Relatório do Trabalho Final

Problemas de concorrência na 4ª Grande Guerra Ninja

Kleber Rodrigues da Costa Júnior

20/0053680

21 de janeiro de 2023

Universidade de Brasília – Instituto de Ciências Exatas

Departamento de Ciência da Computação – CIC0202 – Programação Concorrente

2022.2 – Turma A – Professor Eduardo Adilio Pelinson Alchieri

Prédio CIC/EST – Campus Universitário Darcy Ribeiro

Asa Norte 70919-970 Brasília, DF

kleberrjr7@gmail.com

Resumo: Este documento consiste no relatório da implementação de um problema proposto para o trabalho final da disciplina Programação Concorrente. O problema é tematizado no universo de Naruto e envolve locks, variáveis de condição e semáforos.

Palavras-chave: Programação concorrente, condições de corrida, variáveis condicionais, locks, semáforos, ninjas, Naruto, algoritmo.

1. Introdução e explicação do problema

O problema proposto é ambientado no universo do anime Naruto.

A 4ª Grande Guerra Ninja chegou e Naruto, juntamente com seus aliados, precisa combater o grande vilão da história. Embora Naruto seja muito poderoso, alguns de seus amigos precisam de ajuda para conseguirem se tornar fortes o suficiente durante a batalha e conseguirem derrotar o inimigo.

Figura 1. Naruto envolto de poder.

Figura 2. O vilão mascarado.

Graças às habilidades de Naruto, ele consegue transferir seu poder para qualquer aliado que precise. Sabendo disso, seu amigo Rock Lee constantemente pede uma porção de poder à Naruto. Por estar ocupado com a batalha, Naruto não pode pessoalmente ajudar Lee, mas pode utilizar seu *jutsu clone das sombras* para fazer clones de si mesmo e os enviarem ao auxílio de seu amigo.

Figura 3. Jutsu clones das sombras em ação.

Figura 4. Rock Lee.

Toda vez que um clone da parte de seu poder a Lee, ele perde força vital e precisa recuperá-la com o auxílio de uma ninja médica. Entretanto, só há uma ninja médica no campo de batalha e ela precisa dar atenção especial aos demais ninjas feridos, só podendo revitalizar os clones quando nenhum ninja normal estiver precisando de ajuda. Seu nome é Sakura.

Figura 5. Clone dando poder ao Rock Lee.

Figura 6. Sakura curando Naruto.

Sakura utiliza sua reserva de *chakra* (poder dos ninjas) para curar feridos e revitalizar clones, porém só consegue atender 1 pessoa de cada vez. Além disso, toda vez que seu chakra se esgota ou fica muito baixo, ela deve parar com os atendimentos e reunir mais chakra.

Como organizar a Aliança Ninja para que Lee consiga lutar com o auxílio dos clones e nenhum ninja fique ferido por muito tempo?

2. Solução e implementação

Antes de nos aprofundarmos na solução, precisamos criar as threads que vão representar a abstração dos personagens. Como temos um número de clones e ninjas normais "indefinido" (o número está definido nas constantes `NUM_CLONES` e `NUM_REG_NINJAS`, respectivamente), criaremos um vetor de threads para cada grupo. Já nossa ninja médica Sakura e o nosso amigo Rock Lee só precisam de uma thread para representa-los.

```
pthread_t sakura;
pthread_t rock_lee;
pthread_t regular_ninjas[NUM_REG_NINJAS];
pthread_t naruto_clones[NUM_CLONES];
```

Deste ponto em diante, considere todos os mecanismos utilizados (locks, variáveis de condição e semáforos) devidamente declarados e inicializados, pois não iremos mostrar esses trechos a fim de economizar páginas.

Abordando a primeira parte do problema: precisamos garantir que os ninjas normais e os clones possam ser curados pela mesma ninja médica sem problemas de concorrência. O recurso compartilhado em questão é o chakra de Sakura, que é drenado para curar os feridos e revitalizar os clones, nesse caso a zona de cura é uma região crítica e deve ser protegida com um lock para garantirmos a exclusão mútua.

```c
void *f_sakura(void *arg) {
    while(1) {
        pthread_mutex_lock(&healing_zone);

        pthread_mutex_unlock(&healing_zone);
    }

    pthread_exit(0);
}
```

Sakura só pode fazer 2 coisas: reunir/recarregar chakra e curar os feridos, logo precisamos estabelecer as condições que levarão ela a executar uma tarefa ou outra. Para curar um ninja Sakura gasta 25% de seu chakra e para revitalizar um clone ela gasta 15% de seu chakra. Como ninjas são prioridade, enquanto Sakura tiver um nível de chakra acima de 25%, ela continuará com seus cuidados médicos dormindo na condição e liberando o lock da zona de cura, caso contrário ela deverá parar os atendimentos para recarregar seu nível de chakra e após isso enviar um sinal para todos os ninjas e clones esperando seu retorno. É importante que utilizemos um `pthread_cond_broadcast`, pois temos mais de uma thread esperando e todas elas precisam competir pelo recurso.

```c
void *f_sakura(void *arg) {
    while(1) {
        pthread_mutex_lock(&healing_zone);
            // Enquanto houver houver chakra suficiente, a Sakura pode descansar
            while(sakura_chakra > CHAKRA_TO_HEAL_NINJA) {
                printf("Sakura não precisa reunir mais chakra por hora... -- Nivel de chakra atual: %d\n", sakura_chakra);
                pthread_cond_wait(&sakura_cond, &healing_zone);
            }

            // Se o chakra for insuficiente, Sakura deve reunir mais
            printf("Sakura esta reunindo chakra... -- Nivel de chakra da Sakura: %d\n", sakura_chakra);
            sakura_chakra = MAX_CHAKRA_LEVEL;
            sleep(4);
            printf("Sakura esta com 100 por cento de seu chakra!\n");

            // Acorda os ninjas e os clones para que eles possam entrar na zona de cura
            pthread_cond_broadcast(&regular_ninja_cond);
            pthread_cond_broadcast(&naruto_clone_cond);
        pthread_mutex_unlock(&healing_zone);
    }

    pthread_exit(0);
}
```

Os ninjas normais “utilizam” o chakra de Sakura para se curarem, e portanto devem acessar esse recurso dentro do mesmo lock utilizado anteriormente. Por, terem prioridade, sempre que conseguirem entrar na região crítica incrementaremos uma variável para indicar que um ninja ferido está precisando de atendimento. Uma vez dentro da zona de cura, o ninja só pode fazer 2 coisas: ser curado ou esperar. Ele só vai esperar caso Sakura não tenha chakra suficiente para lhe curar, nesse momento ele deve avisar (acordar) a ninja médica para que ela vá reunir mais chakra (`pthread_cond_signal`, pois só há 1 Sakura). Caso contrário, ele irá decrementar o recurso compartilhado retirando a

quantidade de chakra que precisa para ser curado, além da variável indicadora de ninjas feridos, e retornará para o campo de batalha liberando o lock da zona de cura.

```c
void *f_regular_ninja(void *arg) {
    int id = *((int *) arg);

    printf("Ninja %d entrou no campo de batalha!\n", id);

    while(1) {
        sleep(5+rand()%5);

        // Ninja se machuca e pega o lock da zona de cura
        pthread_mutex_lock(&healing_zone);
            // Incrementa a variável para indicar que um ninja normal precisa de cura
            regular_ninjas_injured++;
            printf("O ninja %d se machucou e precisa ser curado!\n", id);

            // Enquanto a sakura não tiver chakra o suficiente, o ninja aguarda
            while(sakura_chakra < CHAKRA_TO_HEAL_NINJA) {
                printf("Sakura nao tem chakra suficiente para curar o ninja %d -- Nivel de chakra da Sakura: %d\n", id, sakura_chakra);
                pthread_cond_signal(&sakura_cond);
                pthread_cond_wait(&regular_ninja_cond, &healing_zone);
            }

            regular_ninjas_injured--;
            sakura_chakra = sakura_chakra - CHAKRA_TO_HEAL_NINJA;
            printf("Ninja %d foi curado e vai retornar para o campo de batalha! -- Nivel de chakra da Sakura: %d\n", id, sakura_chakra);
        pthread_mutex_unlock(&healing_zone);
    }

    pthread_exit(0);
}
```

Assim como os ninjas e a Sakura, os clones precisam acessar a região crítica através do lock da zona de cura, uma vez dentro eles podem as mesmas 2 coisas mencionadas no parágrafo anterior para os ninjas: ser revitalizado e esperar. Entretanto, os clones não possuem prioridade no atendimento, portanto as condições de parada devem ser:

a. Sakura está com o nível de chakra abaixo de 15% (necessário para revitalizar o clone) ou
b. Há ninjas feridos esperando atendimento

Caso qualquer uma dessas condições seja verdadeira, o clone irá esperar em `pthread_cond_wait(&naruto_clone_cond, &healing_zone)`, liberando o lock da zona de cura e acordando a Sakura. Em determinado momento, é possível que o clone acorde Sakura no momento em que ela já está acordada, mas isso não causa nenhum tipo de deadlock devido à condição de parada de Sakura. Sakura só vai “dormir” quando seu nível de chakra for inferior à quantidade necessária para curar um ninja, logo se houver ninjas esperando por atendimento, então certamente Sakura estará juntando mais chakra, se houver ninjas ativos, então eventualmente um deles irá acordar Sakura. Os clones não causam deadlock.

Se nenhuma condição for satisfeita, o clone é revitalizado, ganhando 15 pontos de poder vital e liberando o lock da zona de cura.

```c
void *f_naruto_clone(void *arg) {
    int id = *((int *) arg);
    int vital_power = 0;
    int value = 0;

    printf("Naruto fez o clone de numero %d!\n", id);

    while(1) {
        sleep(rand()%3);

        pthread_mutex_lock(&healing_zone);
            printf("O clone %d precisa de mais forca vital para ajudar o Lee! -- Forca vital atual: %d\n", id, vital_power);

            // O clone só será atendido quando não houver ninjas normais esperando pelo atendimento e o chakra da sakura for o suficiente
            while(sakura_chakra < CHAKRA_TO_VITALIZE_CLONE || regular_ninjas_injured > 0) {
                if (sakura_chakra < 15) {
                    printf("Sakura nao tem chakra suficiente para dar força vital ao clone %d -- Nivel de chakra da Sakura: %d\n", id, sakura_chakra);
                }

                if (regular_ninjas_injured > 0) {
                    printf("O clone %d nao pode ser curado pois ha um ninja machucado -- Nivel de chakra da Sakura: %d\n", id, sakura_chakra);
                }

                pthread_cond_signal(&sakura_cond);
                pthread_cond_wait(&naruto_clone_cond, &healing_zone);
            }

            vital_power = vital_power + 15;
            sakura_chakra = sakura_chakra - CHAKRA_TO_VITALIZE_CLONE;
            printf("O clone %d recebeu um pouco de poder vital! -- Forca vital atual: %d -- Nivel de chakra da Sakura: %d\n", id, vital_power, sakura_chakra);
        pthread_mutex_unlock(&healing_zone);
```

Para doar uma porção de poder a Rock Lee, cada clone precisa de 40 pontos de poder vital, logo testamos em loop infinito quantos pontos o clone possui. Nesse momento utilizaremos um semáforo que serve como contador para as porções de poder dadas à Lee. Toda vez que um clone junta 40 pontos ou mais, ele incrementa as permissões do semáforo com um `sem_post(&power_portions)`.

```c
// Enquanto o clone tiver poder vital para ajudar o Lee, ele o fará
while (vital_power >= POWER_TO_HELP_LEE) {
    sem_post(&power_portions);
    sem_getvalue(&power_portions, &value);
    vital_power = vital_power - POWER_TO_HELP_LEE;
    printf("O clone %d deu uma porcao de poder ao Rock Lee, agora ele tem %d! -- Forca vital atual: %d\n", id, value, vital_power);
}
```

A função de Rock Lee é simples: ele irá utilizar uma porção de poder sempre que estiver disponível, isto é, sempre que o semáforo tiver permissões. Estas permissões serão incrementadas sempre que um clone doar uma porção de poder, como mencionado anteriormente. Sobre o semáforo, ele deve ser iniciado com 0 permissões, pois inicialmente Lee não tem a capacidade de lutar e deve ficar bloqueado.

```c
void *f_rock_lee(void *arg) {
    int value = 0;

    printf("Rock Lee entrou no campo de batalha!\n");

    while(1) {
        // Sempre que houver porções de poder para o Lee, ele pega uma e vai lutar...
        sem_wait(&power_portions);
        printf("Rock Lee utilizou uma porcao do poder do Naruto! -- Quantidade de porcoes restantes: %d\n", value);
        printf("Rock Lee esta lutando contra os inimigos...\n");
        sleep(2+rand()%5);
    }

    pthread_exit(0);
}
```

3. Análise de resultados

Durante todo o código foram colocados diversos prints para entendermos o que está acontecendo, isso nos permitirá analisar a execução do programa de forma detalhada e entender a concorrência das threads.

Observe a seguinte imagem com os resultados:

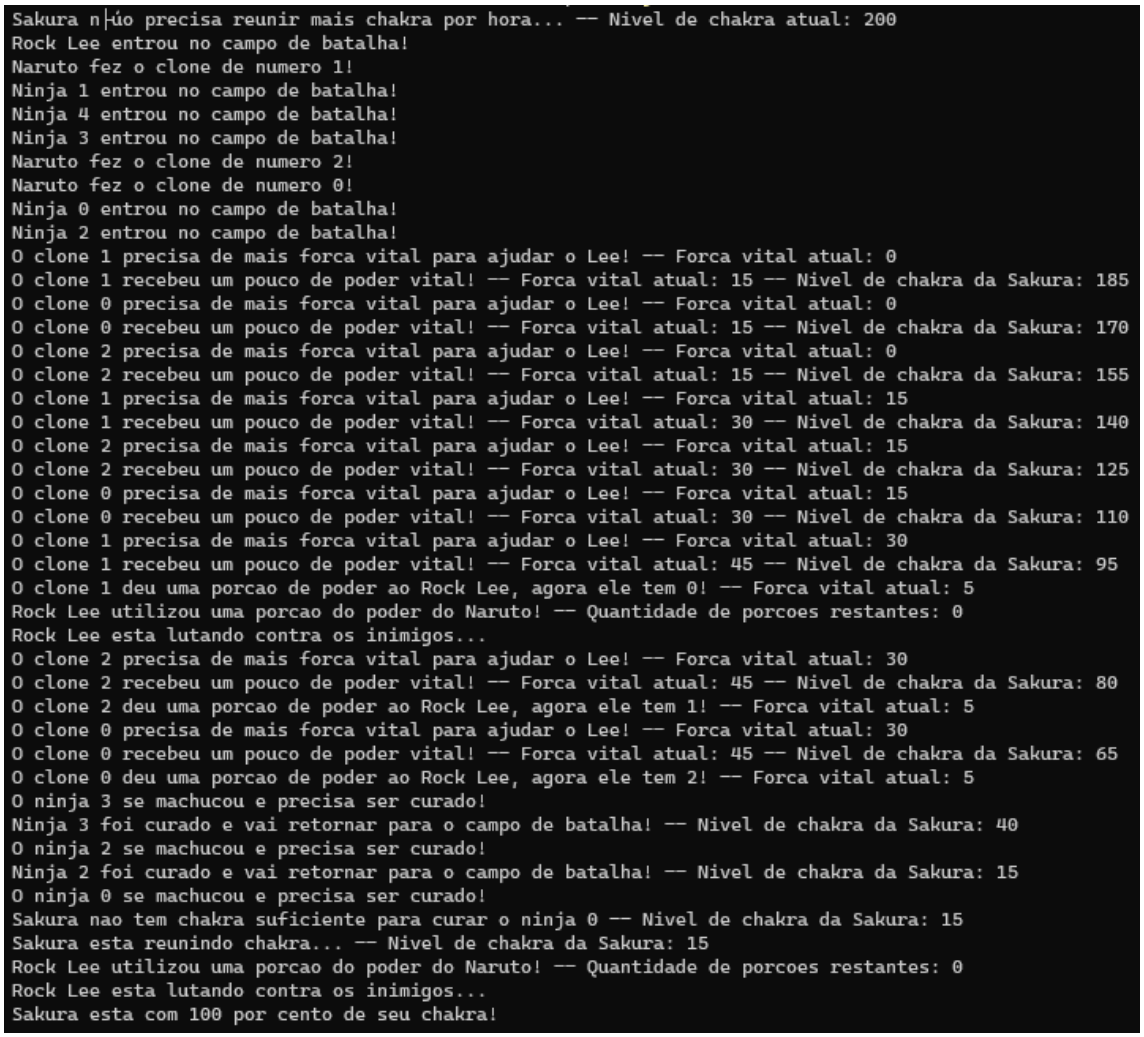

Figura 7. Resultados do programa executado no terminal.

Vamos entender o que significa cada frase:

1. ***"O ninja X entrou no campo de batalha!"***
   Indica que uma thread representando um ninja comum foi criada e entrou em sua respectiva função.

2. ***"Naruto fez o clone de número X"***
   Indica que uma thread representando um clone do Naruto foi criada e entrou em sua respectiva função.

3. ***“Rock Lee entrou no campo de batalha”***
   Indica que a thread representando o Rock Lee foi criada e entrou em sua respectiva função.

4. ***“O clone X precisa de mais forca vital para ajudar o Lee! -- Forca vital atual: Y”***
   Indica quando o clone ainda não atingiu os 40 pontos de poder vital necessário para doar uma porção de poder ao Lee. Como iniciam o programa com o poder vital em 0, é a primeira frase que é printada quando entram na região crítica.

5. ***“O clone X recebeu um pouco de poder vital! -- Forca vital atual: Y...”***
   Indica quando um clone consome o recurso compartilhado (chakra da sakura) e ganha 15 pontos em seu poder vital.

6. ***“O clone X deu uma porcao de poder ao Rock Lee, agora ele tem Y!...”***
   Indica que o clone perdeu 40 pontos de força vital e incrementou a permissão do semáforo que conta as porções de poder do Lee, permitindo que ele avance.

7. **“*Rock Lee utilizou uma porcao do poder do Naruto!...*”**
   Indica que o Lee decrementou o número de permissões do semáforo e portanto utilizou uma porção de poder.

8. **“*Rock Lee esta lutando contra os inimigos...*”**
   Indica que Lee está fora da região crítica protegida pelo semáforo e logo seu poder irá acabar.

9. **“*O ninja X se machucou e precisa ser curado!*”**
   Indica que o ninja entrou na região crítica protegida pelo mutex `healing_zone` e está prestes a testar a condição de parada.

10. ***“Sakura nao tem chakra suficiente para dar curar o ninja...”***
    Indica que o nível de chakra da Sakura é inferior à 25% e portanto o ninja irá acordar ela para que ela possa recarregar.

11. ***“Sakura não tem chakra suficiente para dar força vital ao clone...”***
    Indica que o nível de chakra da Sakura é inferior à 15% e portanto o clone irá acordar ela para que ela possa recarregar.

12. ***“O clone X nao pode ser curado pois ha um ninja machucado...”***
    Indica que existem 1 ou mais threads de ninjas dentro da região crítica esperando para utilizarem o recurso compartilhado, portanto o clone deve adormecer em `pthread_cond_wait(&naruto_clone_cond, &healing_zone)`.

Os resultados analisados foram de uma simulação iniciada com estas constantes:

a. `NUM_REG_NINJAS` = 5

b. `NUM_CLONES` = 3
c. `CHAKRA_TO_HEAL_NINJA` = 25
d. `CHAKRA_TO_VITALIZE_CLONE` = 15
e. `MAX_CHAKRA_LEVEL` = 200
f. `POWER_TO_HELP_LEE` = 40

Você pode encontrar mais explicações e ver ao vivo a simulação em vídeo, além de encontrar o código fonte nesse repositório do Github.

4. Conclusão

Após a conclusão deste projeto, conclui-se que o sistema desenvolvido cumpre o objetivo proposto com sucesso. Através do uso de lock, variáveis condicionais e semáforos pôde-se trabalhar com threads e garantir o acesso e a modificação de regiões críticas de forma concorrente, sem nenhum sinal de deadlock ou starvation. Também não ocorre espera ocupada ou qualquer outro tipo de problema.

Referências Bibliográficas

1. Aulas online no Aprender3
2. Aulas presenciais