COBENGE 2016
XLIV CONGRESSO BRASILEIRO DE EDUCAÇÃO EM ENGENHARIA
27 a 30 de setembro de 2016
UFRN / ABENGE

SIMULADOR WPR QUE DETECTA DEADLOCK PARA AUXILIAR NO APRENDIZADO DE ESTUDANTES DE ENGENHARIA: UMA ABORDAGEM DA DISCIPLINA DE SISTEMAS OPERACIONAIS

Wellington D. de Almeida – wellingtondantas39@gmail.com
Instituto Federal de Educação, Ciência e Tecnologia do Ceará
Av. 13 de Maio, N°. 2081
60040-531 – Fortaleza - Ceará
Polycarpo S. Neto – policarpo7.fb@gmail.com
Rebeca M. Silva – rebeca.matos94@gmail.com
Francisco J. A. de Aquino – fcoalves_aq@ifce.edu.br
Fernando P. Garcia – fernandoparente@ifce.edu.br

Resumo: *Este trabalho tem como abordagem apresentar uma ferramenta didática para auxiliar no processo de aprendizagem dos estudantes de engenharia, em particular da disciplina de Sistemas Operacionais, nos cursos de Engenharia da Computação e Engenharia de Telecomunicações do Instituto Federal do Ceará (IFCE). Nessa disciplina foi proposto o desenvolvimento de um programa que detectasse a ocorrência de deadlock entre processos, denominado WPR. O WPR foi desenvolvido durante o decorrer da disciplina, e teve como principal objetivo incentivar os estudantes a terem uma visão prática do funcionamento de um sistema operacional. É apresentado nesse trabalho um estudo introdutório de sistemas operacionais modernos, fundamentação teórica sobre os conceitos de deadlock, metodologias utilizadas para o desenvolvimento do programa, a aplicação do software e os resultados visíveis com a interface do programa.*

Palavras-chave: *Ensino de engenharia, Aprendizado, Deadlock, Sistemas operacionais.*

1. INTRODUÇÃO

Os sistemas de computadores estão evoluindo a cada dia, principalmente por que os usuários estão exigindo sistemas mais sofisticados e robustos, que tenham mais funções e recursos para auxiliarem nas atividades do cotidiano. Os novos sistemas operacionais (SO) devem ser capazes de trazer velocidade e agilidade na realização de tarefas, assim como, devem evitar *bugs* de funcionamento e *deadlock* entre processos.

Ensinar aos estudantes sobre a demanda computacional moderna, apenas pela abordagem conceitual, pode não os levar ao nível ideal de entendimento, principalmente pela ausência de uma abordagem prática. Sabendo disso, trabalhos de implementação e simulação ajudam os estudantes a terem um pensamento mais crítico e menos acomodativo, adquirindo a compreensão necessária sobre fenômenos e resolução de problemas propostos (GUEDES, 2014).

O foco da disciplina de Sistemas Operacionais é ensinar aos alunos, as funções e estruturas de um sistema operacional, apresentando conceitos, entre eles os de processos e threads. O programa desenvolvido ajuda a simular situações que podem ser despercebidas observando somente os algoritmos e conceitos, ampliando assim o entendimento organizacional sobre um sistema de computador e fazendo que a referida disciplina não seja somente expositiva.

2. FUNDAMENTAÇÃO TEÓRICA

Em sistemas operacionais, um *deadlock* ocorre quando há uma situação em que dois ou mais processos que partilham do mesmo recurso, impedem o acesso do recurso pelo processo concorrente, resultando no não funcionamento de ambos os programas. Os sistemas de computadores têm inúmeros recursos adequados ao uso de somente um processo por vez (TANENBAUM, 2010). Uma situação onde isso acontece é quando o processo A está de posse do recurso impressora e solicita o recurso scanner, enquanto o processo B está de posse do recurso scanner e solicita o recurso impressora. Nesse caso, ambos os processos permanecerão bloqueados para sempre. Essa situação é chamada de *deadlock*.

A tentativa de dois processos usarem a mesma entrada da tabela do sistema de arquivos, inevitavelmente levará ao corrompimento do sistema de arquivos (TANENBAUM, 2010). Idealmente, o sistema operacional deve evitar que dois ou mais processos venham a interferir em ações do outro, principalmente, quando compartilham a mesma base de dados.

2.1. Tipos de recursos

O recurso pode ser um dispositivo de *hardware* (como uma impressora, por exemplo) ou um fragmento de informação (como um registro bloqueado na base de dados, por exemplo). No geral, um computador tem vários recursos que podem ser obtidos. Dizemos ser um recurso, algo que possa ser utilizada por apenas um único processo em qualquer instante de tempo (TANENBAUM, 2010).

Os recursos podem ser de dois tipos: o preemptível e o não preemptível. Um recurso é dito preemptível, quando pode ser tirado de um processo proprietário e entregue a outro, sem efeitos danosos, um exemplo é a memória. Em contraste, um recurso não preemptível é aquele que não pode ser tirado de seu atual dono, sem causar falha, um exemplo é a impressora.

Em geral, impasses envolvem recursos não preemptíveis (TANENBAUM, 2010), isso porque utilizando o exemplo da impressora, o recurso não pode ser retirado de um processo e repentinamente entregue a outro processo, sem causar prejuízo ao sistema. Saindo dessa ideia, há um caso onde a CPU pode ser compartilhada entre processos, saindo de um estado e quando o processo é retomado, aproveita esse estado antigo, seria o de chaveamento de contexto, não abordado neste trabalho.

2.2. Condições para ocorrência de deadlock

Há quatro condições que (SILVA & LIMA, 2015) identificou que são necessárias para alcançar um *deadlock*:

- ***condição de exclusão mútua:*** cada recurso ou é atualmente atribuída a exatamente a um processo proprietário ou está disponível;

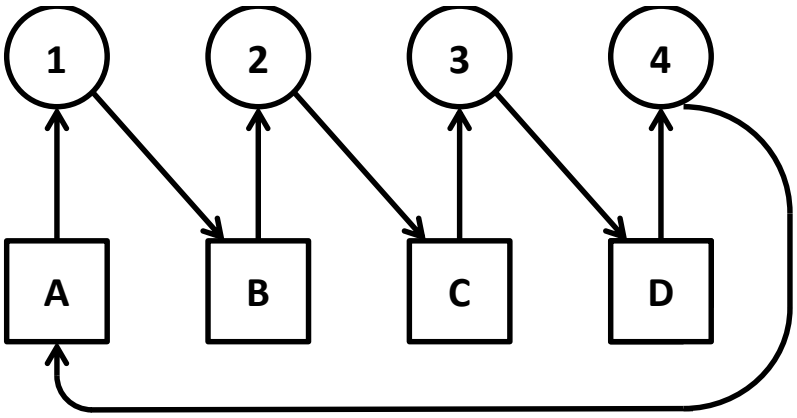
- ***condição de posse e espera*:** processos que atualmente detêm recursos concedidos anteriormente podem solicitar novos recursos;
- ***condição de não preempção*:** recursos anteriormente concedidos a um processo, não podem ser tirados à força. Os recursos devem esperar ser liberados voluntariamente pelo processo que o mantém;
- ***condição de espera circular:*** deve haver uma cadeira circular de dois ou mais processos, cada um dos quais está à espera de um recurso que está sendo usado pelo membro da cadeia.

Só ocorrerá *deadlock* se estás quatro condições estiverem presentes, ou seja, pode-se garantir que não ocorrerá *deadlock* na ausência de pelo menos uma destas condições.

2.3. Detecção de deadlock

Existem dois métodos para detectar *deadlock*, o método por grafos e o método por matrizes. O modelo de grafos indica *deadlock* quando um recurso de cada tipo e processos, estão em um ciclo (DA SILVA, 2009). Como por exemplo, utilizaremos quatro processos, simbolizado por círculos, e quatro recursos, simbolizado por quadrados. Onde o processo 1 tem o recurso A e solicita o recurso B, o processo 2 tem o recurso B e solicita o recurso C, o processo 3 tem o recurso C e solicita o recurso D, o processo 4 tem o recurso D e solicita o recurso A, formando assim uma cadeia circular de processos e recursos. Como pode ser observado na Figura 1.

Figura 1 – Impasses no modelo de detecção de grafos.



Ocorre *deadlock* no modelo de grafo da Figura 1, porque o processo 1, está bloqueado à espera de B, processo 2 está bloqueado à espera de C, processo 3 à espera de D e por fim, processo 4 está bloqueado à espera da liberação de A. Como pode ser observado, formou-se uma cadeia circular onde sucessivamente um processo espera a liberação de um recurso que já se encontra em posse de outro processo. Existem várias formas de corrigir esse impasse entre os processos, dois desses, seria se os recursos estivessem disponíveis ou se os processos não solicitassem outros recursos.

O modelo que nós nos propusemos a desenvolver para esse trabalho foi o usando o modelo matricial, porque esse é o método mais comum nos computadores atuais. Com esse método podemos simular um computador que pede várias instâncias (considere uma instância

como sendo uma cópia de um recurso) de um mesmo recurso, e vários outros recursos que também podem ter várias instâncias de mesmo recurso. Para resolver esse método é necessário um algoritmo baseado em matrizes para detectar *deadlock*. Na Figura 2 é apresentado o esquema para detectar *deadlock* com múltiplos recursos de cada tipo.

Figura 2 – Modelo de detecção matricial

Recursos Existentes (a)
$(E_1, E_2, E_3, \dots, E_m)$

Recursos Disponíveis (b)
$(A_1, A_2, A_3, \dots, A_m)$

Matriz de Alocação Atual (c)

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} & \dots & C_{1m} \\ C_{21} & C_{22} & C_{23} & \dots & C_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ C_{n1} & C_{n2} & C_{n3} & \dots & Cnm \end{bmatrix}$$

A linha n é a locação atual para o processo n

Matriz de Requisições (d)

$$\begin{bmatrix} R_{11} & R_{12} & R_{13} & \dots & R_{1m} \\ R_{21} & R_{22} & R_{23} & \dots & R_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ R_{n1} & R_{n2} & R_{n3} & \dots & Rnm \end{bmatrix}$$

A linha n informa qual é a necessidade do processos n

Na Figura 2(a) é apresentado a matriz de recursos existentes, que contém o número total de instâncias dos recursos E_1 à E_m. A Figura 2(b) mostrar a matriz de recurso disponíveis, que contém a quantidade de instâncias dos recursos A_1 à A_m que ainda não foram alocados pelos processos, ou seja, se A_1 igual a 0, significa que todas as instâncias do recurso A_1 estão alocados em algum processo qualquer. A Figura 2(c) apresenta a quantidade de instâncias de cada recurso que foram alocados em um instante qualquer pelos processos. A Figura 2(d) apresenta a matriz de requisição de recursos.

Detecta-se *deadlock* percorrendo as linhas da matriz de requisições e verificando se algum dos processos pode ser executado com os recursos disponíveis. Se sim, segue os passos:

1. Simular a entrega dos recursos para o processo;
2. Atualizar as matrizes de alocação, requisições e de recursos disponíveis;
3. Simular a conclusão do processo e a liberação dos seus recursos alocados;
4. Volta a percorrer as linhas da matriz de requisições e verifica as condições inicias.

Caso contrário, então existe *deadlock* entre os processos que não podem ser executados.

O método apresentado na Figura 2, obedece a coerência da Equação 1, onde o somatório das instâncias da linha 1 até a *n*-enésima linha da matriz de alocação, mais a coluna 1 até a *m*-enésima coluna de recursos disponíveis, a soma é igual a coluna 1 até a *m*-enésima coluna de recursos existentes.

$$\sum_{i=1}^{n} C_{ij} + A_j = E_j \qquad (1)$$

2.4. Formas de Recuperação de Deadlock

Uma vez detectado *deadlock* no sistema, existem três formas de recuperação (SILVA & LIMA, 2015):

- ***recuperação por preempção*:** Toma temporariamente um recurso preemptível de seu atual proprietário, para dá-lo a outro processo;
- ***reversão de estados*:** Nesse método, os processos geram pontos de salvaguardo (*checkpoints*). Com isso, quando ocorrer impasses os processos podem ser reiniciados para um momento anterior. Se o processo que sofreu reversão de estados tentar obter novamente o recurso, ele esperará até o recurso está disponível;
- ***eliminação de processos*:** É a maneira mais grosseira, porém a mais simples de eliminar um *deadlock*. Eliminando um ou mais processos presente no ciclo até o impasse ser quebrado.

3. METODOLOGIA

Na metodologia apresentaremos um breve resumo da ferramenta de desenvolvimento e da linguagem de programação utilizada para desenvolver o WPR que detecta *deadlock*.

3.1. Ferramenta para desenvolvimento: *Eclipse*®

O *Eclipse*® é um IDE para desenvolvimento Java, que teve a sua concepção iniciada na IBM que desenvolveu a primeira versão do programa e doou-o como *software* livre para a comunidade. É de fácil usabilidade para o desenvolvido da linguagem Java, oferecendo um amplo suporte com vários *plug-ins* que ajudam o desenvolver em diversas necessidades, o *software* foi feito em Java e segue o modelo de código aberto (LAGE *et.al.*, 2012).

3.2. Linguagem de programação: Java

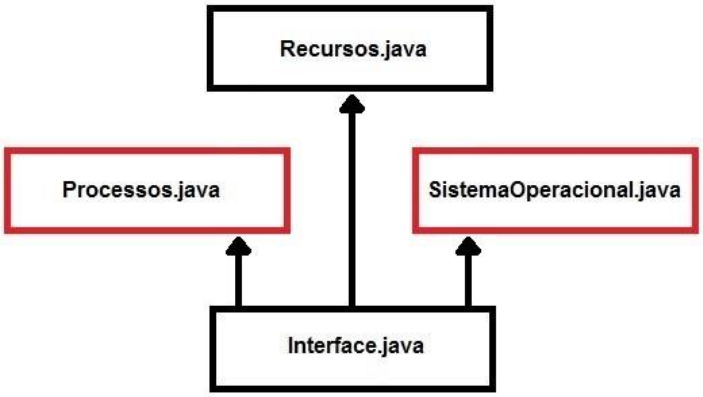
A escolha por desenvolver em Java, se deu porque é uma linguagem de programação orientada a objetivos, que teve seu projeto iniciado na empresa Sun Microsystems, na década de 90. É bastante conhecida no meio acadêmico, e possui bibliotecas de classes de fácil manipulação, exemplo disso, a classe Semaphore.java que limita o número de threads que podem acessar um recurso e que foi útil para o desenvolvimento do simulador de *deadlock*.

4. DESENVOLVIMENTO DA APLICAÇÃO

Como proposta para auxiliar os estudantes de engenharia foi desenvolvido o programa WPR para simular um sistema operacional, que faz todo o gerenciamento de recursos. No programa desenvolvido pelos autores, é possível criar ou excluir, até 10 recursos e até 10 processos, e inserir o tempo que ficará constantemente verificando o sistema. Na inserção de dados, é requerido que em processo seja informado um tempo de solicitação e o tempo de liberação de cada recurso, e em recurso é solicitado o nome e a quantidade de instâncias que cada recurso possui. Na interface do programa, é amostrada a matriz de recursos alocados que é modificado dinamicamente a cada solicitação e a cada liberação do recurso.

Para a implementação do programa foi desenvolvido quatro classes em Java que pode ser observado na Figura 4, sendo as classes Processos.java e SistemaOperacional.java threads.

Figura 4 – Classes desenvolvidas no programa



A classe Processos.java solicita, utiliza e liberar recursos existentes no sistema. A classe SistemaOperacional.java tem a função de verificar periodicamente se existe algum *deadlock* no sistema. Se houver, ela informada ao aluno quais processos então em *deadlock* e quais estão impedidos de rodar. A classe Recursos.java guarda em uma matriz os recursos que foram criados pelo usuário, e a classe Interface.java faz a comunicação entre as classes e apresenta os resultados dinamicamente das operações que são feitas no sistema.

5. RESULTADOS

A interface gráfica do simulador WPR auxilia e estimula os estudantes a quererem se aprofundar mais nos conceitos para depois testar na prática. Na interface do WPR é possível inserir os parâmetros necessários para a execução; tem a tabela de processos criados, que informar a situação desses processos, que podem está em situação de prontos, rodando, bloqueados ou em deadlock; tem também, a tabela de recursos criados, com a respectiva quantidade de instâncias; a matriz de recursos existentes, a matriz de recurso disponíveis e a matriz de alocação de recursos; e mostrar a situação do sistema que pode informar duas situações: sistema normal ou em *deadlock*.

Apresentaremos um exemplo para ser simulado no simulador, na Figura 5 apresentamos uma matriz de recursos existentes que tem 10 recursos com múltiplas instâncias, onde impressora tem 4 instâncias, unidade de DVD tem 3 instâncias, e assim por diante. A Tabela 1 mostra os parâmetros de tempo de solicitação e tempo de liberação dos processos inseridos no *software*, e na Figura 6, apresentamos o exemplo sendo simulado no WPR.

Figura 5 – Matriz de recursos existentes de 10 recursos.

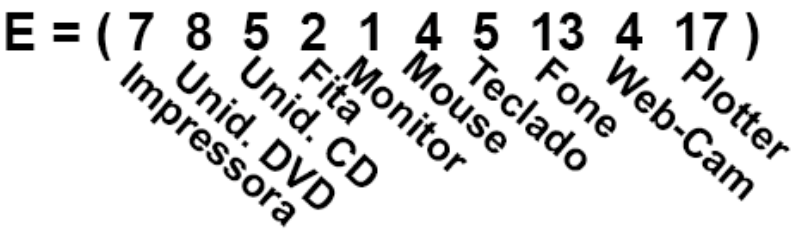
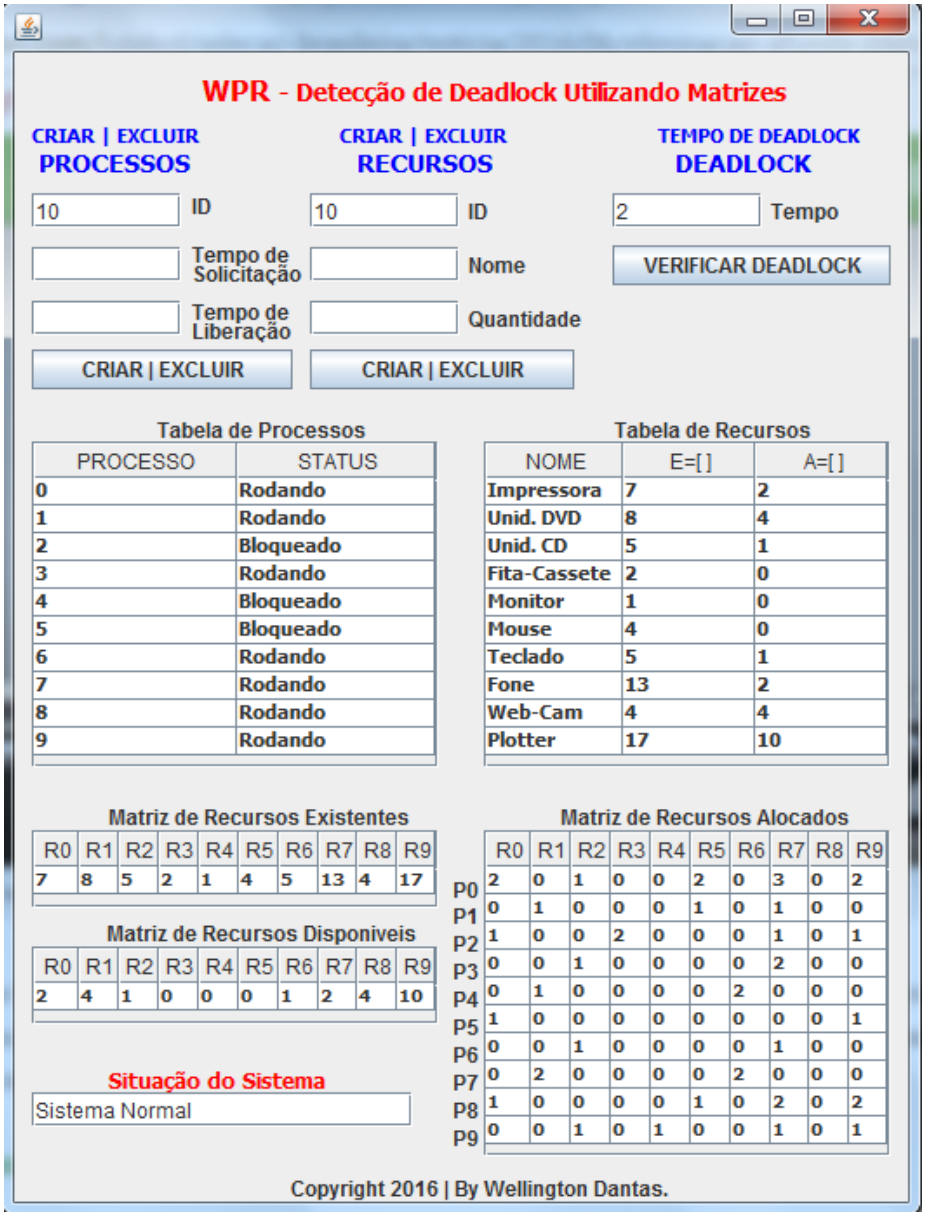


Tabela 1 - Parâmetros dos processos inseridos no simulador WPR

Processo	P0	P1	P2	P3	P4	P5	P6	P7	P8	P9
Tempo de Solicitação	1	1	1	1	1	1	1	1	1	1
Tempo de Liberação	10	3	5	3	3	2	2	4	6	4

Figura 6 – Interface do sistema que detectar a ocorrência de *deadlock*.



Para os parâmetros de criação dos processos, utilizamos tampos iguais a 1 segundo para solicitação e variados tempos para liberação. É importante explicar, que mesmo que o processo esteja em situação de bloqueado, não significa que esteja em *deadlock*, e nesse projeto em caso

de *deadlock*, é possível a recuperação por meio da eliminação de processos até que o sistema se normalize. A Figura 7 mostra para o mesmo exemplo, a detecção de *deadlock*.

Figura 7 – WPR detectando *deadlock*

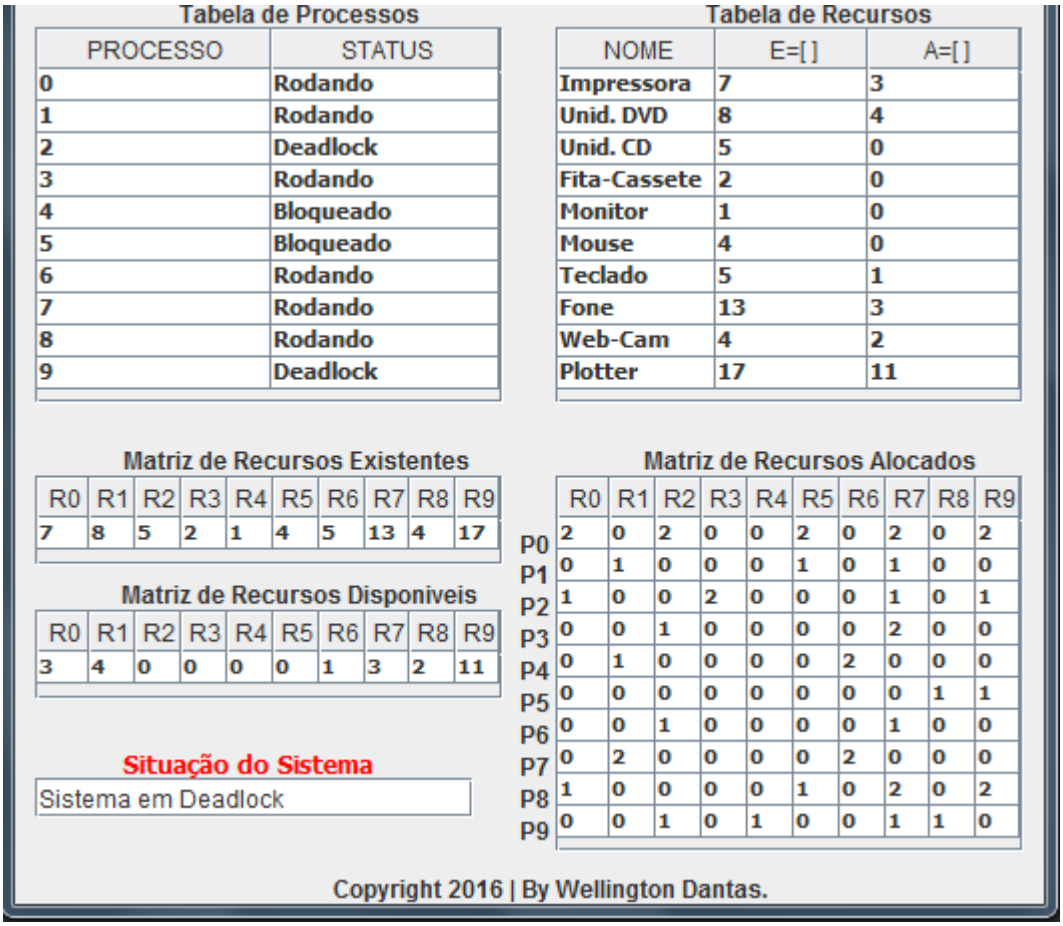


Tabela de Processos

PROCESSO	STATUS
0	Rodando
1	Rodando
2	Deadlock
3	Rodando
4	Bloqueado
5	Bloqueado
6	Rodando
7	Rodando
8	Rodando
9	Deadlock

Tabela de Recursos

NOME	E=[ ]	A=[ ]
Impressora	7	3
Unid. DVD	8	4
Unid. CD	5	0
Fita-Cassete	2	0
Monitor	1	0
Mouse	4	0
Teclado	5	1
Fone	13	3
Web-Cam	4	2
Plotter	17	11

Matriz de Recursos Existentes

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9
7	8	5	2	1	4	5	13	4	17

Matriz de Recursos Disponiveis

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9
3	4	0	0	0	0	1	3	2	11

Matriz de Recursos Alocados

	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9
P0	2	0	2	0	0	2	0	2	0	2
P1	0	1	0	0	0	1	0	1	0	0
P2	1	0	0	2	0	0	0	1	0	1
P3	0	0	1	0	0	0	0	2	0	0
P4	0	1	0	0	0	0	2	0	0	0
P5	0	0	0	0	0	0	0	0	1	1
P6	0	0	1	0	0	0	0	1	0	0
P7	0	2	0	0	0	0	2	0	0	0
P8	1	0	0	0	0	1	0	2	0	2
P9	0	0	1	0	1	0	0	1	1	0

Situação do Sistema

Sistema em Deadlock

Copyright 2016 | By Wellington Dantas.

No console do *Eclipse*®, Figura 8 é possível ser as etapas de solicitação, e liberação dos processos pelo sistema. Também a quantidade de instâncias que cada recurso possui.

Figura 8 - Solicitação e liberação de recursos pelos processos

```
Console
<terminated> Exemplo [Java Application] C:\Program Files\J
Processo 1 solicitou Unid.CD - R[2]=3
Processo 1 usar Unid.CD - R[2]=2
Processo 0 solicitou Impressora - R[0]=6
Processo 3 solicitou Monitor - R[4]=1
Processo 0 usar Impressora - R[0]=5
Processo 3 usar Monitor - R[0]=0
Processo 1 liberou Unid.CD - R[2]=3
Processo 7 solicitou Teclado - R[6]=3
Processo 7 usar Teclado - R[6]=3
Processo 4 solicitou Monitor - R[4]=0
Processo 4 bloqueado!
```

Observando-se a Figura 7, verificar-se que os processos solicitam aleatoriamente os recursos, e quando o uso é autorizado pelo sistema os recursos decrementam, concomitantemente, quando os recursos são liberados pelos processos, os recursos incrementam. Quanto maior for o *timesleep* de um processo para liberar um recurso, mais chance haverá de ocorrer impedimentos entre os processos, e assim como consequência *deadlock.*

6. CONSIDERAÇÕES FINAIS

Tendo como objetivo contribuir ao ensino, o software WPR realiza os procedimentos para encontrar *deadlock,* utilizando o método de detecção por matrizes, fazendo as tarefas básicas que um sistema operacional realiza, como alocação de recursos e recuperação do sistema em casos de *deadlock.*

Do ponto de vista educacional, o programa contribui para o entendimento do estudante de forma prática. A visualização de como os recursos são alocados e as situações de *deadlock*, trazem ao aluno a compreensão sobre a ocorrência deste fenômeno. Em pesquisa realizada pelos autores direcionada aos alunos, foi perguntado se a prática desse simulador melhorava na compreensão do conteúdo visto em sala de aula, e com base nas respostas de 20 alunos, foi obtido que 90% dos alunos entrevistados, ou seja, 18 destes, acham que o *software* WPR contribui significativamente ao ensino.

Agradecimentos

Os autores agradecem ao Laboratório de Processamento Digital de Sinais (LPDS), a todo o Departamento de Telemática (DETEL) e aos órgãos de fomentos CNPq e FUNCAP que possibilitaram de alguma forma, seja por meio de materiais e instalações ou financeiramente, os resultados aqui obtidos.

REFERÊNCIAS BIBLIOGRÁFICAS

DA CRUZ, L. B. Projeto de Um Framework para o Desenvolvimento de Aplicações de Simulação Distribuída, 2009. 108p, il. Dissertação (Mestrado).

GIANCARLO F. A, ALESSANDRO B, VOLMIR E. W; Concepção e testes de software no processo de ensino-aprendizagem da disciplina pesquisa operacional. Anais: XL – Congresso Brasileiro de Educação em Engenharia, Belém: UFPA, 2012.

GUEDES, A.; Do Papel e Lápis ao Mundo Real: Estudo de Caso no Ensino da Mecânica de Fluidos. Anais: XLII – Congresso Brasileiro de Educação em Engenharia, Belém: UFJF, 2014.

LAGE, A.V.; LEAL, C. D. O.; DA SILVA, E. A.; TRINDADE, F.C.; OLIVEIRA, G. D. C e CARVALHO, M.Tutorial de Instalação e Uso do Eclipse. RevISTa,Rio de Janeiro, 2012.

SILVA, L. de F.; LIMA, G. D. de O. Detecção e correção de situações de deadlock nas redes de petri. Revista GeTeC, v. 4, n. 7, 2015.

TANENBAUM, A. S; Sistemas Operacionais Moderno. 3 ed. LTC, São Paulo: Pearson, 2010.

PROGRAM DETECTING DEADLOCK WITH MULTIPLE RESOURCES TO ASSIST IN ENGINEERING STUDENTS LEARNING: AN APPROACH TO OPERATING SYSTEMS OF DISCIPLINE

Abstract: *This work presents an educational tool to aid the apprenticeship process in Computer Engineering and Telecommunications Engineering students that attend the Operating Systems discipline at Federal Institute of the Ceará (IFCE).In that discipline. It was proposed a program to detect the incident of deadlock between processes, termed WPR. The main objective of the WPR is to open a reflexive vision, for the students, of how an operating system works. An elementary study was conducted concerning the modern operating systems in an introductory way, the theoretical foundation with respect to the concepts of deadlocks, the program development methodologies, the software application itself and the evident results from the graphical interface.*

Key-words: *Engineering education, Learning, Deadlock, Operating systems.*