Relatório - Trabalho Final

Jogo de Bola no Condomínio

Giovanni M. Guidini - 16/0122660

1º de Julho de 2019

1. Introdução

Processos concorrentes são aqueles que compartilham recursos do sistema e agem simultaneamente para a conclusão de uma tarefa. Uma característica importante desses processos é a intercalação (em inglês, interleaving) durante a execução, ou seja, os processos concorrentes afetam a execução uns dos outros, sendo necessário empregar mecanismos de sincronização e comunicação entre eles para garantir resultados esperados em todas as execuções. Este é um problema não trivial, uma vez que o programador não tem como conhecer a ordem nem a velocidade de execução relativa entre os processos concorrentes.

Ao longo da disciplina de Programação Concorrente foram apresentados diversos mecanismos para criação e sincronização desse tipo de processos, e a utilização desses mecanismos em diversas situações inerentemente concorrentes. Essa foi também a motivação deste trabalho, que consiste na formalização de um problema concorrente e da proposta de uma solução para este problema, utilizando os mecanismos aprendidos ao longo do curso de maneira apropriada.

O problema proposto é um **jogo de bola no condomínio**: Em um condomínio existe uma quadra esportiva que é utilizada pelas N crianças que moram no local e brincam de bola, sempre em dois times com o mesmo número de jogadores. No entanto somente M das N crianças possuem uma bola para brincar. As outras crianças devem esperar que alguém com uma bola esteja na quadra para poderem brincar também. Além disso, eventualmente a mãe de uma das crianças a chama para casa, momento no qual ela para de jogar. Se a criança indo para casa for dona de uma bola, ela leva sua bola consigo. Se isso fizer com que nenhuma outra criança que ficou na quadra seja dona de uma bola, o jogo termina e todas as crianças vão para casa até que consigam brincar novamente.

Na Seção 2 o problema introduzido acima é formalizado. Na Seção 3 a solução proposta para o problema é apresentada e discutida. Na Seção 4 são comentados os resultados e dificuldades do projeto.

O código completo está disponível no GitHub.

2. Formalização do Problema

2.1 Descrição do Problema

Suponha que em um condomínio existe exatamente uma quadra e também N crianças que brincam na quadra durante seu tempo livre. Mais nenhum morador utiliza a quadra, portanto as crianças podem usá-la sempre que quiserem.

Das N crianças que moram no condomínio, M dessas possuem uma bola (M < N).

As crianças brincam de bola na quadra, sempre em 2 times com o mesmo número de jogadores. Para que o jogo aconteça, pelo menos 1 das crianças que estão brincando precisa ser dona de uma bola, logo, as crianças não vão para a quadra a não ser que pelo menos 1 criança com bola já esteja lá. Ao chegar na quadra, uma criança só pode participar do jogo se sua presença mantiver os dois times balanceados (i.e. o número de crianças brincando é sempre par). Isso quer dizer que nunca pode existir um número ímpar de crianças jogando. Caso uma criança entre e não possa jogar imediatamente ela fica esperando até que outra criança chegue, para que os times continuem balanceados, ou até que uma das crianças jogando tenha que sair, de forma que quem estava esperando toma o seu lugar.

As crianças param de brincar e vão imediatamente para casa quando sua mãe a chama ou quando não há mais crianças com bola na quadra. Quando uma criança sai, seu lugar no time é tomado por alguma criança que esteja esperando para jogar. Caso ninguém esteja esperando para jogar, então a criança saindo escolhe alguém para parar de jogar, mantendo os dois times balanceados. Se a criança saindo for a única na quadra que possui uma bola, ela leva sua bola com ela e o jogo termina.

2.2 Sobre a Implementação

Para a implementação, cada criança é representada por uma *thread*. Foram criadas 24 *threads* (crianças). 3 das "crianças" possuem uma bola. As crianças executam um loop infinito: entrar na quadra, brincar, sair da quadra, aguardar poder ir para a quadra novamente.

As mães das crianças estão representadas na *thread main*. A cada segundo uma criança aleatória pode ser chamada para sair da quadra por sua "mãe".

A *main* também é responsável por mostrar as informações sobre o estado do jogo na tela (ver Figura 1). As informações são: Número de crianças brincando, Número de crianças dentro da quadra, Número de crianças que possuem uma bola dentro da quadra, qual criança está esperando para jogar (caso haja), quais crianças estão jogando, e quais foram as últimas crianças que entraram/saíram da quadra.

Figura 1: Captura de tela da saída do programa para solução do problema apresentado.

3. Solução do Problema

3.1 Variáveis de sincronização

Para garantir a sincronização das *threads* e os resultados esperados foram utilizados locks, variáveis de condição e semáforos (Veja Figura 2). O lock `court` controla a entrada e saída de crianças da quadra. O lock `states` é um lock recursivo que controla o estado atual de uma criança. O lock `events` controla a inserção de novos eventos. A variável de condição `lets_play` avisa crianças da chegada de alguém com bola na quadra. Finalmente, o vetor de semáforos `child_pb` permite acordar uma criança que está brincando.

```
pthread_mutex_t court = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t events = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t states; // Lock recursivo
pthread_mutexattr_t states_attr;
pthread_cond_t lets_play = PTHREAD_COND_INITIALIZER;

sem_t child_pb[KIDS];   // Semaforo da criança
int child_state[KIDS];  // Situação da criança na quadra
bool played[KIDS];      // Se a criança ja brincou ou nao
```

Figura 2: Variáveis de sincronização utilizadas.

3.2 Regiões Críticas

A primeira região crítica é a entrada de crianças na quadra. Só uma criança por vez pode entrar, para manter o contador de crianças que estão na quadra e o número de crianças com bola na quadra consistente. Também é necessário parar as crianças que não são donas de bola caso ninguém com bola esteja dentro da quadra no momento que essa criança tente entrar. O Código 1 mostra a implementação desta parte do código. Note que são utilizadas as variáveis `court` e `lets_play`. Além disso a função `set_my_state` também é executada em exclusão mútua internamente.

```
// Criança quer brincar
pthread_mutex_lock(&court);
if(info->has_ball){
  // Criança com bola ja vai brincar
  // E chama os amiguinhos para brincar tambem
  ball_owners_in_court++;
  pthread_cond_broadcast(&lets_play);
} else {
  // Nenhuma criança esta brincando
  // Espera alguem com bola chamar para brincar
  while(ball_owners_in_court == 0){
     pthread_cond_wait(&lets_play, &court);
  }
}
// Consegue entrar na quadra
// Insere novo evento
kids_in_court++;
played[info->id] = false;
set_my_state(info->id, WANNA_PLAY);
pthread_mutex_unlock(&court);
```

Código 1: Controle da entrada de crianças na quadra. As chamadas de função à variáveis de sincronização estão destacadas em vermelho.

Outra região crítica é quando a criança vai brincar, pois precisa ser respeitada a restrição de só haver um número par de crianças jogando. Além disso, a *thread* da criança precisa ficar bloqueada enquanto ela está jogando, até que algo a acorde, sendo que ela pode ser acordada por 4 motivos:

- Quer brincar mas precisa esperar mais alguém para ter um número par de crianças jogando;
- Sua mãe a chama para casa;
- Alguém sai e tira ela do jogo;
- A última pessoa com bola sai e todos tem que sair também.

Para isso cada criança fica aguardando um post no seu semáforo próprio. Quando esse post vem ela toma uma ação apropriada de acordo com seu estado

atual. Os estados possíveis são `WANNA_PLAY`, `LETS_PLAY` e `COME_HOME` que se referem à criança estar na quadra, mas não brincando, estar na quadra brincando e não estar na quadra, respectivamente. Como esta implementação é mais extensa o leitor interessado pode encontrá-la nas linhas 127 à 145 código. Essa região crítica utiliza as variáveis de sincronização `child_pb e states.`

Uma outra região crítica é a saída de crianças da quadra. Como a saída de uma criança pode afetar o estado de outras crianças e afeta o número de pessoas na quadra, esta região utiliza todas as variáveis de sincronização (com exceção de `events).` As ações que cada criança deve tomar ao sair da quadra estão relacionadas ao fato dela possuir ou não uma bola, e ela ter conseguido jogar ou não. Este último ponto é muito importante, pois é possível que uma criança seja chamada para casa enquanto está esperando para jogar, e não jogando de fato. Quando a criança que sai estava jogando, precisa manter os times balanceados após sua saída. A implementação referente a esta parte se encontra nas linhas 148 à 209 no código.

4. Conclusão

Neste trabalho, por meio da formalização e proposta de solução de um problema concorrente, foi possível pôr em prática os diversos conceitos vistos em sala de aula no que tange à programação concorrente. Em particular a sincronização entre *threads*, que é um processo relativamente complicado pois possui diversos detalhes a serem observados, foi bem feito, garantindo os resultados esperados sempre que o programa é executado.

A variedade de estruturas de sincronização utilizada é um bom indicador de um problema com complexidade relativamente alta, e traz boas oportunidades de aprendizagem. Em particular cabe aqui a observação que cada estrutura de sincronização diferente foi utilizada de maneira diferente e apropriada às suas características, o que era um dos objetivos do trabalho.

Houve certa dificuldade na sincronização correta das *threads*, principalmente no processo de sair da quadra e nas mudanças de estado que cada criança possui. A complexidade de depuração em um programa *multithread* complicou consideravelmente o tempo de arrumação do código. Variáveis extras tiveram que ser empregadas, como um vetor para saber se a criança que está saindo brincou ou não. Além disso o uso de semáforos para acordar as crianças foi escolhido devido à sua facilidade de uso, porém trouxe suas consequências. Por exemplo, um dos erros que demorou mais a ser percebido era o fato de que o semáforo que acordava uma criança podia ter permissões a mais, que causavam essa criança “ir brincar com ela mesma”, e achar que isso mantinha os times balanceados. O conserto dessa situação foi simples, apenas impedir uma criança de se remover do estado de querer brincar, mas perceber o erro foi um pouco demorado.

No geral acredito que o trabalho seja relevante dentro de sua proposta, o problema escolhido possui um certo grau de complexidade que permite um

bom aprendizado e prática do uso das estruturas aprendidas em sala. Pessoalmente estou bastante satisfeito com o algoritmo desenvolvido e os resultados do trabalho.

5. Referências

- Material de aula, Prof. Eduardo Alchieri, Univerisdade de Brasília. https://cic.unb.br/~alchieri/disciplinas/graduacao/pc/pc.html
- Manual POSIX Threads. Blaise Barney, Lawrence Livermore National Laboratory. https://computing.llnl.gov/tutorials/pthreads/
- How to Declare a Recursive Mutex with POSIX threads? Pithicos. Stack Overflow. https://stackoverflow.com/questions/7037481/c-how-do-you-declare-a-recursive-mutex-with-posix-threads