Trabalho Prático I - Pthreads

Sistemas Operacionais

Gabriel Victor Carvalho Rocha

gabrielcarvalho@dcc.ufmg.br - 2018054907

1. Introdução

O projeto se resume ao controle do compartilhamento de um forno entre oito personagens, sendo que seis deles formam casais e dois são solteiros, possuindo prioridades diferentes e podendo gerar um ciclo (deadlock).

Para isso, temos que utilizar um mutex para gerenciar o uso desse forno, aliado a uma condição para a liberação do forno quando um personagem acabar de utilizá-lo.

2. Implementação

A implementação consiste na manipulação de um vetor onde cada índice corresponde ao id do personagem. Mapeando 0 para Sheldon, 1 para Leonard, 2 para Howard, 3 para Stuart, 4 para Kripke, 5 para Amy, 6 para Penny e 7 para Bernadette, onde os casais são os pares 0-5, 1-6 e 2-7. Como as namoradas possuem a mesma prioridade que seus namorados, podemos encontrar a prioridade delas utilizando mod 5 através de um mapa onde as chaves correspondem aos id's mod 5 e o valor é um set dos id's nos quais possui prioridade.

O vetor é iniciado com 0, significando que o personagem não está esperando. Entrando na thread há um sleep de espera que dura entre 1 e 6 segundos, em seguida temos um mutex e uma condição que atua no gerenciamento da inserção no vetor e no caso onde há deadlock. Então, caso o personagem não possua parceiro ou seu parceiro não esteja na fila, é colocado o valor 1 na vetor[id], porém, se o parceiro estiver na fila, é colocado o valor vetor[parceiro(id)] + 1 e é reiniciada a variável global prox_casal, na qual é utilizada para armazenar o id do parceiro que chegou por último quando há formação de casal na fila, sendo este o próximo após ele terminar.

O mutex então bloqueia o uso do forno e os personagens irão esperar sua vez através do while que checa se o seu id é o próximo por meio de uma variável global prox, porém o primeiro a entrar na thread irá utilizar o forno através da flag FIRST. Quando um personagem vai utilizar o forno, é colocado o valor 0 na vetor[id] correspondente ao seu id, ou seja, ele sai da fila e é chamada a função para esquentar algo. Em seguida, chamamos a função para encontrar o próximo personagem de maior prioridade na fila.

A função encontra_proximo inicia checando se prox_casal já possui quem deve ser o próximo, caso positivo, apenas colocamos o personagem em prox e prox_casal é reiniciada. Porém, se prox_casal não possuir um id (EMPTY), então é analisado se há um deadlock, que é feito checando se há apenas um de cada casal utilizando XOR, ou se há todos os três casais utilizando AND, se houver deadlock é colocada a flag DEADLOCK na variável prox para que o Raj consiga tratar, sendo assim bloqueado o mutex para inserção na fila. Caso contrário, teremos dois casos: Se houver algum casal, irá checar entre os três possíveis casais, qual possui a maior prioridade. Como somamos 1 no último parceiro a entrar na fila, o próximo é o que tiver menor valor, guardando o seu parceiro em prox_casal; Se não houver casal, então basta checar todos os que estão esperando na fila e colocar como próximo o que possuir maior prioridade. Em seguida, o forno é desbloqueado e é enviado um sinal através da função pthread_cond_broadcast para liberar quem tiver maior prioridade.

Por fim, o personagem irá comer e sua thread dorme de 3 a 6 segundos, após isso, o personagem vai trabalhar e sua thread dorme de 3 a 6 segundos novamente.

Raj fica verificando se a variável prox é igual a DEADLOCK, que como explicado anteriormente, significa que foi achado um deadlock. Quando isso ocorre, a função trata_deadlock é chamada e tratará no primeiro if quando houver casais, escolhendo aleatoriamente o próximo e o seu antecessor (prox_casal). Caso não haja casais, apenas é escolhido um personagem aleatoriamente e se ele não estiver na fila, é escolhido o seu parceiro, removendo o deadlock, desbloqueando a fila e enviando um broadcast para as condições da fila e do forno.

3. Instruções

Para compilar, basta digitar o comando “make”, gerando o executável “main” que pode ser executado da seguinte forma:

./main {num_vezes}, onde num_vezes é o parâmetro de quantas vezes cada personagem irá usar o forno.

4. Conclusão

Dessa forma, o trabalho foi uma experiência de muito aprendizado, pois foi possível entender o funcionamento de multithreads. Acredito que foram abrangidos todos casos, porém não compreendi onde e como utilizar a condição do casal sugerida no enunciado do trabalho, então ela não foi utilizada. Houveram outras dificuldades, como: A falta de deadlocks ocorrendo nas execuções, onde foi necessário forçar alguns casos para que ocorressem de fato e checasse se estavam funcionando, porém mesmo forçando, ocorriam muito raramente; Foi necessário colocar um mutex e uma condição quando se entra na espera e imprime “{nome(id)} quer usar o forno” que também é controlada pela função que trata deadlock, pois algumas impressões ocorriam em ordem errada ou entravam mais personagens enquanto um deadlock estava sendo tratado .

Tomei algumas decisões no que tange especificações que não ficaram claras no enunciado do trabalho, como:

- Quando um personagem começa a usar o forno, ele sai da fila. Ou seja, se Sheldon estiver usando o forno e chegar Penny e Amy, Penny tem prioridade, pois Sheldon já saiu da fila.
- Se houver Howard > Bernadette > Penny > Leonard e se Howard sair para esquentar, ficando Bernadette > Penny > Leonard e se Amy e Sheldon entram na fila antes que Bernadette use o forno, a fila ficará:

  Penny > Leonard > Amy > Sheldon > Bernadette.

- Se houver Sheldon > Amy e se Sheldon sair para esquentar, ficando apenas Amy, caso Penny entre na fila antes que Amy use o forno, a fila ficará:

  Amy > Penny