Universidade Federal de Santa Catarina

Departamento de Informática e Estatística

Ciência da Computação

INE5410 – Programação Concorrente

Professore s: Márcio Bastos Castro e Frank Augusto Siqueira

Equipe: Anthony Bernardo Kamers e Henrique Tridapalli

Relatório - Trabalho Prático I

Simulador de Laser Tag

Florianópolis, 10 de Abril de 2021

Perguntas

1 - Explique que estrutura de dados foi utilizada para armazenar os equipamentos (arma laser, colete e capacete):

Em prateleira.c, fizemos uma variável local para armazenar um struct de arranjos, para deixar o código organizado. Nesse *struct*, contém um arranjo de inteiros (pois deve armazenar somente o ID do equipamento) para cada item que o jogador pode escolher, ou seja, um arranjo para colete, um para capacete e outro para arma.

O arranjo feito foi uma lista, portanto implementamos os métodos já pré-definidos como *arranjo_colocar* como *push_back* e *arranjo_remover* como remove. Além disso, foram criados mais alguns métodos auxiliares como *arranjo_vazio* e *arranjo_cheio*, para facilitar o uso do arranjo. Foi criado também uma função *arranjo_pop*, para retirar um elemento em um determinado índice.

2 - Que estratégia foi utilizada para controlar o acesso de jogadores às equipes A e B?

Foi adicionado um semáforo no arranjo. Então, cada vez que é criado um arranjo, deve ser determinado o valor inicial de seu semáforo. No caso das equipes, é passado o valor de njogadores (passado à função equipe_setup em equipe.c, que é o mesmo que params->jogadores_por_equipe). Dessa maneira, ao iniciar o jogo, cada equipe já começa com a quantidade de espaços no semáforo necessários para que os jogadores possam entrar nela.

Além disso, quando jogador chama a função jogador_escolhe_equipe e o gerente libera para que possa escolher a equipe, pegamos um valor aleatório (usando a função fornecida) entre 0 e 2, ou seja equipe A ou equipe B. É feito então um sem_trywait no semáforo da equipe determinada pela função aleatório. Se não conseguir "pegar" o semáforo, quer dizer que a equipe já está cheia. Dessa maneira, faz um sem_wait na equipe oposta conseguindo, então, entrar na equipe.

3 - Que estratégia foi utilizada para controlar o acesso aos equipamentos (arma laser, colete e capacete)?

Para controlar o acesso aos equipamentos, foi determinado um identificador para cada tipo de equipamento "*_id*", e foi utilizado um semáforo para garantir o limite de acesso aos equipamentos disponíveis na prateleira `semaforo_equipamentos_disponiveis`, que é inicializado com "*2 * params->jogadores_por_equipe*", garantindo que no início da partida, tenha a quantidade suficiente para cada jogador.

É feito então um *sem_wait* em *jogador_aloca_equipamentos* (indicando que foi pego cada equipamento necessário da prateleira), e feito um *sem_post* em *prateleira_libera_equipamentos* (indicando que retornou à prateleira cada equipamento alocado previamente.

4 - Como você garante que não ocorre *deadlock* na alocação dos equipamentos?

Foi definido um semáforo no struct de *partida_t* denominado *semaforo_equipamentos_disponíveis*, que é iniciado com a quantidade de 2 * *params->jogadores_por_equipe*, ou seja, quando é iniciada a partida, é garantido que cada jogador consiga alocar seu equipamento. Então quando jogador chama a função *jogador_aloca_equipamento*, é chamado um *sem_wait* no semáforo estabelecido. Dessa maneira, se tiver equipamentos na prateleira, chama a função *prateleira_pega_equipamentos* e prossegue para a próxima etapa, do contrário vai esperar ter equipamentos disponíveis na prateleira.

Para que não ocorra deadloc, é feito um *sem_post* no semáforo destacado, na função *prateleira_pega_equipamentos*, após colocar os equipamentos de um jogador da partida anterior novamente na prateleira. Dessa maneira, libera para que o jogador que estava esperando liberar um item na prateleira, consiga continuar seu código.

5 - Como você garante que uma partida somente se inicia quando as duas equipes estiverem completas e todos os jogadores possuírem seus equipamentos?

No struct de *partida_t* foi colocado uma variável inteira auxiliar denominada *jogadores_esperando*, inicializada com 0. Dessa maneira, cada vez que o jogador chega à função *jogador_espera_partida_começar*, é incrementada essa variável e espera com um sem_wait em um semáforo que foi colocado em *partida_t* também, chamado *semaforo_comecar_partida*.

O último jogador que incrementar essa variável, dispara um sem_post no semáforo que também foi colocado no struct da *partida_t*, denominado *semaforo_gerente_jogadores_esperando*, fazendo com que a thread do gerente avance em seu código e dispare um laço de 2 * *params->jogadores_por_equipe* em semaforo_comecar_partida. Dessa maneira, todos os jogadores que estavam esperando pelo disparo deste semáforo vão entrar no jogo "oficialmente".

6 - As estratégias adotadas para controlar o acesso às equipes e aos equipamentos permitem um alto grau de concorrência/paralelismo na execução da simulação? Justifique sua resposta.

O uso de semáforos para o controle de acesso às equipes e aos equipamentos pelas *threads* principais, permitem que estas sempre estejam submetidas a um contador de “permissões” referentes aos semáforos, permitindo o acesso controlado e concorrente aos recursos compartilhados.

Como dito anteriormente, para haver o controle de acessos, é utilizado o contador dos semáforos. Com isso, visto a quantidade de semáforos desenvolvida pela equipe, foi notada uma restringência de concorrência,

Tal restrição de concorrência pode ser evidenciada pela necessidade dos jogadores estarem sempre esperando um sinal (através dos waits de semáforos específicos para cada parte) do gerente. Dessa maneira, faz com que as *threads* de jogadores precisem sempre depender do thread gerente, fazendo com que fique mais lento a execução.