Leitores e Escritores sem inanição

Aline Freire de Rezende, Gilberto Lopes Inácio Filho

2 de dezembro de 2019

Sumário

1 Introdução

O trabalho consiste em projetar uma solução para um problema clássico da computação concorrente, o problema dos leitores e escritores, garantindo a ausência de inanição.

Além disso é necessário que se gere um registro de execução do programa que possa ser validado por uma ferramenta auxiliar, de modo que se possa verificar a corretude do algoritmo.

1.1 Problema clássico de leitores e escritores

O problema dos leitores e escritores consiste em dois tipos de thread acessando um mesmo recurso. As threads leitoras se limitam a ler o recurso, enquanto as escritoras o alteram.

Na formulação clássica do problema, nós temos três condições que devem ser observadas:

- Um escritor não pode escrever enquanto outro escritor estiver escrevendo.
- Um escritor não pode escrever enquanto houverem leitores lendo.
- Os leitores podem ler simultâneamente de modo livre.

É simples encontrar uma solução para este problema, mas para este projeto deve-se levar em conta a possiblidade de que as threads sofram de inanição.

1.2 Solucionando a questão da inanição

As soluções mais simples causam inanição para a escrita, pois os leitores podem acessar o recurso livremente enquanto houver ao menos um leitor já na seção crítica.

Outras soluções mais elaboradas geram inanição para a leitura, pois de maneira quase análoga dão prioridade às threads escritoras fazendo com que os leitores só possam acessar o recurso se não houverem escritoras interessadas.

A solução que implementamos em nosso código faz uso de contadores adicionais para manter controle das threads que estiverem aguardando, tanto leitoras como escritoras. Além disso também usamos uma variável para implementar uma "alternância de turnos"entre leitoras e escritoras, fazendo com que ambas tenham a mesma prioridade.

2 Algoritmo concorrente

Cada thread leitora, ao tentar entrar na seção crítica, avalia se já há escritores na mesma, ou se há escritores esperando e o turno atual é deles. Se for um desses casos, ela se registra como uma thread leitora aguardando, e se bloqueia. Caso contrário, a thread se registra como uma leitora ativa e segue em frente.

Ao sair da seção crítica, a thread se retira do contador de leitoras ativas e define o turno como sendo das escritoras. Deste modo, se houverem escritoras esperando, as próximas leitoras não conseguirão entrar na seção crítica, e a thread leitora sinalizará para o desbloqueio de um dos escritores.

De maneira parecida, ao tentar entrar na seção crítica, uma thread escritora verifica se já há outra escritora ou leitoras acessando a mesma, ou se há leitoras aguardando no turno destas. Nestes casos a escritora se registra como esperando e se bloqueia, seguindo em frente caso contrário.

Já na saída, a thread escritora passa o turno de volta às leitoras, e sinaliza às threads que estiverem aguardando, se houver.

2.1 Entrada e saída de leitores

```
void entraLeitura() {
  pthread_mutex_lock(&mutex);
  while(escritores || (escritores_esperando && !turno_leitores)) {
    leitores_esperando++;
    pthread_cond_wait(&cond_leitor, &mutex);
    leitores_esperando--;
  }
  leitores++;
  pthread_mutex_unlock(&mutex);
}

void saiLeitura() {
  pthread_mutex_lock(&mutex);
  leitores--;
  turno_leitores = 0;
  if(escritores_esperando) pthread_cond_signal(&cond_escritor);
  pthread_mutex_unlock(&mutex);
}
```

Listing 1: Funções de entrada e saída de leitura

2.2 Entrada e saída de escritores

```
void entraEscrita() {
  pthread_mutex_lock(&mutex);
  while(leitores || escritores || (leitores_esperando && turno_leitores)) {
    escritores_esperando++;
    pthread_cond_wait(&cond_escritor, &mutex);
    escritores_esperando--;
  }
  escritores++;
  pthread_mutex_unlock(&mutex);
}

void saiEscrita() {
  pthread_mutex_lock(&mutex);
  escritores--;
  turno_leitores = 1;
  if (leitores_esperando) pthread_cond_broadcast(&cond_leitor);
  if (escritores_esperando) pthread_cond_signal(&cond_escritor);
  pthread_mutex_unlock(&mutex);
}
```

Listing 2: Funções de entrada e saída de escrita

3 Validação da execução do programa

A validação da execução do programa foi efetuada em duas partes. A primeira foi a geração de um registro de execução mantendo o controle de cada ação efetuada pelas threads, e do estado do programa quando da efetuação destas ações.

A segunda parte consistiu na implementação de uma ferramenta capaz de processar este registro e verificar a corretude do programa de acordo com os requisitos do problema.

3.1 Geração do registro de execução

O registro foi gerado de modo que cada ação executada pelas threads adiciona uma linha, num formato válido de chamada de função em Python, linguagem escolhida para a implementação da ferramenta auxiliar.

Por exemplo, se um leitor está saindo da seção crítica, podemos ter no registro algo como: `leitor_saiu(2, 4)`, que indica que o leitor com id **2** terminou seu acesso, e o valor do recurso neste momento era **4**.

Procurou-se manter no registro as informações das threads que entravam e saiam, das que estavam aguardando a condição de entrada na seção crítica, das threads ativas, do valor do recurso ao término do acesso de cada thread, das condições de bloqueio, e do turno vigente.

3.2 Ferramenta auxiliar para validação do registro

A ferramenta auxiliar foi programada em Python e correspondeu em grande parte às implementações das funções geradas no log. Através da execução dessas funções, buscamos reconstruir e avaliar o estado do programa em cada momento, e confrontá-lo com as ações realizadas pelas threads.

Dentre as verificações, as mais significativas para avaliar a nossa solução do problema foram as garantia de que:

- Escritores não acessaram a seção crítica junto com outros escritores, ou com leitores;
- Escritores não acessaram a seção crítica na existência de leitores esperando no turno destes;
- Leitores não acessaram a seção crítica ao mesmo tempo em que escritores;
- E por fim, leitores não acessaram a seção crítica com escritores esperando no turno destes.

Ao término da execução da ferramenta auxiliar, se um erro foi encontrado ele é exibido e a execução é encerrada. Em caso de execução bem sucedida, são exibidas mensagens dos pontos críticos para a ausência de inanição, em que uma thread deixou de executar haver uma de outro tipo aguardando no turno desta. Neste caso, ao final, é exibida uma mensagem de execução positiva.

Com todas estas condições cumpridas, e com o auxílio da alternância de turnos, pudemos constatar que de fato o programa executou sem inanição de threads, ou seja, nenhuma thread foi obrigada a aguardar indefinidamente.

4 Conclusão

Ao término do trabalho, obtivemos sucesso na implementação de um algoritmo que solucionasse o problema dos leitores e escritores garantindo a ausência de inanição, ainda que não tão eficiente quanto outras implementações.

Além disso também pudemos fazer a construção de uma ferramenta auxiliar eficaz com o uso específico de validar a execução do nosso programa principal, mantendo um controle do estado do nosso programa através de uma aplicação em Python.