UNIVERSIDADE FEDERAL DE MINAS GERAIS
Instituto de Ciências Exatas
Departamento de Ciência da Computação
DCC605 -- Sistemas Operacionais
Trabalho Prático 1
Guilherme de Abreu Lima Buitrago Miranda - 2018054788
Victor Hugo Silva Moura - 2018054958

1 - Introdução

No primeiro trabalho da disciplina de Sistemas Operacionais, os personagens da série de TV The Big Bang Theory compraram um microondas e, para organizar a fila de prioridades, foi criado um programa. Nele, além de observar a ordem de precedências entre personagens e/ou casais de personagens, para exercitar a programação de sincronização entre processos e os efeitos da programação concorrente, foi utilizada a biblioteca POSIX (pthreads).

Assim, cada personagem que pretende usar o forno é tratado como um processo e, para tal, uma solução em C++ foi desenvolvida. Na seção imediatamente abaixo, encontram-se os detalhes da implementação, seguidos pelos desafios enfrentados, conclusões e referências bibliográficas.

2 - Implementação

Para a implementação do trabalho, em primeiro lugar, foi necessário criar a classe forno, que implementa o padrão de projeto Singleton, ou seja, durante toda a execução do programa, apenas uma instância da classe é criada. Além disso, a classe também tem um vector de strings que representa a fila de espera e variáveis de controle pthread_mutex_t ou booleanos. Por fim, ela também implementa os principais métodos que os personagens utilizam, como o método esperar, liberar, verificar, insereFila e stuartKripke.

O primeiro deles é o chamado quando o personagem quer usar o forno. Assim, dentro dele, é chamado o método insereFila (posteriormente detalhado) e, logo em sequência, há um while, responsável por verificar duas condições: se o personagem está no topo da fila de prioridades e se não há um deadlock. Caso ambas sejam verdadeiras, o personagem pode, finalmente, usar o forno.

O método liberar também é bastante simples, e é executado logo após o personagem acabar de esquentar sua comida, apagando o mesmo da fila. Contudo, essa remoção só acontece enquanto não houver um deadlock. Essa condição foi necessária pois, caso haja um deadlock e o personagem saia da fila, seu respectivo companheiro perderia a prioridade e seria direcionado a uma das últimas posições da fila, pois não está mais em casal.

Já o método insereFila, são feitas uma série de comparações para verificar quem é o personagem, quem é seu companheiro e, por fim, se seu companheiro está na fila, conferindo prioridade a ambos. Logo em sequência, o personagem (e possivelmente seu companheiro) é inserido na fila de espera, observando, evidentemente, as precedências apresentadas no enunciado do trabalho. Tal método também observa a presença de deadlocks e sinaliza uma variável booleana de controle como verdadeira, para que o Raj, quando chamado, possa resolvê-lo. Por fim, caso o personagem a ser inserido seja o último da fila (a frente apenas do Stuart e do Kripke, se eles estiverem presentes), é chamado o método stuartKripke, que é responsável por fazer a inserção do mesmo no lugar correto.

Por último, o método verificar é chamado apenas pela thread do personagem Raj. Nele, caso a variável booleana de controle acuse um deadlock, um personagem aleatório é escolhido na fila para passar a frente de todos (caso esse personagem esteja em casal, o seu companheiro também ganha prioridade). Em seguida, é necessário reorganizar a fila, pois, caso a fila tenha Sheldon, Howard e Leonard, por exemplo, e o Howard seja sorteado para ir primeiro, a fila precisa ser reorganizada para que fique na ordem Howard, Leonard e Sheldon.

Tratando, agora, do código presente no arquivo main.cpp, além de receber, via argumentos na linha de comando, o número de vezes que cada personagem utilizará o forno, o mesmo cria as threads para cada um dos personagens. Tais threads são definidas com o método *personagem_thread, que é responsável por chamar os métodos principais da classe forno (esperar e liberar). Além disso, no main está definida a struct personagem_data, que é passada para a thread, com o nome do personagem e o número de execuções do mesmo. A *raj_thread também está definida no main e é executada a cada 5 segundos, chamando o método verificar da classe forno.

Por fim, é interessante observar que, ao longo do código, existem sleeps com tempos aleatórios, a fim de tornar diferente as execuções do programa, para que se possa observar como são tratadas várias configurações de personagens na fila.

3 - Desafios

Dentre os principais desafios enfrentados durante o trabalho, destacam-se os aspectos relacionados ao entendimento do enunciado do trabalho, assim como as regras de precedências e seus casos mais extremos, eventualmente não contemplados pelo documento. Além disso, a programação concorrente com threads e o lock em estruturas de dados também se mostrou bastante desafiadora, sobretudo no início do trabalho.

Para o primeiro desafio, consideramos as seguintes regras:

- Um homem e uma mulher, de casais diferentes, sozinhos na fila, tem a mesma prioridade de caso fossem apenas homens ou apenas mulheres. Em outras palavras, sem a presença de seus respectivos companheiros, Amy tem preferência com relação a Howard;
- Caso aconteça um deadlock entre pessoas sozinhas e o Raj resolver mas, nesse meio tempo, o par de outra pessoa chega, a prioridade criada pelo Raj é perdida;
- O Raj nunca sorteia o Stuart ou o Kripke para resolver um deadlock (ainda que eles estejam na fila) pois, caso o fizesse, depois do uso do forno, um novo deadlock iria se configurar.

Já para o segundo desafio, a leitura da documentação da biblioteca, assim como fontes de pesquisa como o GeeksForGeeks foi bastante valiosa. Em particular, buscou-se implementar programas bem mais simples, apenas para observar o comportamento das variáveis do tipo pthread_mutex_t para depois, finalmente, utilizá-las no presente trabalho.

4 - Conclusões

Para além da prática com a linguagem C++, a implementação do trabalho foi bastante proveitosa para compreender efetivamente os conceitos vistos até o presente momento na disciplina. O aprendizado se deu, sobretudo, no último desafio citado na seção anterior, mas também com outras dúvidas dos colegas

apresentadas no grupo do WhatsApp. Com isso, considera-se que o trabalho foi de grande proveito para o processo de aprendizagem.

5 - Referências Bibliográficas

- Mutex lock for Linux Thread Synchronization - https://www.geeksforgeeks.org/mutex-lock-for-linux-thread-synchronization/
- std::find - http://www.cplusplus.com/reference/algorithm/find/