CIC Royale

Programação Concorrente 1/2019
Aluno: Frederico Pinheiro Dib
Matrícula: 15/0125925
Professor: Alchieri

1) Introdução

Esse trabalho visa a utilização de conceitos aprendidos na matéria de Programação concorrente Para resolver problemas computacionais de maneira mais simples e mais prática.

2) Descrição do Problema

O problema se baseia em uma mistura de campeonato com battle royale. Em uma ilha existe um número N de jogadores participando. Cada jogador começa com o um nome aleatório, 50 de vida e a arma palmada, que possui 5 de dano.

Durante esse período na ilha, cada jogador fica procurando por armas e comida, cada uma tendo um número máximo em jogo. Quando um jogador encontra uma arma, se essa arma for melhor que sua arma atual, ele troca pela norma arma encontrada, Se for pior, o jogador simplesmente ignora a arma. Ao encontrar alguma comida, Se a comida for boa, a vida total daquele jogador aumente, se for ruim, aquele jogador perde vida.

A cada X tempo, um alarme é tocado, quando isso acontece, todos os jogadores devem parar o que estão fazendo e ir ao centro da ilha. No centro da ilha, dois jogadores são escolhidos aleatoriamente para batalharem até a morte. Cada vez um ataca, até algum dos dois morrer. Após a batalha todos os jogadores vivos voltam para ilha, e continuam procurando por armas e comida até o alarme tocar novamente.

O ciclo se repete até só restar um vencedor. Quando só sobrar um jogador, ele será o vencedor.

3) Concorrência no problema

3.1) Tipos de Threads:

Para solucionar de maneira mais simples cada jogador realizar suas ações em paralelo, utilizei programação concorrente.

Existe dois tipos de threads no sistema:

1. Thread Jogador
2. Thread Batalha.

Para cada um dos N jogadores existe uma thread do tipo jogador para ele. Nessa thread, existe um loop infinito, dentro desse loop o jogador procura em tempo em tempo por armas e comida, a thread so sai do loop e chega ao fim quando o jogador morrer.

A thread da batalha possui um funcionamento parecido, ela possui um while infinito, e só chega ao fim quando restar 1 ou 0 jogadores em campo. Porém diferente da Threads do jogadores, que possuem várias, a thread da batalhe possui apenas uma rodando.

3.2) Variáveis de controle:

Para resolver o problema, depois de pensar cheguei a conclusão que apenas locks seriam o bastante.

Dois locks são utilizados no sistema:

1. `pthread_mutex_t p_arma`
2. `pthread_mutex_t p_comida`

Na função dos jogadores os locks foram utilizados da seguinte forma:

1. Loop
2. lock(`p_arma`)
3. verifica se está vivo
4. procura arma
5. unlock(`p_arma`)
6. sleep
7. lock(`p_comida`)
8. verifica se está vivo
9. procura comida
10. verifica se está vivo
11. unlock(`p_comida`)
12. sleep

Utilizando os locks dessa forma, eu impeço que dois jogadores acessem o vetor de arma ou o de comida ao mesmo tempo, impedindo que dois jogadores peguem uma arma ou uma comida que só tenha uma unidade. Porém um jogador consegue acessar o vetor de armas ao mesmo tempo que outro jogador acesse o vetor de comidas. Em nenhum caso algum sleep fica dentro do lock, para não travar o progresso de outro jogador.

Sempre após a entrada de cada lock, seja o lock de comida ou o lock de arma a primeira ação é a thread verificar se o jogador de indice equivalente ao seu id ainda está vivo, caso não esteja mais vivo, é dado um unlock, logo em seguida um pthread_exit.

Essa verificação é importante, pois a qualquer momento antes da entrada do lock pode ter havido uma batalha e aquele jogador ter morrido nela. Por essa mesma linha de raciocínio após procurar comida é verificado se o jogador ainda está vivo, pois caso ele coma alguma comida ruim, ele pode morrer, mesmo fora de uma batalha.

Na função de procurar arma, é gerado um número aleatório de 0 a 5. Se o número sorteado for diferente de 0, nada acontece, caso seja 0, o jogador terá encontrado uma nova arma. Após isso é sorteado um novo valor entre 0 e NUMERO_DE_ARMAS - 1, e esse número é o índice da arma. Caso a arma sorteada tenha quantidade = 0, nada acontece pois a arma já está esgotada, caso seja maior que 0, é verificado se a arma do jogador é melhor que a nova, se a arma encontrada for maior, a arma do jogador é substituída, e o contador de quantidade daquela arma é decrementada.

Na função de procurar comida, é gerado um número aleatório de 0 a 5. Se o número sorteado for diferente de 0, nada acontece, caso seja 0, o jogador terá encontrado uma nova comida. Após isso é sorteado um novo valor entre 0 e NUMERO_DE_COMIDAS - 1, e esse número é o índice da comida. Caso a comida sorteada tenha quantidade = 0, nada acontece pois a comida já está esgotada, caso seja maior que 0, o jogador consome aquela comida, o valor dela é adicionado à vida dele, seja esse valor negativo ou positivo.

Na função da batalha, os locks foram utilizados da seguinte forma:

1. Loop
2. sleep
3. lock(`p_arma`)
4. lock(`p_comida`)
5. verifica vencedor
6. escolhe jogadores
7. acontece batalha
8. verifica vencedor
9. unlock(`p_comida`)
10. unlock(`p_arma`)

Após o tempo de sleep a thread da batalha toma posse dos dois locks, para travar todas as outras threads, e impedir que alguma delas procure por arma ou comida.

Posteriormente ao pegar posse dos locks, é verificado se houve algum vencedor, pois é possível que algum jogador tenha morrido por comida entre as batalhas. Caso ainda não tenha vencedor, é sorteado dois jogadores para batalhar, posteriormente, após a batalha é verificado novamente se possui algum vencedor. Caso não haja, a thread de batalha retira a posse dos locks e as threads voltam a executar normalmente. Caso existe algum vencedor, o programa encerra.

4) Conclusão

Esse trabalho foi importante para colocar em prática os conceitos de programação concorrente. O trabalho serviu para pensar em um problema que se encaixaria bem no conceito de programação concorrente, além dos problemas tradicionais que vimos em sala. E também trabalhamos a implementação desse algoritmo na prática.