Laboratório 8
Problemas clássicos de concorrência usando locks e variáveis de condição (barreira e leitores/escritores)

Computação Concorrente (MAB-117)
Prof. Silvana Rossetto

[1]DCC/IM/UFRJ — 27 de outubro de 2016

Introdução

O objetivo deste Laboratório é implementar problemas clássicos de concorrência usando locks e variáveis de condição. Veremos hoje: **barreira** e o problema dos **leitores/escritores**.

Atividade 5

Objetivo: Implementar uma "barreira" para sincronização das threads de uma aplicação.

Descrição: Implemente um programa concorrente em C com 5 threads. Todas as threads devem executar o mesmo trecho de código mostrado abaixo. A cada passo (iteração), cada thread deve incrementar sua variável contadora, imprimir o seu ID, o valor da sua variável contadora e o passo atual, e só podem continuar depois que todas as threads completarem esse passo.

Roteiro:

1. Implemente a função `barreira()` para forçar todas as threads a esperar ao final de cada iteração.
2. Implemente o programa completo e avalie a sua corretude.
3. **Depois comente a função** `barreira()` **e observe como a aplicação se comporta**.

```
#define NTHREADS 5
#define PASSOS  5
//variaveis globais
...

void barreira(int nthreads)
{ ... }

void *A (void *arg) {
  int tid = *(int*)arg, i;
  int cont = 0, boba1, boba2;

  for (i=0; i < PASSOS; i++) {
    cont++;
    printf("Thread %d: cont=%d, passo=%d\n", tid, cont, i);
```

```
    //sincronizacao condicional
    barreira(NTHREADS);

    /* faz alguma coisa inutil pra gastar tempo... */
    boba1=100; boba2=-100; while (boba2 < boba1) boba2++;
  }
  pthread_exit(NULL);
}

int main(int argc, char *argv[])  { ... }
```

A saída do programa deverá ser como mostrado abaixo. Apenas a ordem das threads na coluna 1 pode variar de uma execução para outra.

```
Thread 0: cont=1, passo=0
Thread 1: cont=1, passo=0
Thread 2: cont=1, passo=0
Thread 3: cont=1, passo=0
Thread 4: cont=1, passo=0
Thread 4: cont=2, passo=1
Thread 0: cont=2, passo=1
Thread 2: cont=2, passo=1
Thread 1: cont=2, passo=1
Thread 3: cont=2, passo=1
Thread 3: cont=3, passo=2
Thread 4: cont=3, passo=2
Thread 0: cont=3, passo=2
Thread 2: cont=3, passo=2
Thread 1: cont=3, passo=2
Thread 1: cont=4, passo=3
Thread 3: cont=4, passo=3
Thread 4: cont=4, passo=3
Thread 0: cont=4, passo=3
Thread 2: cont=4, passo=3
Thread 2: cont=5, passo=4
Thread 1: cont=5, passo=4
Thread 4: cont=5, passo=4
Thread 0: cont=5, passo=4
Thread 3: cont=5, passo=4
```

Atividade 2

Objetivo: Projetar e implementar uma aplicação com o padrão **leitores/escritores**.

Roteiro: Implemente uma aplicação em C com uma **variável global compartilhada** que possue dois campos (struct): um contador (inteiro) e um identificador de thread (inteiro). O conteúdo dessa variável será lido por threads **leitoras** e escrito/alterado por threads **escritoras**. As *threads escritoras* alteram a variável **incrementando o valor do contador e armazenando o seu ID** (identificador da thread) no outro campo da estrutura. As *threads leitoras* lêem o valor da variável (os dois campos) e o armazena em uma variável local. Depois executam um processamento "bobo" sobre esse valor e voltam a solicitar nova leitura.

Relembrando as condições lógicas do problema leitores/escritores: *mais de um leitor pode ler ao mesmo tempo, apenas um escritor pode escrever de cada vez e se um*

escritor está escrevendo nenhum leitor pode estar lendo.

1. Crie um número N de threads leitoras ($N >= 2$) e um número M de threads escritoras ($M >= 2$).
2. Acresecente no seu código a impressão de **informações que permitam acompanhar a execução da aplicação para verificar se as condições lógicas do problema são satisfeitas**.
3. Execute a aplicação **várias vezes** e avalie os resultados obtidos.
4. Altere o número de threads leitoras e escritoras e reexecute a aplicação.
5. Mostre a aplicação executando para a professora.

Relatório:

Atividade 3 (desafio!)

Objetivo: Projetar e implementar outra solução para o problema dos **leitores/escritores** (mesma aplicação da atividade 1), agora com **prioridade para escrita**: quando um escritor deseja escrever, a entrada de novos leitores na seção crítica é impedida permanecendo assim até que a fila de escritores aguardando para executar escrita se esgote.

Roteiro:

1. Crie um número N de threads leitoras ($N >= 2$) e um número M de threads escritoras ($M >= 2$).
2. Insira no seu código a impressão de informações que permitam acompanhar a execução da aplicação para verificar se as condições lógicas do problema são satisfeitas.
3. Execute a aplicação **várias vezes** e avalie os resultados obtidos.
4. Altere o número de threads leitoras e escritoras e reexecute a aplicação.
5. Mostre a aplicação executando para a professora.