UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
DEPARTAMENTO DE INFORMÁTICA APLICADA

INF01151 – SISTEMAS OPERACIONAIS II N

TRABALHO 2: JANTAR DOS FILÓSOFOS
SEMÁFOROS E VARIÁVEIS DE CONDIÇÃO

INTEGRANTES:
BRUNO GABRIEL FERNANDES
MARCELO DE OLIVEIRA ROSA PRATES
SUYÁ PEREIRA CASTILHOS

1. Ambiente de teste

Como ambiente de Teste, foi utilizado uma máquina virtual com auxílio do Oracle VM VirtualBox, com 4 cores e 512 mb de memória, executando o Sistema Operacional Ubuntu 11.04, tudo isso em cima de um Hospedeiro Intel I3 de 2 cores físicos e 2 cores virtuais com 4 gb de memória.

Como compilador, foi utilizado o GCC na versão 4.5.2.

2. Versões: Semáforos e Variáveis de Condição

Foram desenvolvidas duas versões destinadas à resolução independente do mesmo problema: o Problema do Jantar dos Filósofos. Neste problema, N filósofos se dispõem numa mesa circular de N lugares e com N garfos. Os filósofos passam o tempo pensando e comendo. Para comerem, é preciso que os filósofos portem dois garfos: um na mão esquerda e um na mão direita. Algum tempo após comerem, os filósofos voltam ao estado “pensando”. Alternativamente, após algum tempo pensando, os filósofos desejam passar para o estado “comendo”, mas para isso dependem de que ambos os garfos (esquerdo e direito) estejam disponíveis. O problema levanta questões sobre exclusão mútua e inanição.

2.1. Versão Semáforos

Na versão implementada usando semáforos, cada filósofo é implementado como uma thread. Adicionalmente, cada garfo é implementado como um semáforo de valor inicial 1. Para impedir que dois filósofos obtenham o mesmo garfo, é feito “wait” do semáforo correspondente ao garfo.

Os filósofos, no momento do chaveamento para a sua thread, se comportam da seguinte maneira:

- Estando anteriormente no estado inicial 'I', o filósofo passa para o estado 'T' (thinking, pensando) e a thread correspondente dorme por uma quantia randômica de tempo situada entre 1 e 10 segundos.

- Estando anteriormente no estado 'T' (thinking), o filósofo passa para o estado 'H' (hungry, esfomeado) e tenta pegar os garfos direito e esquerdo (nessa ordem, se for o filósofo de índice 0, e na ordem inversa, esquerdo e direito, caso contrário). A ação de tentar pegar um garfo se traduz, em termos de programação, na chamada da primitiva sem_wait() com o semáforo correspondente ao garfo desejado sendo passado como parâmetro.

- Estando anteriormente no estado 'H', o filósofo passa para o estado 'E' (eating, comendo) e dorme por uma quantia randômica de tempo situada entre 1 e 10 segundos.

- Estando anteriormente no estado 'E', o filósofo passa para o estado 'T' e libera ambos os garfos. A ação de liberar um garfo se traduz, em termos de programação, na chamada da primitiva sem_post() com o semáforo correspondente ao garfo liberado sendo passado como parâmetro.

Toda vez que um filósofo troca de estado, é impresso na tela uma linha que representa o estado de todos os filósofos na mesa, do tipo:

E H T T E H

Como é uma mesa circular, devemos nos lembrar que o último filosofo está sentado diretamente à esquerda do primeiro.

Para evitar que a impressão de diversas linhas seja intercalada, e mais importante, para evitar a impressão de linhas com estados inconsistentes, a ação de trocar de estado de um filósofo e sua consecutiva impressão da linha de estados é encapsulada em uma seção crítica que impede que dois filósofos troquem de estado e imprimam a linha ao mesmo tempo.

Se fosse permitido que os filósofos trocassem de estado fora da seção crítica, poderiam ocorrer erros de impressão de linhas inconsistentes, por exemplo:

1- O filósofo A acabou de trocar de estado (qual estado é irrelevante) e começa a impressão da linha de estados, e imprime o estado do filósofo B, que está comendo (“E”).
2 - Logo após a impressão do estado do filósofo B, ocorre uma preempção do filósofo B, avisando que terminou de comer, e passou para o estado “T”, liberando os garfos.
3 - Logo após isso, o filósofo C, que estava esfomeado (“H”) pega os garfos e passa para o

estado comendo (“E”).
4 - Voltando para a impressão do filósofo A, ele imprime o estado do próximo filosofo, C, que está comendo (“E”).
5- Assim, apesar de os dois filósofos B e C terem comido em tempos diferentes, eles apareceriam na linha de estados como dois E’s, um estado inconsistente.

Para evitar o problema de Deadlock que ocorre quando todos os filósofos obtem o garfo da esquerda ao mesmo tempo, foi adicionado o “filósofo idiossincrático” (de índice 0) que, no caminho oposto ao de seus colegas, insiste em pegar os garfos na ordem trocada (direita,esquerda) ao invés de (esquerda,direita). Dessa forma, esta situação de Deadlock é resolvida.

Essa modelagem resolve o problema evitando os imprevistos de violação de exclusão mútua e de deadlock, pois garante exclusão mútua na manipulação dos garfos por meio dos semáforos e evita o deadlock por meio da adição do

2.2. Versão Variáveis de Condição

A versão de variáveis de condição carrega a mesma lógica da versão de semáforos para representação dos filósofos, mudança de estados, prevenção de deadlocks, etc. O que diferencia ambas as versões (semáforos e variáveis de condição) é a representação dos garfos: na primeira versão, cada garfo está associado a um semáforo binário. Na segunda versão, alternativamente, cada garfo está associado ao mesmo tempo a um estado ‘S’ para “ocupado” e ‘N’ para “livre” determinando se o garfo está ou não disponível e a uma variável de condição (*pthread_cond_t*). Adicionalmente, foi criado um *mutex* (*pthread_mutex_t*) adicional responsável por assegurar exclusão mútua sobre as porções de código que lidam com as variáveis de condição.

Antes de comer, cada filósofo chama, em laço, a primitiva *pthread_cond_wait* sobre a variável de condição dos seus garfos esquerdo e direito enquanto o estado de cada um desses garfos não for ‘N’ (livre). Essa porção de código é protegida por um *mutex*. Além disso, depois de comer, cada filósofo chama a primitivia *pthread_cond_signal* sobre a variável de condição dos seus garfos esquerdo e direito. Essa porção de código também é protegida por um *mutex*.

3. Conclusões

A eliminação de deadlocks segue o mesmo princípio tanto na versão semáforos quanto na versão variáveis de condição: o procedimento dos filósofos ao tentarem adquirir os garfos é independente do mecanismo de sincronização utilizado. Isso indica que a dicotomia semáforos vs variáveis de condição, neste caso, levanta questões apenas sobre a sintaxe de cada mecanismo. Sobre isso, algumas considerações podem ser feitas: no caso do problema dos filósofos, a tendência é de que as duas versões compartilhem muitas semelhanças, pois cada garfo pode ser utilizado por apenas um filósofo simultaneamente. Ou seja, podemos resolver o

problema tanto com semáforos binários quanto com variáveis de condição. Fosse o caso em que a sincronização dependesse de lógicas mais complexas do que a exclusão mútua sobre um recurso (garfo), uma possibilidade é que as variáveis de condição, com a sua semântica mais genérica, fossem o mecanismo preferido para a implementação da solução.