Problema de Comunicação entre Processos Banheiro na Escola

João Lucas Azevedo Yamin R. da Cunha
(17/0013171)

2 de junho de 2019

Resumo

Desenvolvido para a disciplina de Programação Concorrente da Universidade de Brasília, este relatório tem como objetivo apresentar e solucionar um problema de sincronização entre processos, através de uma implementação em C.

1 Introdução

Apesar de permitirem maior rendimento e expandirem o leque de possibilidades para solução de problemas computacionais, as aplicações *multithread* podem causar dificuldades de complexidade maior que as causadas por aplicações que são somente sequenciais. De sincronização dos processos a problemas de compartilhamento de memória, é necessário um entendimento mais profundo sobre esta área da computação para conseguir gerir bem diversas *threads*.

Com isso em mente, foi proposta a elaboração de um problema que levasse em conta a comunicação entre processos para o compartilhamento de memória. Neste relatório, será apresentado o problema elaborado e, em seguida, a descrição do algoritmo de solução, com trechos do código implementado.

2 Formalização do Problema Proposto

2.1 Inspiração

Este problema é uma analogia entre o compartilhamento e revezamento de uso da memória com a utilização dos banheiros pelos alunos de escolas de Ensino Infantil e Fundamental.

É comum, nessas escolas, terem problemas para controlar o fluxo de alunos dentro e fora das salas de aula. Além de ser necessário pedir permissão ao/à professor(a) para ir ao banheiro, algumas escolas contam com monitores ou coordenadoras nos corredores para limitar a quantidade de alunos fora de sala.

Em algumas escolas, é utilizado uma forma de passe. Às vezes como um cartão, outras como um crachá, é uma plaquinha que o aluno leva ao banheiro como forma de limitar o número de crianças que podem sair simultaneamente para ir no banheiro e como forma de identificação para os monitores de quem foi autorizado pela professora a sair da sala.

Como esse método de passes, na minha concepção, lembra muito um mecanismo de sincronização de processos, como o semáforo, escolhi essa questão para elaborar o seguinte problema.

2.2 Descrição do Problema

Em uma determinada escola, há apenas um banheiro para cada sexo para ser dividido entre duas turmas (A e B) do 2º ano do Ensino Fundamental. Cada banheiro comporta, no máximo, três alunos. Caso estejam cheios, os alunos fazem fila para poder utilizá-lo.

Para controlar o número de alunos indo ao banheiro, o aluno precisa levar um passe ao banheiro e devolvê-lo ao voltar. Há dois passes para cada sexo em cada sala. Ou seja, por sala, podem ter, no máximo, 4 alunos indo ao banheiro.

Tanto na turma A quanto na turma B há X alunos, Y alunas e uma professora. Os valores de X e Y podem ser alterados para verificar mudança no resultado do algoritmo.

A professora, de qualquer uma das turmas, a qualquer instante, pode interromper a saída dos alunos da sala para apresentar algum conteúdo importante. A saída fica bloqueada até que a explicação acabe.

3 Descrição do Algoritmo de Solução

Serão criadas as seguintes *threads*: uma para cada professora (*profA e profB*); uma para cada aluno e aluna da turma A (*alunoA[X], alunaA[Y]*); e uma para cada aluno e aluna da turma B (*alunoB[X], alunaB[Y]*).

Quando aos mecanismos de sincronização, serão criados quatro semáforos (*passeAM, passeAF, passeBM, passeBF*) para os passes de cada sexo de cada turma, cada um com valor inicial de 2, e dois semáforos (*banheiroM, banheiroF*) para o uso dos banheiros, cada um com valor inicial de 3. Além disso, serão criados dois *locks* para o bloqueio da saída de alunos feita pela professora (*mutexA,*

mutexB).

O ciclo de execução de uma thread de aluno, genericamente, funciona da seguinte forma:

1. Espera a vez de ir no banheiro através do semáforo de passe (*sem_wait*);
2. Verifica se a professora deseja apresentar um conteúdo importante, através de uma condicional;
   (a) Se sim, espera liberação da professora através de um *lock* seguido de um *unlock*;
3. Segue para o banheiro e espera a vez de usar o banheiro ((*sem_wait*);
4. Utiliza o banheiro (através de um *sleep*);
5. Ao sair do banheiro, libera espaço para a utilização de outro aluno ((*sem_post*);
6. Volta à sala, devolvendo o passe para que outro aluno possa usar ((*sem_post*);
7. Anota no caderno e presta atenção na aula antes de precisar ir ao banheiro novamente (através de um *sleep*).

Assim, um código resumido para essa implementação genérica de aluno em C se dá como:

```
while (1) {
    sem_wait(&passe);          //Pega o passe;
    if (querExplicar) {        //Verifica se tera explicacao;
       pthread_mutex_lock(&mutex);
       pthread_mutex_unlock(&mutex);
    }
    sem_wait(&banheiro);       //Espera uma cabine livre
    sleep(2);
    sem_post(&banheiro);       //Libera a cabine
    sem_post(&passeBF);        //Libera o passe
    sleep(6);                  //Anotando
}
```

Note que a verificação da vontade da professora de passar um conteúdo importante só ocorre após o aluno pegar o passe, mas antes de ir ao banheiro. Como o aluno que está com a prioridade de ir ao banheiro está esperando, todos os outros alunos também estarão esperando, seja para ir ao banheiro, seja por causa da explicação. Assim, a verificação só precisa ser feita uma vez.

Já para a rotina da professora, podemos resumir da seguinte forma:

1. Permanece escrevendo no quadro por certo tempo;
2. Bloqueia a saída para o banheiro (através de um *lock*);
3. Anuncia o bloqueio (através de uma variável marcada para 1);
4. Explica o conteúdo importante;
5. Libera a saída para o banheiro (através de um *unlock* e a marcação da variável como 0);

Logo, em código C:

```
while (1) {
   sleep(rand() % 10);   // Escrevendo no quadro
   pthread_mutex_lock(&mutexA); // Inicia o bloqueio
   explicaA = 1;         // Anuncia o bloqueio
   sleep(rand() % 7);    // Explica
   explicaA = 0;         // Libera o bloqueio
   pthread_mutex_unlock(&mutexA);
}
```

Através desse algoritmo de solução, os alunos (*threads*) irão se revezar para utilizar os banheiros (espaços de memória em comum) entre si até o momento que alguma professora realize o bloqueio (priorização da *thread*).

No código fonte implementado para a entrega, foram adicionados outros elementos, desde contadores e impressões até *locks* para melhor controle e entendimento da execução.

3.1 Código na íntegra

O código completo pode ser encontrado no seguinte repositório: https://github.com/JLYamin/TrabalhoPC. Contém, também, instruções para execução do código.

4 Conclusão

Como visto, através da programação *multithreading* é possível trabalhar com uma gama diversa de implementações, incluindo analogias do mundo real, a exemplo do problema proposto neste relatório. Através do mesmo, foi possível abordar vários conceitos próprios da programação concorrente bem como a aplicação de duas ferramentas de sincronização entre processos, o semáforo e o *lock*.

5 Bibliografia

Referências

[1] POSIX Threads Programming. Disponível em: <https://computing.llnl.gov/tutorials/pthreads/>. Acesso em: 2 de junho de 2019.

[2] Programação Multithreading. Disponível em: <https://www.devmedia.com.br/programacao-assincrona-multithreading-em-net-com-csharp/23357>. Acesso em 2 de junho de 2019.