Exercícios com semáforos - 2015/2

Computação Concorrente (MAB-117)
Prof. Silvana Rossetto

[1]DCC/IM/UFRJ
5 de janeiro de 2016

Problema 1 Para usar um determinado recurso é necessário que duas threads (do mesmo tipo) o faça ao mesmo tempo (i.e, se uma thread quer usar o recurso, ela deve aguardar até que outra thread também queira usá-lo). O acesso ao recurso é exclusivo, então assim que as threads terminam de usá-lo, o seu acesso deve ser liberado para outros pares de threads. O código abaixo implementa uma solução para esse problema usando **semáforos**. (a) Verifique se esse código está correto, i.e., se ele atende aos requisitos do problema e não leva a aplicação à condição de **deadlock**. (b) Comente as linhas do código.

```
//globais
sem_t em, par;
int cont=0;
//inicializacoes
sem_init(&em, 0, 1); sem_init(&par,0,0);
//codigo das threads
void *T(void *args) {
  while(1) {
     //executa código que não depende do recurso...
     sem_wait(&em);
     cont++;
     if(cont!=2) {
        sem_post(&em);
        sem_wait(&par);
        sem_post(&par);
     } else {
        sem_post(&par);
     }
     //executa o código que usa o recurso...
     sem_wait(&par);
     cont--;
     if(cont==0)
        sem_post(&em);
     else
        sem_post(&par);
  }
}
```

Problema 2 No programa abaixo uma função de **barreira** (sincronização coletiva) foi implementada para forçar as threads a executarem um passo da iteração (da linha 22) e esperarem todas as demais threads executarem esse mesmo passo para então prosseguir. Entretanto, em uma execução da aplicação, ocorreu de uma thread "A" executar a linha 23 pela terceira vez antes que todas as demais threads "A" tivessem executado essa mesma linha duas vezes. (a) Descreva pelo menos uma sequência de execução que pode ter gerado essa situação. (b) Mostre como corrigir o programa.

```
1: #define NTHREADS 4
2: #define PASSOS 3
3: int threads = 0; sem_t mutex, cond;
4: void barreira(int numThreads) {
5:    sem_wait(&mutex);
6:    threads++;
7:    if (threads < numThreads) {
8:      sem_post(&mutex);
9:      sem_wait(&cond);
10:     sem_wait(&mutex);
11:     threads--;
12:     if (threads>0) sem_post(&cond);
13:     sem_post(&mutex);
14:   } else {
15:     threads--;
16:     sem_post(&cond);
17:     sem_post(&mutex);
18:   }
19: }
20: void *A (void *t) {
21:    int my_id = *(int*)t, i;
22:    for (i=0; i < PASSOS; i++) {
23:      printf("Thread %d: passo=%d\n", my_id, i);
24:      barreira(NTHREADS);
25:    }
26:    pthread_exit(NULL);
27: }
28: int main() {
29:   sem_init(&mutex, 0, 1);    sem_init(&cond, 0, 0);
30:   for(i=0; i<NTHREADS; i++) {
31:     id[i]=i;
32: pthread_create(&threads[i], NULL, A, (void *) &id[i]);
33:   }
34:   pthread_exit(NULL);
35: }
```

Problema 3 Uma aplicação dispõe de um recurso que pode ser acessado por dois tipos de threads (A e B). O código abaixo implementa a lógica para o controle de acesso ao recurso (código das threads A e B). (a) Descubra qual é essa lógica, i.e., qual é a regra que controla o acesso ao recurso?

```
//globais
int a=0, b=0; //numero de threads A e B, respectivamente
sem_init(&emA, 0, 1);
sem_init(&emB, 0, 1);
sem_init(&rec, 0, 1);

//funcao executada pelas As
void *A (void *threadid) {
  while(1) {
     sem_wait(&emA); a++;
     if(a==1) {
        sem_wait(&rec);
     }
     sem_post(&emA);
     //SC: usa o recurso
     sem_wait(&emA); a--;
     if(a==0) sem_post(&rec);
     sem_post(&emA);
  }
}

//funcao executada pelas Bs
void *B (void *threadid) {
  while(1) {
     sem_wait(&emB); b++
     if(b==1) {
        sem_wait(&rec);
     }
     sem_post(&emB);
     //SC: usa o recurso
     sem_wait(&emB); b--;
     if(b==0) sem_post(&rec);
     sem_post(&emB);
  }
}
```

Problema 4 O código 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). (a) Quais devem ser os valores iniciais dos semáforos (em_e, em_l, escr, leit) para que o código funcione corretamente? (b) Altere o código das duas threads substituindo o maior número possível de semáforos por locks. (c) Descreva se os locks usados são recursivos ou não e justifique a escolha feita. (d) É possível simplificar o código das threads?

```
sem_t em_e, em_l, escr, leit; int e=0, l=0; //globais
```

Leitores:

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

Escritores:

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