Módulo 3 - Lista de Exercícios 3 (2020/2 REMOTO)

Computação Concorrente (MAB-117)
Prof. Silvana Rossetto

[1]DCC/IM/UFRJ — 25 de maio de 2021

Questão 1 (2,5 pontos) Carros vindos do norte e do sul chegam em uma ponte de uma única pista, apenas carros na mesma direção podem compartilhar a ponte, carros em direção oposta devem esperar até a ponte ficar liberada. O último carro a cruzar a ponte libera a travessia para os carros aguardando na direção oposta. O algoritmo proposto abaixo resolve esse problema usando semáforos (código para os carros do sul e do norte). Ele faz uso das funções `espera` (faz os carros esperarem) e `libera` (libera os carros).

```
//variaveis globais
sem_t em, dorme, e; int cont=0, n_sul=0, n_norte=0;
//inicializacoes feitas na main...
sem_init(&em,0,1); sem_init(&dorme,0,0); sem_init(&e,0,1);

espera() {
 sem_wait(&em); cont++; sem_post(&em);
 sem_wait(&dorme); cont--;
 if(cont>0) sem_post(&dorme);
 else sem_post(&em); }

libera() {
  sem_wait(&em);
  if(cont>0) sem_post(&dorme);
  else sem_post(&em);
}

//Pseudo-codigo dos carros do norte
sem_wait(&e);
while(n_sul > 0) {
  sem_post(&e);
  espera();
  sem_wait(&e);
}
n_norte++;
sem_post(&e);
//...aqui código para atravessar a ponte
sem_wait(&e);   n_norte--;
if(n_norte == 0)
   libera();
sem_post(&e);

//Pseudo-codigo dos carros do sul
sem_wait(&e);
while(n_norte > 0) {
   sem_post(&e);
   espera();
   sem_wait(&e);
}
n_sul++;
sem_post(&e);
//...aqui código para atravessar a ponte
sem_wait(&e);     n_sul--;
if(n_sul == 0)
   libera();
sem_post(&e);
```

Examine esse algoritmo e responda, **justificando todas as respostas** (as justificativas contam na avaliação da questão):

1. O que acontece se um carro do norte é o primeiro a chegar na ponte? Ele segue em frente ou fica retido?
2. O que acontece se em seguida outros dois carros do norte chegam na ponte, enquanto o primeiro carro ainda está atravessando? Eles seguem em frente ou ficam retidos?
3. O que acontece se antes dos carros do norte terminarem de atravessar a ponte, chega um carro do sul? Ele segue em frente ou fica retido?
4. O que acontece se outros dois carros do sul chegam na ponte, enquanto os carros do norte ainda estão atravessando a ponte? Eles seguem em frente ou ficam retidos?
5. O que acontece quando os carros do norte terminam de atravessar a ponte?
6. O código possui três varáveis globais (`cont, n_sul, n_norte`) que são acessadas pelas threads que modelam os carros. O acesso a essas variáveis está protegido para evitar condição de corrida em todos as etapas algoritmo?
7. Há possibilidade de ocorrência de *starvation* neste algoritmo?

Questão 2 (2,5 pontos) O código abaixo implementa uma solução para o problema do produtor/consumidor, considerando um buffer de tamanho ilimitado. Em uma execução com apenas um produtor (thread que executa a função `prod` e um consumidor (thread que executa a função `cons`), ocorreu do consumidor retirar um item que não existia no buffer. Identifique a causa do erro no código e proponha uma maneira de resolvê-lo (sem acrescentar novos semáforos e sem limitar a possibilidade de execução concorrente das funções `consome_item` e `produz_item`). **Justifique suas respostas.**

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

```
void *prod(void *args) {
  int item;
  while(1) {
    produz_item(&item);
    sem_wait(&s);
    insere_item(item);
    n++;
    if(n==1) sem_post(&d);
    sem_post(&s);
  }
}
```

Questão 3 (2,5 pontos) O algoritmo abaixo implementa uma solução para o problema dos *leitores e escritores* (mais de um leitor pode ler ao mesmo tempo; apenas um escritor pode escrever de cada vez e nenhum leitor pode estar lendo) com **prioridade para escrita** (sempre que há um escritor esperando, o acesso para novos leitores é bloqueado até que o escritor seja atendido). Examine esse algoritmo e responda, **justificando todas as respostas** (as justificativas contam na avaliação da questão):

1. Quais devem ser os valores iniciais dos semáforos (em_e, em_l, escr, leit) para que o algoritmo funcione corretamente?
2. O que acontece se um leitor tenta ler quando não há ninguém lendo ou escrevendo? Ele consegue ler ou fica retido?
3. O que acontece se em seguida outro leitor tenta ler antes do primeiro ter concluído a leitura? Ele consegue ler ou fica retido?
4. O que acontece se um escritor tenta escrever antes dos dois primeiros leitores terem terminado a sua leitura? Ele consegue escrever ou fica retido?
5. O que acontece se em seguida um novo leitor tenta ler antes dos dois primeiros leitores terem terminado a sua leitura? Ele consegue ler ou fica retido?
6. O que acontece quando os dois primeiros leitores terminam de ler?

7. É possível simplificar esse código (reduzir o uso dos mecanismos de sincronização)?

```
sem_t em_e, em_l, escrita, leitura; //semaforos
int e=0, l=0; //globais
```

Leitores:

```
while(1) {
 sem_wait(&leitura);
 sem_wait(&em_l); l++;
 if(l==1) sem_wait(&escrita);
 sem_post(&em_l);
 sem_post(&leitura)
 //le...
 sem_wait(&em_l); l--;
 if(l==0) sem_post(&escrita);
 sem_post(&em_l);
}
```

Escritores:

```
while(1) {
 sem_wait(&em_e); e++;
 if(e==1) sem_wait(&leitura);
 sem_post(&em_e);
 sem_wait(&escrita);
 //escreve...
 sem_post(&escrita);
 sem_wait(&em_e); e--;
 if(e==0) sem_post(&leitura);
 sem_post(&em_e);
}
```

Questão 4 (2,5 pontos) Implemente a seguinte variação do problema produtor/consumidor: *todo item inserido no buffer deve ser consumido por todos os consumidores antes de ser retirado do buffer*. Os demais requisitos do padrão permanecem iguais. Implemente as funções de **inserção** e **remoção** no *buffer* (não precisa criar as threads nem implementar a função `main`), seguindo a lógica descrita acima. Use **semáforos** para tratar todas as demandas de sincronização do problema (minimizando ao máximo os custos dessa sincronização). Considere que o número de threads produtoras é P; o número de threads consumidores é C; e o tamanho do *buffer* é N.