Universidade Federal de Minas Gerais
Disciplina: Sistemas Operacionais
Aluna: Vitória Mirella Pereira do Nascimento
Matrícula: 2020420656
Professor: George Luiz

Relatório
Trabalho Prático 1

Belo horizonte
31/01/2021

I. O problema

O desafio proposto objetiva exercitar a programação de sincronização entre processos e os efeitos da programação concorrente. Para tal, foi proposto o problema de organizar o uso do forno micro-ondas pelos personagens de Big Bang Theory, respeitando algumas precedências pré-definidas. Para tal, precisava-se usar conceitos como threads, inanição, deadlock, sincronização, dentre outros.

II. Compilando e executando

Para compilar o trabalho, é necessário ir até o diretório principal e utilizar o Makefile. Dessa forma, será possível executar com os seguintes comandos:

make clean

make && make run ARGS=<parametro>

O comando **make clean** vai remover qualquer arquivo **.o** que já exista, enquanto o comando **make && make run ARGS=<parametro>** vai compilar todos os arquivos, criar todos os .o necessários e rodá-los, onde <parametro> se refere à quantidade de vezes que cada personagem vai usar o forno.

III. A implementação

Na realização do trabalho, foram usados alguns módulos, de forma a facilitar e organizar o código. Todos estão cuidadosamente comentados e portanto é possível obter mais detalhes olhando o código em si.

III.I Arquivo threadManipulations

Responsável por criar as threads dos personagens, criar a thread Raj, colocar as threads para dormir quando necessário e chamar os métodos do forno.

Especificando melhor, a função create threads vai criar o array de threads e inicializar cada um passando a função a ser executada como parâmetro.

A função runThreads vai ser executada por todos os personagens do problema, com exceção do Raj. Nesse sentido, o personagem vai tentar acessar o forno pelo número de vezes que é passado pra ele. Após esquentar algo, vai comer e depois trabalhar.

A função runRaj vai ser responsável por procurar por um deadlock a cada 5 segundos.

III.II Arquivo utils

Responsável por criar funcionalidades úteis que serão usadas pelos outros dois métodos principais. Nele, cria-se a função que pega o nome do personagem dado seu id e a função que retorna um número aleatório em um

intervalo passado por parâmetro (esse valor é usado normalmente nas chamadas de usleep).

III.III Arquivo monitor

Responsável por implementar todas as funcionalidades e métodos do monitor do forno. Nela, são definidos e usados mais efetivamente os conceitos de mutex e variáveis de condição. De maneira geral, o monitor é responsável por escolher o próximo a esquentar a comida, delegar acesso e checar deadlocks.

A estrutura conditions contém as variáveis de condição para cada um dos 8 personagens do problema.

O método getLock controla o acesso ao forno. Ele insere o personagem que pediu acesso na fila de espera e o retira quando for selecionado.

O método unlock é responsável por liberar o forno. Além disso, chama a função para calcular o próximo personagem a ser selecionado para esquentar a comida.

O método setRajChoice é chamado por Raj quando o mesmo detecta um deadlock e recebe um valor aleatório contendo o personagem que deve ser liberado para usar o forno.

O método calculateNextToUse calcula a próxima pessoa a usar o forno. Esse método leva em conta todos os personagens na fila de espera.

O método checkForDeadlock verifica se há um deadlock na fila de espera.

Alguns outros métodos mais específicos de implementação podem ser encontrados de maneira bem explicada no código.

III.IV - Main

Arquivo principal responsável por tratar o argumento passado por parâmetro e chamar a função createThreads.

IV. Decisões de Projeto

- A ordem dos personagens no vetor de personagens é um ponto importante. Ele foi criado naquela ordem para que, caso não haja nenhum corner case, aquela ordem sirva para nos mostrar as precedências. Os personagens de menor id têm maior precedência que os de maior id, salvo nos casos de namorada e namorado, nesses casos a precedência é definida por quem entrou primeiro na fila.
- Citando novamente a estrutura de conditions, ela foi criada para armazenar uma variável de condição para cada personagem, e tem como utilidade definir se o personagem pode ou não entrar no forno depois de pegar o lock.

- As namoradas sozinhas, sem seus respectivos namorados, têm a mesma prioridade que eles teriam, então as 3 sozinhas na fila esperando para usar o forno são um deadlock.
- O próximo personagem a usar o forno é sempre calculado por uma função implementada e é chamada sempre que alguém devolve o lock do forno.
- A função que vai tratar a escolha do Raj de quem sairá do deadlock manda um sinal que estará sendo aguardado pela função que calcula a próxima pessoa a usar o forno.
- A função de checar por deadlock retorna um vetor com as partes envolvidas no deadlock para que um deles seja escolhido pelo Raj.
- Na hora de escolher o próximo personagem a ser pego na fila, olha-se para a fila e conta a quantidade de casais. Após isso, divide-se os em casos: nenhum casal, 1 ou 2 casais. Não tem o caso dos 3 casais, pois o check do deadlock ocorre antes de se fazer o cálculo do próximo da fila.
- Existe uma variável que diz a qualquer momento se o forno está ou não ocupado. Foi a forma encontrada para tratar o seguinte caso: Quando uma pessoa entra na fila e esta está vazia naquele momento, na teoria ela poderia diretamente usar o forno, contudo, se já tiver alguém no forno, não se quer que essa pessoa vá direto. Dessa forma, usa-se a variável citada junto ao tamanho da fila para definir se alguém pode ir diretamente para o forno (sem passar por nenhum tipo de checagem prévia).
- Sempre na hora de enviar um sinal no código, espera-se que a outra parte escute esse sinal. Para fazer isso, tem-se uma variável que representa uma certa condição e um while infinito é feito enquanto essa condição não for verdadeira.
- Todas as funções que envolvem arrays ou a struct estão protegidas por mutex. Tem-se um mutex para o vetor que representa a fila de espera, um para o vetor que representa quem está esperando por um sinal para usar o forno e um para deadlock.

V. Bugs

- Quando há um casal na fila, se um membro do casal usar o forno, o próximo não necessariamente vai usar.
- O código não respeita a condição de o Leonard vir antes do Sheldon, isso também vale para as respectivas namoradas. O motivo é que isso iria infringir a definição das precedências no vetor (Leonard tem id 4 e Sheldon tem id 0). No algoritmo, sempre considera-se o menor id tendo maior prioridade.