Programação Concorrente: Trabalho Final

1st Maria Eduarda Lacerda Dantas
Departamento de Ciência da Computação (CIC)
Universidade de Brasília (UnB)
Brasília, Brasil
maria.dantas@aluno.unb.br

***Abstract*—This document was written with intentions of putting into practice the knowledge acquired in the Concurrent Programming course. The developed project applies locks and semaphores to solve concurrency situations.**

***Index Terms*—projeto final, programação concorrente**

I. Introdução

Diante do semestre cursado, faz-se necessária a aplicação dos conceitos aprendidos através da eloboração de um trabalho prático, o qual consiste na aplicação de ferramentas para concorrência afim de otimizar as atividades exercidas por funcionários do *Twitter*.

Ao longo desse relatório, apresento conceitos e soluções para um problema que coloca em prova o conhecimento do conteúdo de Programação Concorrente [1] ministrado em sala durante o semestre.

II. Problema Proposto

O problema tratado nesse trabalho diz respeito à nova gestão da rede social *Twitter* pelo bilionário Elon Musk. Usuários e funcionários buscam continuar exercendo suas atividades em meio ao caos instituído na empresa. Tais atividades, no entanto, geram situações de concorrência.

Por causa das mudanças feitas pelo novo dono Elon Musk, o site se tornou muito menos confiável e, algumas vezes, instável. Sendo assim, existe um sistema simples implementado na empresa onde usuários são permitidos de escrever uma quantidade limitada de *tweets* no site. Esses *tweets* são, então movidos manualmente para uma *database* pelos funcionários para liberarem espaço para mais *tweets* no site.

Ademais, essa *database* pode atingir sua capacidade máxima. Visando diminuir a caraga de trabalho dos funcionários, os mesmos propuseram a Elon Musk que lesse os *tweets* da *database* quando essa ficasse cheia, pois sabem que o bilionário se irritará com os tweets a apagará todos. Poupando, assim, o trabalho extra dos funcionários de esvaziar a *database*.

Beatriz e seus seguidores são usuários do *Twitter* e *tweetam* muito. *Tweetam* tanto, na verdade que, muitas vezes, se entediam e fazem uma pausa para ler. Eles estão tentando ficar mais cultos.

Os funcionários também se cansam de aturar Elon Musk e, eventualmente, precisam ir ao psicólogo devido à grande carga de trabalho.

A. Beatriz e Seguidores

Beatriz e seus seguidores (usuários) pensam em um *tweet* para escrever e usam um espaço disponível do *Twitter*, caso exista, para escreverem seus *tweets*.

Se Beatriz ou um de seus seguidores atingir a quantidade necessária de tédio para parar e ler um livro, os usuários fazem uma pausa de alguns segundos e voltam prontamente para continuar seu trabalho de escrita de *tweets*.

B. Funcionários

Enquanto não existe um serviço que mova os *tweets* para a *database* de forma automatizada, os funcionários da empresa trabalham para colocá-los manualmente da base de dados.

Caso haja espaços indisponíveis no *Twitter*, os funcionários levam esses *tweets* para a base de dados, onde, depois de cheia, será consultada pelo Elon Musk.

Caso, ao tentar mover um *tweet*, o funcionário perceba que a *database* está cheia, o funcionário vai dormir e chama o Elon Musk para esvaziar o banco de dados.

Caso um dos funcionários atinja a quantidade necessária de necessidade de ir ao psicólogo, os mesmos fazem uma pausa de alguns segundos e voltam prontamente para continuar seu trabalho na empresa.

C. Elon Musk

Elon Musk comprou o site em um impulso motivado por sua crise de meia idade e agora enfrenta dificuldades para gerenciar a empresa e a integridade do *Twitter*.

Seu papel consiste em ler os *tweets* armazenados na *database* pelos funcionários, se irritar com as bobeiras lidas e, em um impulso de raiva, apagar todo o material guardado.

Caso Elon Musk chegue a empresa e a *database* não esteja cheia, Elon Musk volta outra hora para ler os tweets. O empresário só deseja lê-los quando o banco de dados que os armazena esteja cheio.

III. Solução Apresentada

Diante do problema proposto anteriormente, foi desenvolvida uma solução para a situação buscando otimizá-la utilizando concorrência entre agentes, evitando condições de corrida.

A solução desenvolvida utiliza a biblioteca *POSIX Pthreads* e será tratada a fundo nessa seção do relatório.

A. Leitura de arquivos

Os nomes de usuário e os *tweets* estão armazenados em arquivos de texto, são lidos e guardados em um *array* de nomes ***usernames*** e ***tweets***.

B. Inicialização de recursos

A solução desenvolvida utiliza-se dos seguintes elementos para estabelecer concorrência de processos, os quais são inicializados posteriormente no código:

- Um ***lock*** para acessar variáveis;
- Dois **sefáforos** para contarmos a quantidade de *slots* disponíveis no *Twitter*;
- Duas **variáveis de condição** para controlarmos *wait* e *broadcast/signal* de funcionários e do Elon Musk.
- ***Threads*** para Beatriz e seguidores, funcionários e Elon Musk.

Esses elementos são, mais tarde, inicializados na função ***main*** e, quando necessários armezanados em arrays, como é o caso das várias threads de usuários e funcionários.

C. Processo: Beatriz e Seguidores

O processo de Beatriz e seguidores é descrito pela função ***beatriz_e_seguidores*** mostrada a seguir.

```
void * beatriz_e_seguidores(void *arg) {
    char* user = ((char *) arg);
    int quer_ler = 0;

    while (TRUE) {
    char * tweet = pensar_em_tweet(user);

        sem_wait(&slots_disponiveis);
            print_tweet(user, tweet);
        sem_post(&slots_indisponiveis);

        quer_ler++;

        if (quer_ler == PAUSA_LEITURA)
            lendo_livro(user);
            quer_ler = 0;
    }
}
```

Visando o desenvolvimento de uma solução que descrevesse o problema descrito, foram utilizados dois semáforos para a elaboração desse processo.

No início da função, o usuário pensa em um *tweet* para ser elaborado e armazena o mesmo em uma variável denominada ***tweet***. O usuário então pega um espaço disponível no *Twitter*, caso exista um, e escreve um *tweet*. Ademais, o mesmo incrementa o valor do semáforo que representa a quantidade de espaços indisponíveis.

Os usuários, então, incrementam sua vontade de ler um livro e, mais adiante, consultam se a variável ***quer_ler*** é igual ao número estipulado para a pausa da leitura. Caso seja, o funcionário sai por alguns segundos, volta para continuar a elaborar *tweets* e iguala sua vontade de ler a zero novamente.

D. Processo: Funcionário

O processo dos funcionários é descrito pela função ***funcionarios***, a qual será exibida a seguir em partes.

A função é primeiramente iniciada e recebe como argumento a identificação do funcionário e iniciamos um *loop* ***while***, que executará para sempre.

```
void * funcionarios(void *arg) {
    int funcionario = *((int *) arg);
    int psicologo = 0;

    while (TRUE) {
```

A partir dos passos iniciais, o funcionário pega um *slot* (espaço) indisponível do *Twitter* para esvaziá-lo. Além disso, fica em posse de um ***lock*** para acessar a variável ***tweets_que_faltam***.

```
sem_wait(&slots_indisponiveis);

pthread_mutex_lock(&database);
```

O funcionário, então, consulta se a variável ***tweets_que_faltam*** é igual a 0. Caso seja, o funcionário vai dormir e chama o Elon Musk.

```
while (tweets_que_faltam == 0) {
    pthread_cond_signal(&elon_musk);
    pthread_cond_wait(&funcionarios);
}
```

Quando o funcionário sai do *loop*, o mesmo move o *tweet* para o banco de dados e decrementa a variável ***tweets_que_faltam*** já que falta menos um *tweet* para encher o banco de dados.

```
mover_tweets_para_database();

tweets_que_faltam--;

terminou_de_mover_tweets();
```

Após esses passos, o funcionário libera o *lock* para acessar a variável e incrementa o semáforo de espaços disponíveis no *Twitter*. Além de incrementar a necessidade de ir ao psicólogo descrita pela variável ***psicologo***.

```
pthread_mutex_unlock(&database);

sem_post(&slots_disponiveis);

psicologo++;
```

Finalmente, caso a variável ***psicologo*** seja igual ao número necessário para uma pausa, o funcionário vai ao psicólogo por alguns segundos e volta ao trabalho.

```
if (psicologo == PAUSA_PSICOLOGO) {
    indo_pro_psicologo();
    psicologo = 0;
}
```

Terminando, assim, o processo de *funcionarios*.

E. Processo: Elon Musk

O processo que descreve o comportamento de Elon Musk é descrito pela função ***elon_musk*** demonstrada abaixo.

```
void * elon_musk(void *arg) {
while (TRUE) {
    pthread_mutex_lock(&database_twitter);

      while (tweets_que_faltam != 0) {
          pthread_cond_wait(&elon_musk);
      }

    pthread_mutex_unlock(&database);

    elon_musk_lendo_tweets();

    pthread_mutex_lock(&database_twitter);

    tweets_que_faltam = TAMANHO_DATABASE;
    pthread_cond_broadcast(&funcionarios);

    pthread_mutex_unlock(&database);

    elon_musk_esvaziou_database();
}
}
```

Elon Musk usa um *lock* para consultar a variável ***tweets_que_faltam***. Caso essa seja zero, Elon Musk dorme até que os funcionários o acordem.

Ao sair do loop, Elon Musk, então lê os *tweets*, se irrita com o que foi escrito e toma posse do *lock* mais uma vez, alterando o valor da variável ***tweets_que_faltam*** para o valor que corresponde a capacidade total do banco de dados já que o mesmo foi esvaziado pelo bilionário. Além disso, Elon Musk acorda todos os funcionários e anuncia que esvaziou a *database*.

IV. Conclusão

Diante da elaboração do problema e da solução proposta, podemos colocar em prática muitos dos ensinamentos ministrados em sala durante o semestre.

Ao usar *locks* e semáforos, faz-se compreensível a utilização dos mesmos. Tornando o aprendizado mais fluido e natural.

References

[1] Wikipedia, “Programação Concorrente,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, Dezembro 2022.

[2] Ben-Ari, M, “Principles of Concurrent and Distributed Programming,”, Prentice Hall; 2nd ed. , 2006.

[3] Gregory Andrews, “Concurrent Programming: Principles and Practice,” in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[4] Breshears, C., “The Art of Concurrency: A Thread Monkey’s Guide to Writing Parallel Applications,” O’Reilly, 2009.