Lista 3 – Computação Concorrente

Nome: Gabriel Almeida Mendes
DRE: 117204959

Tarefa da Questão 1

a) Essa solução garante as condições do problema, ausência de condição de corrida e de deadlock?
R= Sim, a solução garante todas essas especificações. Analisando em partes o código vemos que: O semáforo rec, começando com valor 1, faz com que as funções threads não executem simultaneamente. Isso ocorre assim porque, no início, apenas uma thread de cada tipo agirá devido ao primeiro sem_wait(&rec), seguidamente, as threads (no máximo uma de cada tipo) entrarão no if fazendo com que apenas uma possa prosseguir. No fim do código de cada uma das funções threads, a thread que conseguir passar lançará um sem_post(&post) liberando qualquer outra thread do mesmo tipo de executar e só permitirá que outra thread de tipo diferente execute quando não houver mais thread do mesmo tipo executando. A justificativa disso se vem do fato de que, o sem_post(&rec) está dentro de um if, sendo a variável residente de cada uma das funções threads ser o tipo de variável utilizada por este tipo de thread.

b) Essa solução pode levar as threads a um estado de starvation (espera por um tempo muito longo para serem atendidas)?
R= Sim, porque existe uma espera muito grande entre uma thread terminar de executar e passar para a seguinte. Isso pode ser visualizado na execução do programa, quando existem blocos enormes em que apenas uma única thread executa até ir para a próxima ou até mesmo ficar para sempre com a thread atual rodando e a outras nunca iniciarem.

c) O que aconteceria se o semáforo rec fosse inicializado com 0 sinais?
R= No caso do valor inicial do semáforo for 0, vai acontecer da primeira chamada de qualquer uma das thread ser bloqueada quando a operação sem_wait(&rec) aparecer no primeiro if de casa uma das funções threads. Isso vai resultar num “congelamento” do programa, impedindo a função thread de prosseguir e terminar, resultando numa não finalização do programa. Pelo menos até que o valor seja alterado para um valor maior que 0 algo que não ocorre no programa.

d) O que aconteceria se o semáforo rec fosse inicializado com mais de um sinal?
R= No caso do valor inicial do semáforo for maior que 0 (ou seja, mais de um sinal). É como se meio que ocorresse uma permissão para que todas as threads sejam executadas todas ao mesmo tempo. O objetivo do semáforo é fazer com que uma thread não possa executar até que a anterior termine, fazendo com que o contrário não seja o desejável porque mataria a ideia de exclusão mutua e estragaria o programa. A ideia do rec é permitir que apenas uma thread acesse de uma só vez, se ela for maior que 1, todas as threads entram juntas.

Questão 2

a) Onde esta(ão) o(s) erro(s) no código?
R= O código está com dois erros e ambos estão na thread consumidora. O primeiro é referente a chamada da função retira_item, porque quando ela é chamada, o resultado tem que ser reatribuído a variável item, corrigindo ela ficaria: item = retira_item(&item). Segundo, o maior problema está na sincronização do semáforo, o sem_post(&s) está sendo chamado muito cedo e isso tá fazendo a produtora liberar o semáforo d sem que d estivesse sido preso anteriormente, aí quando era pra prendê-lo mesmo ela está sendo liberada direto.

b) Proponha uma maneira de resolvê-lo(s) mantendo as funções produz item e consome item fora da exclusão mútua.
R= Embaixo está uma maneira alternativa de resolver esse problema:

```
void *cons(void *args) {
        int item;
        sem_wait(&d);
        while(1) {
                sem_wait(&s);
                item = retira_item(&item);
                n--;
                if(n==0) sem_wait(&d);
                sem_post(&s);
                consome_item(item);
        }
}
```

Tarefa da Questão 4

(a) Qual é a finalidade dos semáforos s, x e h dentro do código? Esses semáforos foram inicializados corretamente?
R= O semáforo x apenas cumpre a função de servir como um mecanismo de exclusão mútua para essas funções auxiliares quando as mesmas forem chamada pelas funções threads, pois essas chamadas ocorreram mais de uma vez e é inadmissível que mais de uma thread acessem ao mesmo tempo a seção crítica do código. Devido a isso a inicialização em 1 está correta para o x. O semáforo h tem a função crucial de realizar os bloqueios das threads na função wait(), através da operação sem_wait(&h), até que ela seja liberada por um notify() ou notifyAll(), através da operação sem_post(&h). Por último, o semáforo s funciona em conjunto com o semáforo h, porque a função dele é garantir que uma thread que estava em espera saiu quando for chamada uma função notify() ou notifyAll(). Se esse mecanismo conjunto não existir pode acontecer de um thread entrar e ser liberada logo em seguida, por isso a inicialização com 0 está correta porque é quando for chamada a operação sem_wait() está consiga bloquear as threads.

(b) Essa implementação está correta? Ela garante que a semântica das operações wait, notify e notifyAll está sendo atendida plenamente?

R= Sim, a implementação está correta. As funções auxiliares wait, notify e notifyAll simulam perfeitamente os métodos/operações de mesmo nome ao se utilizarem dos semáforos dentro de suas implementações. A função wait consegue bloquear as threads até que estás sejam liberadas pelas funções notify() e notifyAll(). Como os semáforos funcionam já foi explicado na questão 4a.

(c) Existe a possibilidade de acúmulo indevido de sinais nos semáforos s, x e h?
R= Não, porque para toda vez que uma operação sem_wait() é chamada em alguma função, existem uma operação sem_post() que lhe complementa e finaliza. Dessa forma não tem como ter acúmulo de sinais.