Trabalho de Programação Concorrente - Sincronização entre Processos

Venda de Ingressos de Cinema com Prioridades

Aluno: Rodrigo Doria Vilaça - 14/0031111

Disciplina: Programação Concorrente

Professor: Eduardo A. P. Alchieri

Universidade de Brasília

Brasília, Novembro de 2018

Introdução

Este relatório tem a finalidade de apresentar um algoritmo de venda de ingressos de cinema, que irá tratar problemas de comunicação entre processos através de uma memória compartilhada. Primeiramente, será feita uma formalização do problema da venda de ingressos, relatando os problemas relacionados à programação concorrente que serão tratados. Em seguida será feita uma descrição do algoritmo desenvolvido para a solução dos problemas apresentados na seção anterior. Por mim uma conclusão e as devidas referências. O algoritmo será realizado na linguagem de programação C, utilizando a biblioteca POSIX Pthreads para tratar os problemas de condições de corrida utilizando locks e variáveis de condição.

Formalização do Problema Proposto

O problema proposto é de um sistema de compra de ingressos de cinema. Os filmes do cinema estarão contidos em um arquivo .txt que será o “Banco de Dados” do cinema. Os filmes que estão passando no cinema, o número de sua sala e suas devidas sessões irão estar contidos neste arquivo. Neste cinema terão 3 tipos de clientes com diferentes preferências para comprar ingresso: Clientes preferenciais, com máxima prioridade para adquirir ingressos, clientes “legais” que possuem um bom senso crítico de filmes e sabem o que irão escolher, e por fim, clientes “chatos”, que são em sua maioria adolescentes que supostamente não sabem o que querem ver e só atrasam mais a fila. Cada cliente estaria usando um terminal de compra que realiza a venda do ingresso, e o número de terminais de compra disponíveis é limitado (através de um define, porém

a princípio é 5), ou seja, apenas 5 clientes podem comprar ingressos ao mesmo tempo, enquanto o resto espera na fila, que possui as prioridades citadas anteriormente. Neste sistema os clientes poderão acessar os terminais de compra "simultaneamente" e escolher seu lugar para compra, porém este processo necessita ser feito de maneira atômica para que não corra o risco de mais de um cliente comprar ingresso para um mesmo lugar, ou mais ingressos serem vendidos do que a sala suportaria. Então além de tratar a prioridade de quem terá preferência para comprar o ingresso, o programa também deverá tratar corretamente a matriz de lugares de cada sessão e o contador de ingressos comprador para cada sessão, de maneira que possa garantir acesso exclusivo à tais variáveis. Uma vez que os filmes, sessões e assentos escolhidos serão aleatórios para fins de apresentação e teste do programa, haverá um contador de tentativas para que ele não fique tentando eternamente achar um lugar válido.

Descrição do Algoritmo Desenvolvido para Solução do Problema Proposto

Primeiramente, são incluídas as bibliotecas que usaremos no código, inclusive a pthread.h, e são feitos algumas definições de variáveis que usaremos ao longo do programa para que elas possam ser alteradas facilmente depois dando maior manutenibilidade ao código:

```
#define n 100 /*número máximo de caracteres no nome do filme*/
#define max_filmes 10 /*Aqui definimos o número de filmes, caso aumente ou diminua no arquivo texto de entrada, é necessário alterar aqui*/
#define periodo 3 /*Aqui definimos o número de sessões para cada filme*/
#define max_lugares 3 /*número de lugares X lugares na sala, sempre será quadrada*/
```

```
#define sala_cheia max_lugares*max_lugares /*definimos como calcular o número máximo de lugares da sala*/
#define max_clientes_comprando 5  /* número máximo de clientes usando terminal de compra juntos */
#define max_tentativas 3 /*número máximo de tentativas de escolhas de filmes (caso esteja cheio) ou lugares (caso esteja ocupado) */
#define numero_preferenciais 3 //número de clientes preferenciais
#define numero_legais 4 // número de clientes legais
#define numero_chatos 5 // número de clientes chatos
```

Em seguida as seguintes variáveis globais são declaradas:

```
pthread_mutex_t lock; //lock
pthread_cond_t prefs; //variável de condição de preferenciais
pthread_cond_t legais; //variável de condição de legais
pthread_cond_t chatos; //variável de condição de chatos
int tem_pref = 0; //contador de preferenciais que querem comprar
int tem_legal = 0; //contador de legais que querem comprar
int clientes_comprando = 0; //contador de clientes usando terminal de compra
```

Assim como a struct na qual serão salvos as informações sobre os filmes, salas, sessões, lugares e capacidade da sessão:

```
typedef struct{
        int lugares[max_lugares][max_lugares];/*Lugares para cada sessão*/
        int capacidade = 0; //começa com 0 e a cada lugar comprado aumenta
}nodo3;

typedef struct{
        char sessao[periodo+1][n];/*sessões para cada filme*/
        nodo3 lugar[periodo+1];
```

```
}nodo2;


/*struct em que salvaremos informacoes sobre os filmes*/
typedef struct{
        int cod[max_filmes+1]; /*codigo*/
        char nome[max_filmes+1][n]; /*nome do filme*/
        int sala[max_filmes+1]; /*sala de cinema*/
        nodo2 info[max_filmes+1]; /*struct com sessoes por horario e lugares*/
}nodo;
```

Já na main, a primeira coisa a ser feita é a análise do banco de dados de filmes através do arquivo "filmes.txt", e salvamos os dados nos respectivos campos da struct. Em seguida inicializamos o lock e as variáveis de condição, e criamos as threads de acordo com o número de clientes preferenciais, legais e chatos definidos. Cada thread irá executar sua devida função. Como o objetivo da matéria é ver como o programa irá lidar com o acesso simultâneo de clientes nos terminais de compras, não faria sentido para fins de apresentação o usuário ter de digitar as informações de compra (filme, sessão, lugar, etc.), pois iria acabar com a simulação de várias compras ao mesmo tempo. Sendo assim, o programa irá aleatoriamente atribuir as informações pedidas, e o programa terá que lidar com os possíveis erros como sala lotada, ou lugar ocupado, etc. Foi implementado uma versão que checava se o dado que o usuário digitou é válido, porém como vou gerar os dados aleatórios (porém sempre válidos), esses tratamentos foram retirados.

As funções que as threads de clientes irão executar, cliente_preferencial(), cliente_legal() e cliente_chato(), irão ser executadas em loop (while(1)) para melhor simulação do problema.

Como as três funções são muito similares (realizam a compra de um ingresso no terminal de compra), porém o que muda é como a prioridade é tratada de acordo com o cliente, explicarei a função “cliente_legal()” apenas:

Primeiramente, são declaradas as variáveis que usaremos, e começa o loop infinito while(1). Então pegamos o lock e o cliente fala que vai comprar ingresso. O contador de cliente na fila é incrementado e então vemos que há terminais de compras disponíveis ou se existe algum preferencial na fila esperando pra comprar. Enquanto não houverem terminais disponíveis, ou houverem preferenciais esperando, o cliente printa quantos clientes legais estão na fila e entra no wait de sua variável de condição, que é acordado quando algum cliente sai do terminal de compra. Ao conseguir entrar no terminal de compra (sair desse while), o número de clientes legais na fila é decrementado e o número de clientes comprando é incrementado (pois agora ele já está no terminal de compra). Em seguida os números do filme e sessão são escolhidos aleatoriamente e é checado se a sessão escolhida está cheia ou não. Caso esteja, outra sessão é sorteada. Caso acabe o número máximo de tentativas de compras, o cliente desiste de comprar, decrementa o contador de clientes comprando, sai do terminal de compra, acorda os que estavam na fila e libera o lock. Caso contrário, a sessão escolhida não está cheia e o lugar é escolhido aleatoriamente e é checado se o lugar está ocupado ou não. Caso esteja, outro lugar é sorteado. Caso acabe o número máximo de tentativas de compras, o cliente desiste de comprar, sai do terminal de compra, decrementa o contador de clientes comprando, acorda os que estavam na fila e libera o lock. Caso contrário o lugar é marcado como ocupado, os lugares da sessão são impressos, e a compra é finalizada. Por fim, o cliente sai do terminal de compra e acorda os que estavam na fila e libera o lock:

```
void * cliente_legal (void *arg){
        int i = *((int *) arg);
        int cod, pp, fileira, assento, tentativas = 0;
        int y[max_lugares];
        while (1){
                pthread_mutex_lock(&lock);
                printf("\nCliente Legal [%d]: Vou comprar ingresso! \n", i);
                tem_legal ++;
                while (clientes_comprando >= max_clientes_comprando || tem_pref > 0){
                        printf("Tem %d clientes preferenciais e %d clientes legais na fila! \n", tem_pref,
tem_legal);
                        pthread_cond_wait(&legais, &lock);
                }
                tem_legal--;
                clientes_comprando++;
                cod = 1+(rand() % (max_filmes));
                pp = 1+(rand() % (periodo));
                printf("Cliente Legal [%d]: Escolhi filme %s, sessão %d \n", i, filme.nome[cod], pp);
                while (filme.info[cod].lugar[pp].capacidade == sala_cheia && tentativas <= max_tentativas){
                        printf("Cliente Legal [%d]: Sala que eu escolhi ta cheia, vou escolher outro! \n", i);
                        cod = 1+(rand() % (max_filmes));
                        pp = 1+(rand() % (periodo));
                        printf("Cliente Legal [%d]: Escolhi filme %s, sessão %d \n", i, filme.nome[cod], pp);
                        tentativas++;
                }
```

```
            if (tentativas > max_tentativas){
                printf("Cliente Legal [%d]: Tentei muito, mas ta tudo cheio, bjs tchau! \n", i);
                clientes_comprando --;
                pthread_cond_signal(&prefs);
                pthread_cond_signal(&legais);
                pthread_cond_signal(&chatos);
                pthread_mutex_unlock(&lock);
                break;
            }
            tentativas = 0;
            fileira = 1+(rand() % (max_lugares));
            assento = 1+(rand() % (max_lugares));
            printf("Cliente Legal [%d]: Escolhi assento %c%d \n", i, fileira+64, assento);
            while(filme.info[cod].lugar[pp].lugares[fileira-1][assento-1] != 0 && tentativas <=
max_tentativas){
                printf("Cliente Legal [%d]: Lugar que escolhi ta ocupado, vou escolher outro! \n", i);
                fileira = 1+(rand() % (max_lugares));
                sleep(1);
                assento = 1+(rand() % (max_lugares));
                printf("Cliente Legal [%d]: Escolhi assento %c%d \n", i, fileira+64, assento);

                tentativas++;
            }
            if (tentativas > max_tentativas){
                printf("Cliente Legal [%d]: Tentei muito, mas ta tudo ocupado, bjs tchau! \n", i);
                clientes_comprando --;
```

```
                pthread_cond_signal(&prefs);
                pthread_cond_signal(&legais);
                pthread_cond_signal(&chatos);
                pthread_mutex_unlock(&lock);
                break;
        }
        tentativas = 0;
        filme.info[cod].lugar[pp].lugares[fileira-1][assento-1] = 1;
        filme.info[cod].lugar[pp].capacidade++;
        printf("\nCompra realizada com sucesso.\n");
        printf("\n----- LUGARES FILME %s SESSÃO %d -------\n", filme.nome[cod], pp);
        for (int k = 0; k <max_lugares; k++){
                        imprime_fileira(k, pp, cod, y);
                }
        printf("\n--------------------------------------------------\n");
        pthread_mutex_unlock(&lock);
        sleep(3); //tempo finalizando compra e saindo terminal
        pthread_mutex_lock(&lock);
        printf("\nCliente Legal [%d]: Vou sair do terminal de compra! \n", i);
        clientes_comprando --;
        pthread_cond_signal(&prefs);
        pthread_cond_signal(&legais);
        pthread_cond_signal(&chatos);
        pthread_mutex_unlock(&lock);
        sleep(2);
}
```

```
 return 0;
}
```

As funções de cliente_preferencial() e cliente_chato() funcionam de maneira semelhante, porém os preferenciais apenas checam se há terminais de compras disponíveis para comprar, e os chatos além de checar os terminais de compra disponíveis e se há clientes preferenciais na fila, também checa se há clientes legais na fila, e só irá comprar caso existam terminais disponíveis, e não existam clientes preferenciais ou legais na fila.

Conclusão

Com este trabalho foi possível melhorar a compreensão sobre programação concorrente, seus conceitos e sua utilidade para resolver problemas do cotidiano. O uso de locks e variáveis de condição foi aprimorado para o uso em um programa com mais complexidades que os passados em sala de aula. Além de exercitar conceitos de programação concorrente. este trabalho também foi muito importante para exercitar a programação em C, que fazia alguns anos que não praticava. Creio que consegui atingir o objetivo proposto com este trabalho, realizando um sistema de venda de ingressos concorrente, com prioridades de clientes e lugares limitados que necessitavam de garantia de acesso exclusivo, o que foi possível graças aos locks e variáveis de condição utilizados.

Referências

[1] Prof. Eduardo Alchieri, Slide: "Condições de Corrida, Regiões Críticas e Exclusão Mútua". Programação Concorrente.

[2] Prof. Eduardo Alchieri, Slide: "Introdução". Programação Concorrente.

[3] Blaise Barney, Lawrence Livermore National Laboratory, "POSIX Threads Programming".

Fonte: "https://computing.llnl.gov/tutorials/pthreads/"

[4] Prof. Eduardo Alchieri, Slides: "Locks", "Variáveis Condição". Programação Concorrente.