Computação Concorrente (DCC/UFRJ)

Módulo 2 - Semana 2: Problemas clássicos de concorrência usando locks e variáveis de condição

Prof. Silvana Rossetto

Dezembro 2020

Barreiras

Problema

- Vários problemas computacionais são resolvidos usando **algoritmos iterativos que sucessivamente computam aproximações melhores para uma resposta procurada** (*ex., resolução de sistemas de equações lineares usando o método de Jacobi*)
- Esses algoritmos manipulam um vetor de valores e a cada iteração executam a mesma computação sobre todos os elementos do vetor, melhorando os valores anteriores

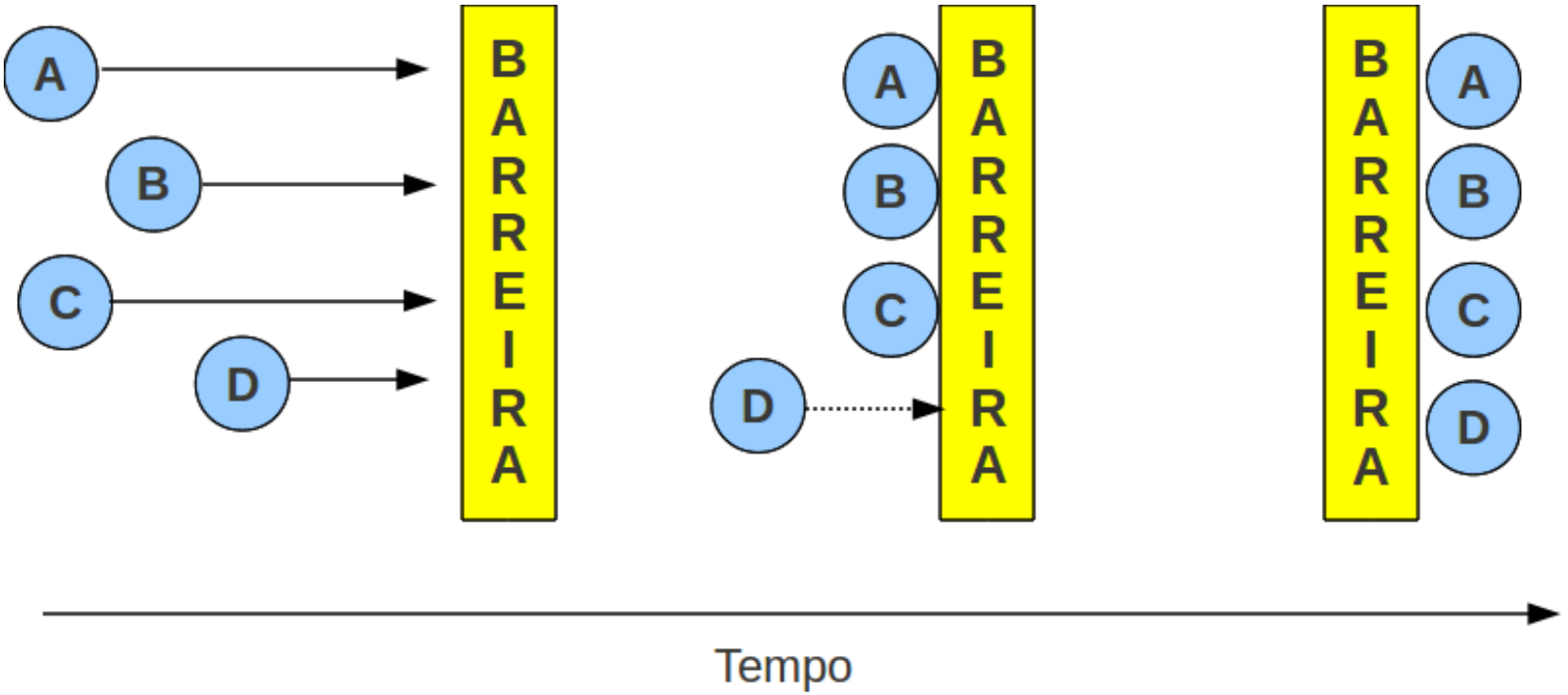
Solução concorrente

- É possível usar várias threads para computar partes disjuntas da solução de forma concorrente/paralela
- Um requisito é que cada iteração depende da anterior, então todas as threads devem completar a iteração corrente antes de qualquer uma delas iniciar a próxima iteração
- Para garantir que as threads trabalhem sempre em fase (na mesma iteração) é necessário usar um tipo de sincronização chamada **sincronização por barreira**

Sincronização por barreira

Um tipo de sincronização coletiva que suspende a execução das threads de um aplicação em um dado ponto do código e somente permite que as threads prossigam quando todas elas tiverem chegado naquele ponto

Exemplo sincronização por barreira



Implementação de sincronização por barreira

Considerando o uso de locks e variáveis de condição:

- um **contador é inicializado com o número total de threads envolvidas**
- cada thread decrementa o contador após alcançar a barreira e então se bloqueia esperando o contador chegar a zero
- quando o **contador chega a zero todas as threads são desbloqueadas**

Exemplo de implementação de barreira

```
int contador=NTHREADS;
pthread_mutex_t mutex;
pthread_cond_t cond_bar;

void barreira() {
   pthread_mutex_lock(&mutex);
   contador--;
   if (contador > 0) {
      pthread_cond_wait(&cond_bar, &mutex);
   } else {
      contador=NTHREADS;
      pthread_cond_broadcast(&cond_bar);
   }
   pthread_mutex_unlock(&mutex);
}
```

O problema dos leitores e escritores

O problema dos leitores e escritores

Uma área de dados (ex., arquivo, bloco da memória, tabela de uma banco de dados) é compartilhada entre diferentes threads:

- as **threads leitoras** apenas lêem o conteúdo da área de dados
- as **threads escritoras** apenas escrevem conteúdo na área de dados

O problema dos leitores e escritores

Condições do problema:

1. os leitores podem ler simultaneamente uma região de dados compartilhada
2. **apenas um escritor pode escrever a cada instante** em uma região de dados compartilhada
3. se um escritor está escrevendo, nenhum leitor pode ler a mesma região de dados compartilhada

Código das threads

```
void *leitor (void *arg) {
  while(1) {
    EntraLeitura();
    //le algo...
    SaiLeitura();
    //faz outra coisa...
  }
}

void *escritor (void *arg) {
  while(1) {
    EntraEscrita();
    //escreve algo...
    SaiEscrita();
    //faz outra coisa...
  }
}
```

Funções para leitura

```
int leit=0, escr=0; //globais
void EntraLeitura() {
   pthread_mutex_lock(&mutex);
   while(escr > 0) {
     pthread_cond_wait(&cond_leit, &mutex);
   }
   leit++;
   pthread_mutex_unlock(&mutex);
}

void SaiLeitura() {
   pthread_mutex_lock(&mutex);
   leit--;
   if(leit==0) pthread_cond_signal(&cond_escr);
   pthread_mutex_unlock(&mutex);
}
```

Funções para escrita

```
int leit=0, escr=0; //globais
void EntraEscrita () {
   pthread_mutex_lock(&mutex);
   while((leit>0) || (escr>0)) {
     pthread_cond_wait(&cond_escr, &mutex);
   }
   escr++;
   pthread_mutex_unlock(&mutex);
}

void SaiEscrita () {
   pthread_mutex_lock(&mutex);
   escr--;
   pthread_cond_signal(&cond_escr);
   pthread_cond_broadcast(&cond_leit);
   pthread_mutex_unlock(&mutex);
}
```

Referências bibliográficas

- *An Introduction to Parallel Programming*, Peter Pacheco, Morgan Kaufmann, 2011