Robson de Arruda Silva – 2019013624

Sistemas Operacionais – 29/09/2020

No código 15:

O código funcionou de início.

O que pude perceber no código:

O acesso ao recurso compartilhado só pode ser realizado por dois processos e sempre de maneira alternada. Com isso, um processo que necessite utilizar o recurso mais vezes do que outro permanecerá grande parte do tempo bloqueado. Caso um processo permaneça muito tempo na função de processamento, é possível que outro processo queira executar sua região crítica e não consiga, mesmo que outro processo não esteja utilizando o recurso. Outro problema existente é que, no caso da ocorrência de algum problema com um dos processos, de forma que a variável de bloqueio não seja alterada, o outro processo permanecerá indefinidamente bloqueado, aguardando o acesso ao recurso.

No código 16:

O código funcionou e não teve uma parada.

O que pude perceber no código:

O uso do recurso nesse algoritmo não é realizado necessariamente alternado. Caso ocorra algum problema fora da região crítica, o outro processo não ficará bloqueado, o que, porém, não resolve por completo o problema. Caso um processo tenha um problema dentro da sua região crítica ou antes de alterar a variável, o outro processo permanecerá indefinidamente bloqueado. Mais grave que o problema do bloqueio é que esta solução, na prática, é pior do que o primeiro algoritmo apresentado, pois nem sempre a exclusão mútua é garantida quando cada processo executa cada instrução alternadamente.

No código 17:

O código funcionou de início.

O que pude perceber no código:

O código introduz um novo problema, que é a possibilidade de bloqueio indefinido de ambos os processos. Caso os dois processos alterem as variáveis identificadoras antes da execução da instrução WHILE, ambos os processos poderão entrar em suas regiões críticas, como se o recurso já estivesse alocado.

No código 18:

O código funcionou de início.

O que pude perceber no código:

Nesse algoritmo há um porém, se no caso de os tempos aleatórios serem próximos e a concorrência gerar uma situação onde nenhuns dos dois processos alterem as variáveis identificadoras para 0 (false) antes do término do loop, nenhum dos dois conseguirá executar sua região crítica. Mesmo com esta situação não sendo permanente, pode gerar algum problema.

No código 19:

O código funcionou, mas após um tempo ocorreu um erro.

O que pude perceber no código:

O código permitiu que dois segmentos compartilhem um recurso de uso único sem conflitos, usando apenas memória compartilhada para comunicação. Se dois processos tentarem entrar em uma seção crítica ao mesmo tempo, o algoritmo permitirá apenas um processo, com base em qual é a sua vez. Se um processo já estiver na seção crítica, o outro processo aguardará a saída do primeiro processo.

No código 27:

O código funcionou e teve um problema.

O que pude perceber no código:

Ocorreram problemas, pois os processos compartilham um buffer de tamanho fixo e o processo produtor coloca dados no buffer e o processo consumidor retira dados do buffer, sendo assim, o produtor coloca dados quando o buffer ainda está cheio e também o consumidor retira dados quando o buffer está vazio, dando um problema.

No código 28:

O código funcionou e teve um travamento no final.

O que pude perceber no código:

Enquanto um filósofo estiver comendo ou pensando, ele permanecerá travado em um número aleatório de segundos entre 0 e 10. Ou seja, em algum momento o programa irá cair na condição de deadlock, podendo ficar assim travados para sempre.

No código 29:

O código funcionou e não ficou parado.

O que pude perceber no código:

Nesse novo código, o filósofo poderá nunca comer. Ele não ficará travado, mas irá ficar tentando fazer a mesma operação de pegar e depois devolver o garfo pra sempre. A consequência será nunca conseguir comer.

No código 30:

O código funcionou de início.

O que pude perceber no código:

No código há uma variável barbeiro, uma variável cadeira de barbeiro e n cadeiras para eventuais clientes esperarem a vez. Quando não há clientes que vá para a função, a variável barbeiro relaciona-se com a cadeira de barbeiro fica parado. Quando chega um cliente na função, ele precisa dar um toque no barbeiro. Se outros clientes chegarem enquanto o barbeiro estiver fazendo o cabelo de um cliente, eles se sentarão (caso houver) ou sairão da barbearia (caso todas cadeiras tiverem indisponíveis). Sendo assim, o código conteve uma função na qual, faz que, quando um cliente que entra na barbearia conte o número de clientes à espera de atendimento. Se este for menor que o número de cadeiras, ele ficará; do contrário, sairá, e qualquer outro cliente (e até o barbeiro) deve esperar a liberação de mutex.