count == N){ down(s1); enter_item(); up(s1); //} up(s3); //count = count + 1; //if (count >= 1){ } } }</pre>	<pre>void consumer(void){ while (true){ down(s3); //if (count == 0) { down(s1); remove_item(); up(s1); //} //count = count - 1; up(s2); // if (count <= N-1){ // } consume_item(); } }</pre>
---	--