DinnerRace: Um simulador para técnicas de exclusão mútua através do problema do Jantar dos Filósofos

Amanda Barbosa[1], **Witássio Oliveira**[1], **Kádna Camboim**[1]

[1]Unidade Acadêmica de Garanhuns, Universidade Federal Rural de Pernambuco (UFRPE)
Av. Bom Pastor, s/n – Boa Vista – 55.292-270 – Garanhuns – PE, Brasil

amandaoliveirabr@gmail.com, witassio@gmail.com, kadna@uag.ufrpe.br

Resumo. *Este artigo discute sobre o uso e aplicação do DinnerRace, um software que simula a aplicação de técnicas de exclusão mútua sobre o Problema Jantar dos Filósofos, como auxiliar no ensino e aprendizagem de conceitos na disciplina de Sistemas Operacionais em cursos de computação.*

Abstract. *This paper discuss the development and application of the DinnerRace, a simulator that applies some methods for mutual exclusion on the Dining Philosophers problem, and how to use it as a tool for visual aid in teaching the mutual exclusion concepts on a Operacional Systems course.*

1. Introdução

O estudo de Sistemas Operacionais (SO) é fundamental em cursos de Computação e Informática. No estudo da disciplina são introduzidos conceitos como gerência de memória, sistema de arquivos, gerência de processos, etc, que são importantes à compreensão do funcionamento e integração entre *software* e *hardware*. Os conceitos abordados, têm, em sua maioria, um alto nível de abstração e complexidade e nem sempre os alunos conseguem obter um entendimento completo e um bom desempenho na disciplina. Segundo Machado e Maia (2004), o problema está tanto no modelo de ensino como na falta de ferramentas capazes de traduzir para a realidade os conceitos teóricos apresentados. Geralmente, as aulas sobre Sistemas Operacionais são teóricas, com poucos exemplos práticos, tornando o aprendizado mais difícil.

Um dos principais conceitos abordados na disciplina é o de exclusão mútua, que está relacionado à concorrência de processos. Para a área de projeto de processadores, esse é um conceito chave em sistemas operacionais, mas, a abstração tem impedido muitos estudantes de compreenderem e aplicarem os conceitos aprendidos. Muitas abordagens foram e vem sendo estudadas a fim de tornar o estudo mais dinâmico, como a construção de laboratórios voltados para a disciplina, o uso prático do kernel estruturado de Linux a fim de trazer conceitos básicos para o estudo da disciplina. Mas a dificuldade de construir e manter esses laboratórios, bem como de familiarizar os estudantes ao uso de Linux, torna essas abordagens, muitas vezes, inviáveis.

Este trabalho aborda o uso de um *software* que simula várias soluções de exclusão mútua, sobre o problema Jantar dos Filósofos, que foi desenvolvido como ferramenta auxiliar para o ensino dos conceitos relacionados à disciplina de Sistemas Operacionais.

2. Jantar dos Filósofos

O problema do Jantar dos Filósofos é frequentemente usado em programação concorrente para demonstrar conceitos de sincronização. Inicialmente, o problema foi proposto por E. W. Dijkstra, um renomado cientista da computação alemão, em 1965. A ideia do jantar dos filósofos pode ser representada como na Figura 1, onde há cinco filósofos prontos para jantar em uma mesa que contém cinco pratos e cinco garfos. Cada filósofo pode comer, pensar ou dormir. Porém, para um filósofo comer são necessários dois garfos (direita e esquerda). Sempre que um filósofo terminar de comer, ele deve colocar os garfos de volta na mesa. O compartilhamento do recurso (garfo) limita a quantidade de filósofos que podem comer ao mesmo tempo, e gerenciar esse recurso é o real problema.

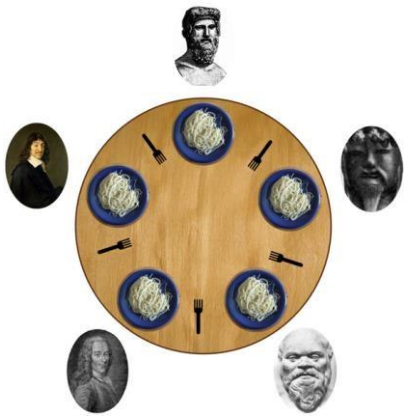

Figura 1: Ilustração do Problema Jantar dos Filósofos

3. Exclusão mútua

No estudo dos Sistemas Operacionais, no que diz respeito à comunicação entre processos, há a comunicação por mensagens e a comunicação através de recursos compartilhados. Essa última pode ser interpretada de maneira similar ao Problema do Jantar dos filósofos. A solução para este tipo de comunicação é a exclusão mútua.

Frequentemente, processos (programas em execução) compartilham dados da memória - variáveis ou arquivos. Essa área compartilhada é chamada de região crítica. É recomendado evitar o acesso simultâneo aos dados, pois, isso pode gerar inconsistência e até erros de sistema.

Exclusão mútua é o que garante que, quando um processo está fazendo uso de sua região crítica, nenhum outro processo poderá fazer uso dessa região. Para garantir a exclusão mútua é preciso respeitar as seguintes premissas (TANENBAUM, 2003): 1) Nenhum dos dois processos pode estar simultaneamente dentro da sua região crítica; 2) Nenhuma suposição pode ser feita sobre as velocidades ou sobre o número de CPUs; 3) Nenhum processo que executa fora de sua região crítica pode bloquear outro processo; 4) Nenhum processo deve ter de esperar eternamente para entrar em sua região crítica.

A fim de garantir a exclusão mútua no problema do Jantar dos Filósofos, o simulador desenvolvido para a realização deste trabalho aborda os quatro algoritmos a seguir: desativação das interrupções, alternância estrita, variável de bloqueio e *sleep and wait.*

3.1. Desativação das interrupções

Cada processo desativa as interrupções imediatamente depois de entrar em sua região crítica. As interrupções são reativadas imediatamente após o processo deixar a região crítica. Com essa solução, a CPU fica impedida de alternar entre os processos para realizar a multiprogramação. Em geral, a CPU só alterna de um processo para outro como resultado de interrupções, como por exemplo, interrupção de relógio. Com a incapacidade de alternar entre os processos, o processo que entrou na região crítica poderá examinar e atualizar a memória compartilhada com a certeza de que nenhum outro processo irá interferir. Essa técnica, apesar de simples traz alguns problemas, pois, não é aconselhável permitir que processos de usuário desativem instruções e não permitam a execução de interrupções. Isso vai de encontro a segurança dos SOs, dado que uma ação como esta poderá levar a uma situação extrema, em que as instruções desativadas nunca virem a ser executadas (SILBERSCHATZ, 2001).

3.2. Variável de Bloqueio

Essa técnica define uma variável de bloqueio compartilhada, que poderá assumir os valores zero (0) e um (1). Inicialmente, essa variável recebe o valor 0. Se um processo entra na sua região crítica, essa variável compartilhada receberá o valor 1.

3.3. Alternância estrita

A solução baseada em alternância estrita, assim como a variável de bloqueio, também utiliza uma variável compartilhada, denominada *turn,* que pode assumir os valores zero (0) e um (1). Essa variável monitora de qual processo é a vez de entrar na região crítica. Para *turn* igual a 0, o primeiro processo terá a vez de entrar na região crítica. O segundo processo testa a variável *turn* continuamente, até que ela seja 1. Quando *turn* igual a 1, o processo entrará na região crítica. Essa solução acarreta numa situação que desperdiça tempo de CPU, a qual é conhecida como espera ativa já que a variável é testada continuamente, até que um determinado valor apareça.

3.4. *Sleep and Wake up*

A primitiva *sleep* causa o bloqueio do processo que fez a chamada, suspendendo-o até que o outro processo o acorde. A primitiva *wakeup* é responsável por desbloquear o processo. Esta solução evita o esperdício de tempo da CPU.

4. Software DinnerRace

O *software DinnerRace* implementa as quatro propostas (acima citadas) para exclusão mútua. A tela inicial do *software DinnerRace* é apresentada na Figura 2. Como pode ser observado, apenas a escolha do algoritmo deve ser feita. Após isso, inicia-se a simulação que dura por um tempo aleatório. O *software* termina de simular cada um dos algoritmos quando todos os filósofos passam por todos os estados.

Após a escolha do algoritmo, o *software* simula o jantar dos filósofos, o que implica em um determinado tipo de execução para cada algoritmo. É exibido na tela o status atual de cada filósofo de acordo com o algoritmo escolhido, sendo os status: Comendo, Faminto e Pensando. A Figura 3 apresenta o *DinnerRace* executando o algoritmo de

alternância estrita. Vale ressaltar, que o exemplo dado é apenas uma parte da simulação, pois, se for observada a barra de rolagem da figura 3, parte da execução não está visível.

O simulador foi desenvolvido utilizando a linguagem Java, cuja escolha se deu pelo fato de ela ser portável – já que geralmente os computadores dos estudantes possuem variadas configurações de hardware e software – bem como, melhor domínio por parte dos desenvolvedores. Com seu uso, os estudantes podem constatar o efeito dos diferentes algoritmos sobre o problema, enfatizando que o *DinnerRace* gerencia os recursos compartilhados, garantindo que não haverá conflitos entre os processos na busca pelo recurso. Caso um processo esteja utilizando o recurso, o processo que chegou por último deve esperar até que o recurso esteja disponível para o seu uso. Assim, pode-se dizer que houve exclusão mútua.

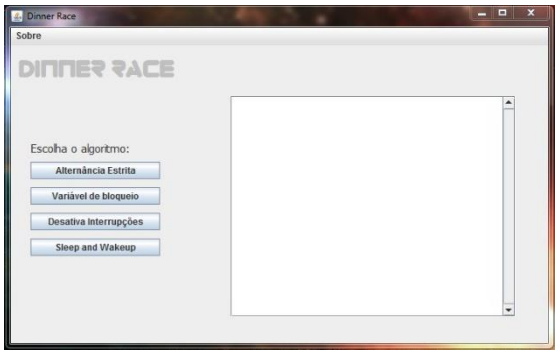

Figura 2. Tela inicial do DinnerRace

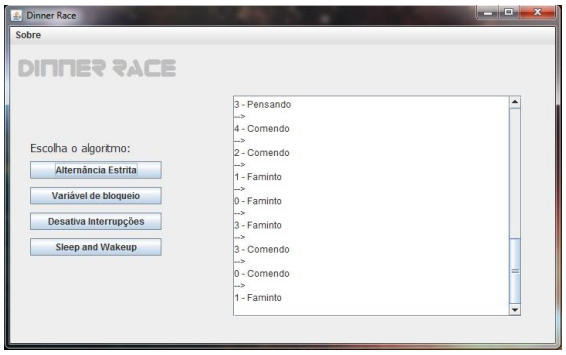

Figura 3. Resultado da execução da simulação com Alternância Estrita

Como propostas para trabalhos futuros, pretende-se implementar os algoritmos de semáforos, monitores e instrução TSL para exclusão mútua do Jantar dos Filósofos. Além disso, pretende-se aprimorar a interface gráfica que permitirá a visualização dos filósofos ao levantar os garfos e pô-los novamente na mesa, enfatizando quais filósofos sentem fome, pensam ou comem. E ainda, serão realizados estudo de casos com alunos da disciplina de SO para corrigir eventuais problemas.

Referências

MACHADO, F.B., MAIA, L.P. Um Framework Construtivista no Aprendizado de Sistemas Operacionais - Uma Proposta Pedagógica com o uso do Simulador SOsim. XII Workshop de Educação em Computação (WEI), XXIV Congresso da Sociedade Brasileira de Computação (SBC), Salvador, BA, ago. 2004.

LAADAN, O., NIEH, L., VIENNOT, N., Teaching Operating Systems Using Virtual

Appliances and Distributed Version Control. 41st ACM Technical Symposium on Computer Science Education (SIGCSE 2010), March 2010

SILBERSCHATZ, Abraham; GALVIN, Peter Baer; GAGNE Greg. Conceitos de

SistemaOperacional. Sexta Edição. Elsevier, 2001.

TANENBAUM, Andrew. S. Sistemas Operacionais Modernos. LTC, Segunda Edição, São Paulo: Prentice-Hall, 2003. (Cap. 2).