Universidade de Brasília

Departamento de Ciência da Computação

Trabalho 1 - Programação Concorrente

Autor:

João Victor Pinheiro de Souza 18/0103407

Brasília
16 de outubro de 2021

Sumário

1 Introdução

Programação concorrente do inglês Concurrent Programming, onde Concurrent significa "acontecendo ao mesmo tempo", sendo que pode ser visto como se tivesse vários fluxos de execução(threads), no qual elas acontece no mesmo tempo lógico.

Tem muitas razões para utilizar programação concorrente, por exemplo em sistemas nos quais existem vários processadores é possível aproveitar esse paralelismo e acelerar a execução do programa, também é utilizado para economizar dinheiro e tempo, resolver grandes problemas complexos que é impraticável ou impossível resolvê-los num único computador, especialmente quando a memória é limitada, Superar as limitações da computação sequencial onde razões físicas e práticas restringem a construção de computadores cada vez mais rápidos.

Por ter vários fluxos de de execução, precisa pensar como vai funcionar de forma simultânea, além disso ele carrega os erros dos programas sequenciais mais os erros associados às interações entre os processos.

No trabalho que foi disponibilizado pelo Professor Eduardo A. P. Alchieri, o aluno deverá criar um problema de comunicação entre processos através de memória compartilhada que envolva condições de corrida, onde deverá elaborar um algoritmo que solucione este problema utilizando os mecanismos de sincronização entre processos estudados em aula (locks, variáveis condições, semáforos, etc.).

2 Formalização do Problema Proposto

O problema consiste em uma campanha de vacina, onde existe um determinado número de pessoas e um determinado numero de enfermeiras, no problema as pessoas vão ter que fazer uma fila para tomar a vacina, sendo que existe uma certa quantidade de doses por dia que são disponibilizados e quando acaba as vacinas o posta é fechado e todos que estava na fila que não tomaram vão ter que ir embora, voltando só no outro dia, criando um loop de repetição dos dias.

E durante esse processo existe outro problema no qual quando uma pessoa chega e observa que a fila está muito grande ele desiste e vai embora, voltando só depois para para ver se está pequena a fila, sendo que ainda existe prioridade na fila, quando uma senhora de idade chega para tomar a vacina, ela entra na frente para tomar primeiro.

3 Descrição do Algoritmo Desenvolvido

No programa é definido a quantidade de pacientes e enfermeiras, para realização do processo, também é definido o tamanho da fila para os pessoas acharem ela grande, e o número de vacinas diárias, onde cada um deles estão definidas respectivamente em **NP**, **EN**, **CAPACIDADE_MAX** e o último para as **VACINAS_DIARIAS**.

Para a resolução do problema foi utilizado três processos, uma para os clientes entrarem na fila, um para as enfermeiras aplicarem as vacinas e a outra para os casos de prioridades em que uma senhora de idade chega para tomar a vacina,sendo que cada um deles vai rodar em seu próprio laço de repetição.

```
 pthread_t c[NP], v[EN], e[NE];
int *id;

    for (i = 0; i < NP; i++){
        id = (int *)malloc(sizeof(int));
        *id = i;
        pthread_create(&c[i], NULL, paciente, (void *)(id));
    }

    for (i = 0; i < EN; i++){
        id = (int *)malloc(sizeof(int));
        *id = i;
        pthread_create(&v[i], NULL, enfermeira, (void *)(id));
    }

    for (i = 0; i < NE; i++){
        id = (int *)malloc(sizeof(int));
        *id = i;
        pthread_create(&e[i], NULL, emergencia, (void *)(id));
    }
```

3.1 Variáveis

As variáveis que foram utilizados nesse projeto compõe-se de Locks, Barreira, condicionas e semáforos

```
sem_t pac;
sem_t enf;
sem_t sem_vaga;

pthread_barrier_t barrier;

pthread_mutex_t lock_fila = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t lock_fila1 = PTHREAD_MUTEX_INITIALIZER;

pthread_cond_t paciente_cond = PTHREAD_COND_INITIALIZER;
pthread_cond_t emergencia_cond = PTHREAD_COND_INITIALIZER;
pthread_cond_t enfermeira_cond = PTHREAD_COND_INITIALIZER;
```

Os semáforos "pac"e "enf"são os controles para liberar ou travar os pacientes e os enfermeiros quando vai tomar a vacina, no "enf"quando o paciente vai tomar a vacina ele acorda a enfermeira e o "pac"é travado para dar tempo de ser aplicado a vacina e quando é concluído libera o "pac"para o paciente ir embora. No semáforo "sem_vaga"serve como um contador indicando quantas pessoas estão na fila.

Nos Locks é repensáveis pela exclusão mutua em regiões críticas, já que estão utilizando variáveis que marca quantas vacinas já foram tomadas, uma que marca prioridade para o senhora entrar na frente. Nas condicionas é feito um controle para o caso em que tenha uma prioridade, colocando o cliente que ia tomar a vacina para dormir, ou quando a meta diária de vacinas é atingida colocando as enfermeiras e os pacientes para dormi.

E por último tem a barreira que é utilizada quando o posto de saúde é fechado, e espera que todos os clientes vão embora para inicializar o outro dia.

3.2 Funções

No trabalho foi dividido em três funções:

```
1 void *enfermeira(void *arg);
2 void *paciente(void *arg);
3 void *emergencia(void *arg);
```

3.2.1 paciente

Na função "paciente"é responsável pelo controle da fila dos pacientes, indicando o tamanho da fila para o caso das pessoas verem ela e decidir se vai embora ou não, dependendo do tamanho dela, sendo que foi utilizado para resolver isso um "try_wait()".

```
1 if (sem_trywait(&sem_vaga) == 0 && aux == 0)
```

Nessa condição ela verifica o tamanho da fila com o semáforo "sem_vaga"e o outro caso é utilizado a variável "aux"que é utilizado se o todas os vacinas foram servidas, dependendo do caso ele entra na fila ou vai embora.

Assim que vai para fila de espera ele entra um um lock para os pacientes entrarem um de cada vez. Então é verificado outra vez se já foram tomadas todas as vacinas, para os que já estão na fila irem embora caso tenha acabado, depois que passa por essa verificação é analisado se tem alguma prioridade em pendência, travando o paciente caso tenha.

Quando terminas as verificação o paciente é chamado para tomar a vacina e na hora em que isso acontece libara uma posição na fila de espera pelo fato que ele deixou a fila para tomar a vacina, e acorda a enfermeira para aplicar a vacina e logo em seguida trava a pessoa, esperando a enfermeira finalizar a aplicação para liberar o cliente e ele ir embora.

Outra rota para os pacientes é quando não tem mais vacinas, então eles são obrigados a irem embora, nessa processo todos são caminhados para uma barreira, esperando que todos os pacientes cheguem nela, Depois que isso acontece é resetados os valores para que comece outro dia da campanha.

3.2.2 emergencia

Nessa função primeiramente é colocado um tempo aleatório para ele demorar um pouco para aparecer usando "sleep(rand() % (i + 1))"onde o "i"seria o id da thread, logo em seguida é verificada se já tomaram todas as vacinas, no qual a variável "aux"é usado como flag para indicar se já acabaram as vacinas.

```
1 while (aux == 1)
2 {
3     pthread_cond_wait(&emergencia_cond, &lock_fila);
4 }
```

para que depois que é analisado vai decidir se coloca a thread para dormi ou para continuar a execução do programa, após isso ele seta a variável "emer"para 1 indicando que a prioridade está acionado e que os pacientes que estavam na fila vão ter que ceder a vez para o senhora que chegou para tomar a vacina, no qual a função faz a mesmo procedimento que os pacientes, acordando a enfermeira e travando a senhora até que seja realizado o procedimento, para em seguida liberar ela, Quando isso acaba é liberado os clientes que cederam a vez para ela, e setar a variável "emer"para 0 mostrando que não tem mais pacientes co prioridades.

3.2.3 enfermeira

Ela começa colocando todas as enfermeira para dormi, sendo só acionado quando o paciente acordar elas, logo depois entra no lock e é analisado se ainda existe vacinas onde vai decidir se coloca pra dormi ou não as enfermeiras, bem parecido com o que foi usado para o de emergência, mudando só as variáveis de condição e o lock. Quando a enfermeira aplica a vacina é acrescentado em um contador quantas vacinas já foram tomadas, até que chega no caso em que é atingido a meta de vacinas diárias onde a variável "aux"e setado para 1 sinalizando que que usaram todas as vacinas, e colocando o posto de saúde para ser fechado, e por fim ele libera o paciente que estava tomando a vacina.

4 Conclusão

Neste trabalho foi colocado em pratica muitos dos conhecimentos que foi usado nos estudos dirigidos que foram passados durante o semestre, como Locks, semáforos, barreiras, e variáveis de condições para a resolução do problema, contudo o trabalho teve uma complexidade a mais em relação ao estudos dirigidos, pelo fato do algoritmo ser relativamente maio com que os já fizera antes na matéria.

E por ter essa complexidade maior, eu tive muitas dificuldades para resolver as resoluções do problema, em relação em como estruturar os threads de forma correta , principalmente em relação quando as vacinas acabam, fazendo com que os que estavam na fila irem embora e resetarem os valores sem com que quebrassem o código, também tive outras dificuldades, em como colocar prioridade para as senhoras de idades colocando os pacientes para dormir e ela seja executada, e em relação aos semáforos com a parte de ativar e desativar elas de forma correta.

Referências

[1] Estudos Dirigidos passados durante o semestre

[2] Aula 8 - Semáforos

[3] Aula 09 - ED7 - Barbeiro dorminhoco (videoaula) - Semana 10 (05/04-11/04)

[4] Estudo dirigido 7 - Problema do barbeiro dorminhoco - Semana 10 (05/04-11/04)

[5] Aula 11 - ED9 e ED10 - Problema do pombo-correio (videoaula) - Semana 12 (19/04-25/04)

[6] Aula 12 - Barreiras (videoaula) - Semana 13 (26/04-02/05)

[7] Aula 06 - Variáveis condição (videoaula) - Semana 7 (15/03-21/03)