Trabalho Prático 1 - Threads
Sistemas Operacionais - 2020/2

Catarina Enya Diniz Pereira - 2017093402
Vitor Lucio dos Santos Ferreira - 2016006999

Este trabalho tem por objetivo exercitar a programação de sincronização entre processos e os efeitos da programação concorrente. Isso deve ser feito utilizando-se os recursos de threads POSIX (pthreads) -- em particular, mutexes (pthread_mutex_t) e variáveis de condição (pthread_cond_t).

O presente trabalho foi implementado utilizando-se da linguagem C++ e sua biblioteca padrão, além das bibliotecas de pthread e mutex que auxiliam na criação e gerenciamento das threads bem como sincronização dos dados.

Resumo de Projeto

Dado o problema de utilização do forno, para implementar os personagens do Big Bang Theory como threads e implementar o forno como um monitor, utilizou-se de pthreads, mutexes e um conjunto de variáveis de condição para controlar a sincronização do acesso dos casais ao forno.

Para tanto, os seguintes pressupostos foram considerados:

- Basicamente, todos os personagem tem um único interesse em comum que é saber se, na hora em que o personagem quiser o forno, lhe será permitido usá-lo.
- Todo personagem precisa de seu próprio recurso compartilhado que informa seu interesse no forno. Tal recurso será usado por pelo menos duas threads: a do personagem em questão e a do personagem que quer saber se o primeiro está querendo usar o forno.
- O personagem estar na fila é equivalente a dizer que o mesmo está querendo usar o forno, ou seja, ter manifestado a intenção de utilizar o forno. Portanto, tal equivalência deve ser considerada durante todo o corpo do presente relatório.

Levando em consideração que os personagens possuem comportamentos semelhantes, exceto pelo Raj, optou-se por definir uma classe Person que representa todos os personagens e outra classe Microwave que representa o forno (monitor), cada qual com suas respectivas ações disponíveis. Considerando ainda a modelagem adotada nas classes, para o segundo pressuposto, criou-se uma variável booleana para cada personagem indicando se a intenção de usar o forno foi manifestada pelo personagem ou não.

Para a variável booleana ligada a cada um dos nove personagens, criamos um array de mutex e array de variáveis de condição para fins de controle das variáveis que potencialmente serão acessadas por várias threads, garantindo que os recursos compartilhados sejam acessados por uma thread de cada vez. Além disso, para registrar a ordem de chegada dos personagens do casal, o momento em que o casal está efetivamente completo e a intervenção do Raj em caso de deadlock, criou-se variáveis globais e mutexes para garantir a sincronização dos dados quando acessados por mais de uma thread.

Na definição das prioridades de cada personagem optamos por criar não uma lista circular mas uma função que, dado o código identificador do personagem, informa qual outro personagem detém a prioridade sobre ele. Tal função *prioridades* segue as regras apresentadas no escopo do trabalho prático e adicionam casos especiais, como o do Stuart e Kripke que possuem mais de um personagem detentor de prioridade sobre eles, e são colocados como sua própria prioridade para tratamento nas funções subsequentes.

Uma variação desta mesma função *prioridades*, mas agora voltada para a definição de *dependentes* também foi implementada para fins de representação da prioridade inversa entre os personagens. O mesmo vale para a função *casais*, variação da função *prioridades*, implementada no contexto da formação de casais do trio principal (Sheldon, Howard e Leonard).

Como tratamento efetivo das prioridades e com auxílio das funções mencionadas acima, a função *trata_prioridades* considera que, sempre que é executada, o desejo do personagem é saber se o mesmo pode utilizar o forno neste exato instante da fila. Para definir a ordem de prioridades com base na manifestação do personagem, faz-se necessário verificar a manifestação da intenção de usar o forno de todos os personagens, exceto pelo Raj, Stuart e Kripke. Neste instante de verificação, e para evitar dessincronização entre os dados, os mutexes de todos os personagens são trancados e inicia-se o processo de verificar se será necessário esperar que alguém encerre o uso do forno primeiro. Durante este fluxo, uma vez que as variáveis de todos os personagens estão trancadas e isso impossibilita a mudança de estado da ação de outros personagens, a liberação das variáveis de todos os outros

personagens é realizada nos casos onde é necessário esperar alguém e isso acaba por mudar o instante verificado quando em loop. Por este motivo, no fim do loop é sempre verificado se o personagem realmente pode usar o forno.

Ainda sobre o tratamento das prioridades para o caso dos 6 personagens principais que possuem dependência circular (Sheldon, Amy, Leornard, Penny, Howard e Bernadette), foram definidos 1 (um) cenário-base e 5 (cinco) outros cenários possíveis sobre os quais a definição de prioridades se pautará:

1. *Cenário-base:* Personagem X está na fila, bem como um componente de cada um dos demais casais, e neste caso seguem-se as regras de precedência definidas no escopo do trabalho prático. Ou seja, inicia-se o processo de espera até que o personagem Y que detém prioridade sobre o Personagem X encerre a utilização do forno, ou até que o Raj intervenha em caso de deadlock;
2. *Cenário 1:* Personagem X está na fila e o outro Personagem Y com o qual compõe o casal não está. Entretanto, o Casal Z que detém prioridade sobre o Personagem X se encontra na fila, e neste caso faz-se necessário esperar pelo fim da utilização de ambos os personagens do Casal Z;
3. *Cenário 2:* Personagem X está na fila e o outro Personagem Y com o qual compõe o casal não está. Entretanto, o Casal Z que depende do Personagem X se encontra na fila, e neste caso, faz-se necessário esperar pelo fim da utilização de ambos os personagens do Casal Z;
4. *Cenário 3:* Personagem X está na fila, bem como o outro Personagem Y com o qual compõe o casal, e neste caso faz-se necessária a verificação da ordem de chegada entre os componentes do casal (Personagem X e Y) para definir qual irá esperar e qual prosseguirá com a utilização do forno. Além disso, optou-se por enviar broadcasts para destrancar todas as threads atualmente trancadas nas variáveis de condição específicas, ou seja, disparar sinais para liberar o Personagem Y caso ele esteja preso por achar que o seu par não se encontra na fila;
5. *Cenário 4:* Personagem X está na fila e o outro Personagem Y com o qual compõe o casal não está. Entretanto, tampouco estão o Casal Z dependente e o Casal W que detém prioridade sobre o Personagem X, e neste caso a utilização do forno é liberada ao Personagem X;
6. *Cenário 5:* Personagem X está na fila, bem como o outro Personagem Y com o qual compõe o casal, sendo que o personagem em questão. Entretanto, tampouco estão o

Casal Z dependente e o Casal W que detém prioridade sobre o Personagem X, e neste caso a utilização do forno é liberada ao Personagem X.

Independente de qual dos cenários acima se materialize, a intervenção do Raj para os casos de deadlock se mantém. A variável booleana de controle é trancada para garantir a sincronização dos dados ainda que sob múltiplas execuções das threads e posteriormente destrancada. Inicia-se então o processo de destrancar as variáveis dos demais personagens principais (Sheldon, Amy, Leornard, Penny, Howard e Bernadette) para que os mesmos sejam capazes de trancar suas próprias variáveis, antes sob domínio de um único personagem, e prosseguir com o fluxo de utilização do forno agora que o deadlock foi desfeito pelo Raj após a liberação de um personagem aleatório. Para estes casos, de intervenção do Raj em caso de deadlock, considerou-se ainda que no caso do Raj ter liberado o Personagem X e o Personagem Y com o qual o mesmo forma casal também se encontra na fila, então o Personagem Y também foi liberado. Neste caso, basta o personagem liberado prosseguir com a utilização do forno, para que em seguida o outro personagem do casal inicie a utilização.

No caso dos personagens Stuart e Kripke, que não apresentam dependência circular e iminência de deadlock entre si, foram definidos 2 (dois)| cenários possíveis sobre os quais a definição de prioridades se pautará:

1. *Cenário 6:* Personagem X é o Stuart e ele deverá aguardar até que os 6 personagens principais (Sheldon, Amy, Leornard, Penny, Howard e Bernadette) encerrem a utilização do forno no fluxo normal;
2. *Cenário 7:* Personagem X é o Kripke e, uma vez que ele deve ser sempre o último a usar o forno, o mesmo deverá aguardar até que os 6 personagens principais (Sheldon, Amy, Leornard, Penny, Howard e Bernadette) encerrem a utilização do forno no fluxo normal e ainda aguardar até que o Stuart encerre a utilização também;

Nos Cenários 6 e 7, independente do personagem em questão ser Stuart ou Kripke, a ideia se mantém a mesma. Serão esperados todos os personagens necessários, até que, em um instante específico, seja verificado que ninguém mais precisa ser esperado e, com isto, o forno será usado, caso esteja disponível.

Para os 7 (sete) cenários descritos acima a ideia principal é que, se todos os personagens liberarem uns aos outros antes de esperar pela utilização do personagem com a prioridade, eventualmente a pessoa que deve usar o forno vai ser capaz de trancar todas as

variáveis novamente e, no processo, verificar que não precisa esperar ninguém, por ser o personagem com a maior prioridade, e iniciar a utilização do forno (enviando o sinal e acionando a remoção de algum personagem da espera). Seguindo esta linha, considerou-se que as outras threads que conseguirem trancar as variáveis dos demais personagens irão ou liberar todas elas ou irão esperar por algum personagem. E, ao fim das possíveis esperas, o personagem deve verificar novamente se pode usar o forno (para o caso de algumas variáveis terem sido liberadas e terem mudado assim o instante verificado) e se não for autorizada a utilização o processo é repetido. Ou seja, o processo de verificação e espera continuará a se repetir a menos que, no instante em que o personagem conseguir trancar as variáveis dos demais personagens, for a vez dele.

Simulando a ação de manifestar a intenção de usar o forno, a função *wait* é a responsável por trancar/destrancar as variáveis do Personagem X, atualizar seu valor e em seguida definir se ele é o primeiro do casal a manifestar interesse no forno, para só então iniciar o processo de tratamento das prioridades descritas acima.

Simulando o fim da utilização e a liberação do forno, na função de *release,* a ideia principal é atualizar o valor da variável de manifestação de interesse no forno e sinalizar que outra pessoa pode usar o forno.

Neste sentido, consideramos o 3 (três) passos abaixo durante o fluxo:

1. *Passo 1:* Personagem X está na fila e o outro Personagem Y com o qual compõe o casal também está. Entretanto, um dos personagens encerrou a utilização do forno neste instante, e neste caso faz-se necessário manter o casal como formado até que o outro membro do casal use o forno;
2. *Passo 2:* No caso de ter ocorrido deadlock anteriormente, envolvendo o Personagem X e o outro Personagem Y com o qual compõe o casal além dos demais personagens, e este mesmo casal tenha sido escolhido aleatoriamente pelo Raj para ser liberado e prosseguir com a utilização do forno. Neste caso, faz-se necessário que os dois membros do casal usem o forno primeiro, começando pelo personagem liberado pelo Raj;

Simulando a ação checar a existência de deadlocks, exclusiva do Raj, a função *check* é a responsável por trancar/destrancar as variáveis dos 6 personagens principais (Sheldon, Amy, Leornard, Penny, Howard e Bernadette), trancar/destrancar as variáveis que representam quem do casal foi o primeiro a manifestar a intenção de usar o forno e também trancar/destrancar as variáveis que representam a formação de um casal.

Seguindo o fluxo de execução desta função, consideramos os 9 (nove) cenários possíveis para a ocorrência e detecção de deadlock:

1. *Cenário 1:* Todos os 3 (três) casais possíveis foram formados e nenhum deles iniciou a utilização do forno ainda.
2. *Cenário 2:* Os personagens Sheldon, Howard e Leonard se encontram na fila, e nenhum outro personagem chegou na fila para que os mesmos pudessem formar seus respectivos casais.
3. *Cenário 3:* Os personagens Sheldon, Bernadette e Leonard se encontram na fila, e nenhum outro personagem chegou na fila para que os mesmos pudessem formar seus respectivos casais.
4. *Cenário 4:* Os personagens Amy, Bernadette e Leonard se encontram na fila, e nenhum outro personagem chegou na fila para que os mesmos pudessem formar seus respectivos casais.
5. *Cenário 5:* Os personagens Amy, Howard e Leonard se encontram na fila, e nenhum outro personagem chegou na fila para que os mesmos pudessem formar seus respectivos casais.
6. *Cenário 6:* Os personagens Sheldon, Howard e Penny se encontram na fila, e nenhum outro personagem chegou na fila para que os mesmos pudessem formar seus respectivos casais.
7. *Cenário 7:* Os personagens Sheldon, Bernadette e Penny se encontram na fila, e nenhum outro personagem chegou na fila para que os mesmos pudessem formar seus respectivos casais.
8. *Cenário 8:* Os personagens Amy, Bernadette e Penny se encontram na fila, e nenhum outro personagem chegou na fila para que os mesmos pudessem formar seus respectivos casais.
9. *Cenário 9:* Os personagens Amy, Howard e Penny se encontram na fila, e nenhum outro personagem chegou na fila para que os mesmos pudessem formar seus respectivos casais.

Para definição dos Cenários 2 a 9 considerou-se que, como um dos casos de deadlock se dá justamente quando há uma dependência circular entre 3 personagens, a quantidade de cenários possíveis seria da ordem de 2^3 combinações. Ou seja, as combinações possíveis de serem realizadas considerando-se os 3 personagens dependentes circularmente como imposto

pela regra de deadlock e o fato de que a prioridade do Personagem X é equivalente a do Personagem Y quando os mesmos são um casal. Para tanto, não basta considerar as combinações em que um dos personagens do casal se encontra, é necessário considerar o outro personagem do casal também. Neste contexto, nos Cenários 2 a 9 o comportamento é semelhante, basta atualizar as variáveis *pessoas_deadlock* que representam as pessoas envolvidas na dependência circular e também a variável booleana *deadlock* que representa a ocorrência de um deadlock. Em caso de detecção de deadlock pelo Raj, um dos casais ou um personagem é escolhido aleatoriamente, a variável *raj_liberou* que representa essa ação é atualizada e dispara-se um broadcast para que a pessoa escolhida pare de esperar alguém e detecte que mesma está autorizada a prosseguir para a utilização do forno, por causa do Raj.

No fluxo da função *main*, optou-se por concentrar funções como as de criação das threads, variáveis de condição, variáveis mutexes, variáveis booleanas e de finalização das threads. Para tanto, a implementação das threads utilizou funções da biblioteca *pthread* como init, destroy e create. Além disso, foram implementadas mensagens de erro condicionadas a falhas na inicialização das variáveis mutexes, das variáveis de condição, bem como falhas na criação das threads também.

A partir da função *main* a função *execute_threads* é disparada, e para cada thread o fluxo a ser executado é semelhante. Ou seja, exceto pelo Raj, a sequência de passos segue como abaixo:

1. *Passo 1:* Utilizar a variável de forno para manifestar a intenção de utilizar o forno;
2. *Passo 2:* Utilizar a variável do personagem para utilizar o forno e cozinhar algo, e neste caso o tempo que a ação levará para ser concluída é equivalente a 1 segundo;
3. *Passo 3:* Utilizar a variável de forno para indicar o fim da utilização do forno;
4. *Passo 4:* Utilizar a variável do personagem para realizar a ação de comer, e neste caso o tempo que a ação levará para ser concluída é gerado com base no intervalo entre 1 e 2 segundos;
5. *Passo 5:* Utilizar a variável do personagem para realizar a ação de voltar ao trabalho, e neste caso o tempo que a ação levará para ser concluída é gerado com base no intervalo entre 3 e 6 segundos;

Neste contexto, para as funções que representam as ações disponíveis a estes personagens, como *cook_something, eat* e *work* a implementação é simples. Uma única ação

de sleep é disparada em cada uma delas para simular o tempo que uma ação demora para ser realizada e também para permitir que casos de deadlock sejam produzidos.

No caso do Raj, a partir da função *main e disparo da função execute_threads*, a sequência de passos segue como abaixo e é reproduzida enquanto os outros personagens não tiverem encerrado suas tarefas:

1. *Passo 1:* Utilizar a função *sleep* para indicar o processo de esperar a execução de outras thread, e neste caso o tempo que a ação levará para ser concluída é equivalente a 5 segundos. Ou seja, a cada 5 segundos e enquanto os outros personagens não tiverem encerrado suas tarefas, o Raj executa sua tarefa de checar ocorrência de deadlock;
2. *Passo 2:* Utilizar a variável de forno para realizar efetivamente sua tarefa de checar a ocorrência de deadlock, considerando os cenários possíveis descritos anteriormente.

Decisões de Projeto

Dada as estratégias apresentadas acima, as seguintes decisões de projeto foram consideradas:

1. Utilização do *pthread_cond_broadcast* para garantir que todas as threads esperando possam receber o sinal;
2. Utilização sempre da mesma ordem de trancamento das variáveis dos personagens para garantir que não serão gerados deadlocks no processo. Ou seja, casos em que um personagem tranca a variável de outro e a recíproca se concretiza, causando dependência circular e espera infinita.
3. Para utilização da função wait ligada às variáveis de condição definidas, utilizou-se de loop while para conferir se as mudanças esperadas aconteceram, e assim evitar um fenômeno chamado "spurious wakeups" que advém da implementação da própria função wait.
4. Ainda que a implementação da função cout fornecida para C++ seja thread safe, ou seja, apresenta comportamento esperado na maioria dos casos quando executada por múltiplas threads simultaneamente, optou-se por definir ainda mutexes para controlar o acesso às saídas apresentadas por meio do cout. Tal decisão se deu depois da constatação de que a função não apresentava o comportamento esperado em algumas execuções.

5. Considerou-se que quando um personagem X está no processo de esquentar algo a mesma ainda está na fila (querendo usar), uma vez que a mesma ainda não concluiu a utilização do forno.

Bugs Conhecidos

Ao final da implementação e subsequentes testes realizados, não foram identificados bugs.

Execução do Trabalho Prático

Para simular a utilização do forno pelos personagens de The Big Bang Theory a partir do código implementado, utilize:

1. *make clean*
2. *make run*
3. *./main [número de repetições]* (e.g: *./main 10*)