Universidade de Brasília

Departamento de Ciência da Computação

Projeto - Programação Concorrente

Autor:

Carlos Eduardo de Oliveira Ribeiro 18/0099094

Brasília
10 de maio de 2021

1 Introdução

A execução simultânea de várias tarefas do computador pode ser simulada e implementada como fluxos de execução diferentes chamado de *threads* que são criadas por um único programa. As *threads* são definidas como tarefas diferentes que um mesmo programa realiza, podendo assim criar uma interação e dividir processos em tarefas que podem ser executadas concorrencialmente.

Com o uso de diferentes tarefas se relacionando, pode-se criar vários programas com um problema de implementação onde os vários fluxos de execução precisam ser controlados.

Nesse contexto, o objetivo do projeto é criar um problema onde existe uma desorganização em um sistema, como condições de corrida. E conseguir desenvolver uma solução com os recursos aprendidos no decorrer do semestre,.

2 Desenvolvimento

2.1 Descrição do Problema

Uma empresa investiu muito bem seu dinheiro e fez uma aquisição de uma nova máquina de café, porém os funcionários e gerente de um departamento gostaram do novo método de fazer café, com isso, durante todo o dia existe uma grande fila para pegar o café. Logo, existe um problema, sempre que é servido uma quantidade grande de café, a cafeteira resolve emperrar fazendo assim o dono da empresa contratar um técnico para ficar sempre disponível caso o evento ocorra, os funcionários têm mais tempo livre então esperam até o técnico terminar o trabalho, enquanto o gerente vai embora pois tem bastante trabalho para fazer.

Na empresa existe um nível de cargo muito respeitado, logo para que os funcionários não sejam despedidos, o gerente sempre têm preferencia na fila para pegar o café.

2.2 Soluções para o Problema

Neste problema, as condições de corrida são bem perceptíveis, já que existe uma única área de acesso para três tipos de usuários, ou seja, uma maquina de café que será usada pelos funcionários e gerente, e um técnico que interceptará o acesso quando a mesma quebrar.

Em geral, para que esse problema fosse resolvido, primeiramente foi tratado o acesso mutuo a fila de funcionários que estão a espera do usuário que está usando a máquina de café. E para a próxima pessoa utilizar a máquina de café, ela precisa da permissão do usuário que acabou de usar, com isso sempre dando a opção de dar permissão ao gerente, caso ele esteja esperando ou dar a permissão ao próximo da fila.

Mas antes do usuário utilizar de fato a máquina de café, ela precisa de uma verificação para saber se vai funcionar, pois no programa é definido a quantidade de vezes que ela vai demorar para quebrar. Quando essa verificação falhar, ou seja, a maquina de café quebrar, é liberado o técnico que consertará a maquina, dando novamente o mesmo número de acessos, e disponibilizando o acesso dos funcionários e do gerente a máquina.

2.3 Explicação do Algoritmo

Inicialmente para compreensão do algoritmo será analisado todas as constantes e variáveis declaradas e por fim suas funções.

Declarações

Primeiramente é feita a declaração de variáveis que não serão mudadas no decorrer do código. As variáveis a seguir são respectivamente numero de funcionários e a quantidade de cafés que a maquina vai servir antes de quebrar.

```
#define N_F 15
#define MAQUINA 20
```

Logo após essa declaração temos os *locks* e variáveis de condição utilizados no código, os *locks* são para o uso exclusivo da maquina de café e da fila de funcionários, enquanto a variável de condição é para o usuário dormir quando a máquina não estiver operando.

```
pthread_mutex_t maquina_cafe = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t fila_lock = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t funcionario_cond = PTHREAD_COND_INITIALIZER;
```

Por fim tem-se a inicialização dos semáforos e criação das threads. Os semáforos inicializados em 0 são aqueles que não precisam iniciar o programa ativo, o técnico e a espera do gerente caso a máquina estivesse quebrada, como ela inicia funcionando, os dois iniciam inativos.

Os que iniciam ativos são as permissões para acesso da máquina de café, o gerente, o funcionário e o que inicia com o valor armazenado em "MAQUINA", que é a quantidade de vezes que a máquina vai funcionar até quebrar.

E as *threads* declaradas são respectivamente o vetor de funcionários e sua quantidade declarada, o gerente e o técnico. E suas criações se relacionam as funções com o mesmo nome.

```
sem_init(&tecnico, 0, 0);
sem_init(&gerente_perm, 0, 0);
sem_init(&funcionario_sem, 0, 1);
sem_init(&gerente_sem, 0, 1);
sem_init(&conserto, 0, MAQUINA);

pthread_t funcionarios[N_F];
pthread_t gerente;
pthread_t tecnico;
```

Funcionários

Para esta função primeiramente foi garantida a exclusão mutua para o incremento da fila do usuário, após isso, é verificado se o técnico está "acordado" arrumando a máquina, se estiver, o usuário vai esperar "dormindo" a máquina ficar pronta para ele usar.

```
pthread_mutex_lock(&fila_lock);
fila++;
sleep(1);
if(tecnico_acordou == 1){
    printf("Maquina quebrada, funcionario %d espera\n", id);
    pthread_cond_wait(&funcionario_cond, &fila_lock);
}
pthread_mutex_unlock(&fila_lock);
```

A próxima etapa é a utilização da máquina de café, primeiramente é esperado uma permissão para acesso da máquina de café, indicando que a mesma está livre. A seguir, tem um *mutex* para assegurar o acesso exclusivo da região, após isso, tem-se um decremento da fila, então é dado um "*sem_try_wait*" para saber se a máquina está funcionando já aplicando o decremento de permissão no semáforo, então a última verificação da função é para saber se existe gerente esperando para usar a máquina, se sim, ele usaria já que tem prioridade no problema, se não o próximo usuário da fila utiliza,

e por fim, caso o "*sem_try_wait*" desse errado, o técnico seria assionado para arrumar a máquina.

```
sem_wait(&funcionario_sem);
pthread_mutex_lock(&maquina_cafe);
fila--;
if(sem_trywait(&conserto) == 0){
    printf("Funcionario %d se servindo, fila: %d", id, fila);
    sleep(1);
    if(gerente == 1){
        printf("%d: Vou dar permissao para o gerente\n", id);
        sem_post(&gerente_perm);
    }else{
        sem_post(&funcionario_sem);
    }
}else{
    printf("Funcionario %d: Chamando o tecnico\n", id);
    sem_post(&tecnico);
}

pthread_mutex_unlock(&maquina_cafe);
```

Gerente

A lógica da função gerente é muito parecida com a funcionário, porém, não precisamos de acesso exclusivo pois só existe um gerente.

Inicialmente é verificado uma variável para saber se tem técnico trabalhando na máquina. Caso exista, o gerente desiste do café e vai embora, pois o técnico demora muito, caso contrário é atualizado a variável gerente para servir de "flag" para a preferência, avisando que ele está querendo utilizar a máquina. Então é esperado sua permissão para acessar a máquina, após isso a lógica é a mesma do funcionário, "*sem_try_wait*" para saber se a maquina está funcionando, caso não esteja, chama o técnico. Única diferença é que não precisa dar preferência pra ninguém, ele dá permissão para o próximo da fila utilizar a máquina.

```
if(tecnico_acordou == 1){
    printf("Maquina quebrada, gerente vai embora\n");
    sem_wait(&gerente_sem);
}else{
    printf("Gerente quer cafe\n");
    gerente = 1;
    sem_wait(&gerente_perm);
```

```
    if(sem_trywait(&conserto) == 0){
        gerente = 0;
        printf("Gerente se servindo\n");
        sleep(1);
    }else{
        printf("Gerente foi chamar o tecnico\n");
        sem_post(&tecnico);
        gerente = 0;
    }

    printf("Vou dar permissao para um funcionario\n");
    sem_post(&funcionario_sem);
}
```

Técnico

O técnico inicia travado em um "*wait*" para só ser executado quando requisitado, em seguida é atualizado sua "$flag$" para travar as funções de funcionário e gerente. Após isso, ele espera a máquina de café parar de ser requisitada com a aplicação do "*lock*", logo ele faz com que o semáforo tenha todas as suas permissões realocadas para o uso, então ele dar um "*broadcast*" para acordar todos os usuários que estavam ali dormindo e o "*post*" para liberar o gerente em sua função. Por fim, ele libera o funcionário que estava esperando a permissão. O gerente não é liberado porque ele não espera o técnico terminar seu reparo.

```
sem_wait(&tecnico);
tecnico_acordou = 1;
pthread_mutex_lock(&maquina_cafe);
    printf("Tecnico consertando\n");
    sleep(8);
    for(int i = 0; i < MAQUINA; i++){
        sem_post(&conserto);
    }
    pthread_cond_broadcast(&funcionario_cond);
    sem_post(&gerente_sem);
    tecnico_acordou = 0;
pthread_mutex_unlock(&maquina_cafe);

sem_post(&funcionario_sem);
```

3 Resultados Obtidos

Quando o programa é iniciado é possível ver os funcionários iniciando uma fila que não aumenta pelo tempo deles se servirem ser o mesmo da fila. Na figura 1 é explicito a preferência do gerente, existe uma fila de 1 funcionário, mas quando o gerente avisa que quer o café, o funcionário 2 o concede a permissão para o uso da cafeteira.

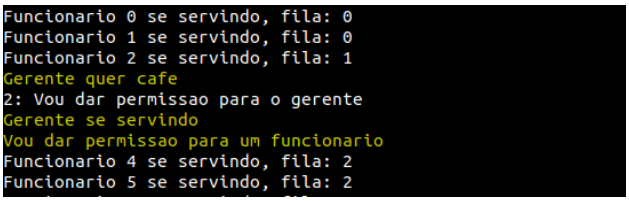
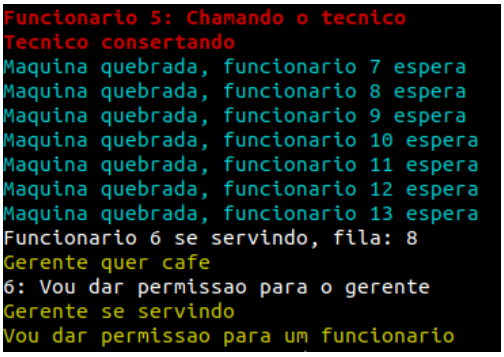


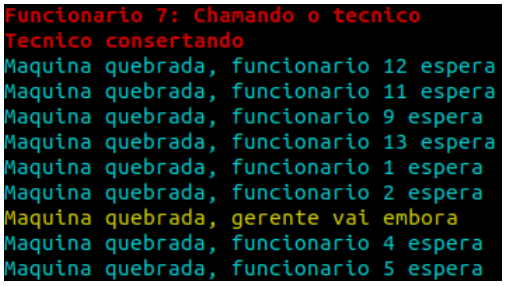
Figura 1: Preferencia do Gerente

Na figura 2a é perceptível o funcionário chamando o técnico ao perceber que a máquina está quebrada, logo os funcionários são adormecidos e a permissão do gerente sendo atendida na frente dos funcionários.

E o último resultado (2b) a ser observado é o gerente indo embora ao perceber que o técnico está arrumando a máquina.



(a) Funcionários em espera.



(b) Gerente em espera.

Figura 2: Técnico colocando os funcionários e o gerente em espera.

4 Conclusão

Concluo que o trabalho foi finalizado com sucesso, conseguindo colocar em prática todos os ensinamentos passados durante o semestre para o manuseio de *threads* na resolução de problemas. Apareceram várias dificuldades fazendo o problema proposto, existem várias formas diferentes de fazer o mesmo problema, com isso, tentei utilizar o máximo de ferramentas diferentes possível visto na disciplina, mas acredito que consegui fazer um bom trabalho contornando os problemas e espero que esse relatório tenha deixado o código mais simples de entender.

Referências

[1] Eduardo Alchieri. Slides Programção Concorrente. https://cic.unb.br/ alchieri/disciplinas/graduacao/pc/pc.html