Uma Ferramenta de Apoio à Aprendizagem de Sistemas Operacionais

Shigueo Isotani[1], Carlos Henrique Jorge[1], Nelson Miguel Quitério Junior[1], Fabiano Souza da Silva[1], Seiji Isotani[2]

[1]Departamento de Ciência da Computação
Faculdade de Ciências da Fundação Instituto Tecnológico de Osasco (FAC-FITO)
Osasco, SP

[2]Department of Knowledge Systems
The Institute of Scientific and Industrial Research
Osaka University, Japan

shigueo.isotani@gmail.com, chenriquej@gmail.com, nmiguelq@gmail.com, sfabiano.souza@gmail.com, isotani@ime.usp.br

Abstract. *We presented a tool to aid the student that studies the discipline of Operating Systems to understand better the behavior of the processes on operating systems. This tool captures user processes that are being executed in the computer in real time and shows those results graphically. The accomplished experiments show that the results obtained by the capture of processes information in the Operating System and that obtained accomplishing the simulations of scheduling algorithms can be compared in a clear and objective way. In that way, the student can easily visualize and distinguish the behavior of the different scheduling algorithms.*

Resumo. *Apresentamos uma ferramenta para auxiliar o aluno que cursa a disciplina de Sistemas Operacionais a compreender melhor o comportamento dos processos em sistemas operacionais. Esta ferramenta captura processos de usuário que estão sendo executados no computador em tempo real e mostra esses resultados graficamente. Os experimentos realizados mostram que os resultados obtidos pela captura de dados dos processos no Sistema Operacional e aqueles obtidos realizando as simulações dos algoritmos de escalonamento podem ser comparados de maneira clara e objetiva. Dessa forma, o aluno pode facilmente visualizar e distinguir o comportamento dos diferentes algoritmos de escalonamento.*

1. Introdução

Nos computadores atuais tem-se a impressão de que são realizadas várias operações ao mesmo tempo, como, por exemplo, editar um texto, baixar arquivos da Internet, entre outras. A falsa impressão de paralelismo ocorre por causa do escalonamento dos processos e da alta velocidade dos processadores, embora o processador execute um processo de cada vez. O paralelismo somente acontece em sistemas multiprocessados,

ou seja, com dois ou mais processadores físicos ou lógicos, em que cada processador executa processos autonomamente [Tanembaum 2000].

Como ferramenta de auxílio à compreensão dos princípios conceituais destes processos, podemos citar a ferramenta pioneira GraphOS [Cañas 1987] que mostra em tempo real o fluxo de informação dos processos em execução. Esta ferramenta foi desenvolvida para complementar as aulas dos cursos de Sistemas Operacionais permitindo ao estudante analisar em tempo real o fluxo de informações dos processos em execução.

Devido à complexidade deste assunto, há muitos estudos sobre algoritmos de escalonamento de processos que procuram mostrar o resultado empírico da comparação entre as várias soluções existentes. A este respeito, existem alguns simuladores de escalonamento de processos voltados para cursos de Sistemas Operacionais, como o SOsim apresentado inicialmente por Maia [Maia 2001] e sua versão 2.0 lançada em 2007 [Machado e Maia 2007], o Simulador para a Prática de Sistemas Operacionais [Carvalho et al 2006], o Sistema Operacional Integrado Simulado: Módulo de Entrada e Saída [Cruz et al 2007] e o wxProc [Rocha et al 2004] que simulam o que acontece durante um escalonamento. Nestes simuladores os processos são personalizados, isto é, o aluno escolhe o tempo de execução, prioridade, e outros parâmetros.

Para aperfeiçoar a utilização destas ferramentas no entendimento de sistemas operacionais, desenvolvemos uma ferramenta que integra a simulação de escalonamentos com a coleta de informação dos processos em tempo real, mostrando-os graficamente. Embora já existam simuladores de escalonamento de processos, a idéia é melhorar e avançar para contribuir no aprendizado deste assunto. Assim, através desta ferramenta, desejamos transmitir de uma forma objetiva o que vem a ser um escalonamento de processos, comparando uma simulação com o comportamento real dos processos em execução na máquina em determinado momento. A nossa idéia é justamente aproximar a teoria com o mundo real através da realização de simulações usando informações de processos reais.

A ferramenta foi desenvolvida utilizando duas linguagens, a linguagem C para coletar informações do sistema operacional, e a linguagem Java para a visualização gráfica dos resultados. A comunicação entre as duas linguagens para a troca de informações é feita via *Java Native Interface* (JNI).

2. Processos e Escalonamentos

No modelo de processos adotado neste trabalho, todos os softwares executáveis, incluindo o sistema operacional, são organizados em processos seqüenciais. O processo é o programa em execução.

Um processo só pode estar em um dos seguintes estados: *em execução*, *pronto* ou *bloqueado*. Em execução é quando o processo está utilizando o processador. Pronto, quando o processo está aguardando sua vez para utilizar o processador. Bloqueado, quando fica incapaz de executar até que algum evento, normalmente algo externo, aconteça para que o processo possa continuar a execução [Tanembaum 2000].

O escalonamento de processos na CPU tem como objetivo otimizar a utilização do processador, e ao mesmo tempo, diminuir o tempo que cada processo irá gastar. De

uma forma geral, todos os processos passam por um escalonamento para que possa ser executado. Cada sistema operacional trabalha com algoritmos diferentes de escalonamento, não havendo um algoritmo ou solução padrão para essa tarefa. Quando há escalonamento na CPU, tirando um processo de execução e colocando outro para executar, o processo escalonado terá seu último estado salvo, assim como o novo processo tem seu estado carregado para a CPU. Tal ação é chamada de troca de contexto. O contexto de um processo é representado por seu bloco de controle de processos contendo o valor dos registradores da CPU, o estado do processo, limite e gerenciamento de memória. Os registradores variam conforme a arquitetura do computador, composto por acumuladores, índice, ponteiros de pilhas, etc [Silberschatz 2004].

A velocidade com que é feita a troca de contexto pode ser muito rápida dependendo muito do hardware em questão, ou seja, velocidade de memória, quantidade de registradores a serem copiados ou armazenados. Esta velocidade de troca varia em torno de 1 a 1000 microssegundos. Quanto mais complexo for o sistema operacional, mais trabalho será realizado em uma troca de contexto requerendo técnicas avançadas de gerenciamento de memória [Silberschatz 2004].

O escalonamento pode ser não preemptivo ou preemptivo. Não preemptivo é a política de permitir que um processo alocado na CPU (em execução) não possa ser interrompido até seu término. Em contrapartida, um escalonamento preemptivo permite que os processos sejam suspensos temporariamente, cedendo a vez a outro processo. O escalonamento de processos no Windows 2000 utiliza um algoritmo com preempção baseado em prioridades [Silberschatz 2004].

Existem vários algoritmos de escalonamento, dentre os quais descreveremos neste trabalho o *Round-Robin* (RR), *First-Come First-Served* (FCFS) e *Shortest-Job-First* (SJF) [Silberschatz 2004].

O algoritmo de escalonamento RR é especialmente utilizado em sistemas de tempo compartilhado. Este algoritmo utiliza a preempção para efetuar a troca de processos na CPU usando uma unidade de tempo chamada quantum. Quando um processo está em execução e termina o seu quantum, esse processo é retirado de execução pelo escalonador e é colocado no final da fila de processos prontos, respeitando a ordem de chegada na fila, dando a todos os processos a mesma oportunidade de utilização da CPU. No caso do processo ser encerrado antes do seu quantum este deixa a CPU, disponibilizando-a para o próximo processo da fila.

O algoritmo FCFS é o mais simples dos algoritmos de escalonamento aqui citados. Quando os processos ficam disponíveis para execução, são colocados na fila de processos prontos para execução. O algoritmo FCFS mantém a ordem de chegada desses processos na fila de prontos, o primeiro processo da fila é executado até o término, independente do seu tamanho. Com isso o tempo médio de espera varia muito, pois tudo vai depender da ordem de chegada dos processos na fila de prontos ocorrendo assim o chamado efeito comboio. Apesar de ser um algoritmo de simples implementação, não é muito aconselhável a sua utilização em sistemas de tempo compartilhado.

O algoritmo SJF ordena os processos de acordo com o tempo de execução de cada um deles. Um algoritmo como este, proporcionará sempre uma solução ótima, mas

o problema é saber de antemão qual o tempo total de execução de cada processo que se encontra na fila de processos prontos. Este é o mais apropriado para escalonamento de processos com tempos de execução já determinados.

3. O sistema

O sistema é estruturado em camadas, a saber, o sistema operacional na base, o leitor de processos e o aplicativo em Java.

O leitor de processos foi desenvolvido em linguagem C sendo responsável pela captura das informações dos processos existentes no Windows, e pela transformação dos dados para que possam ser utilizados pela aplicação Java. A aplicação Java é responsável pela simulação e exibição do resultado em forma de gráficos e tabelas. A comunicação entre as duas linguagens para a troca de informações é feita via JNI. Para gerar as simulações, é necessário que haja algumas informações tais como, identificador único do processo (PID), os nomes dos processos, tempo de permanência e tempo de serviço, entre outras. Para tornar o simulador mais realista, essas informações são capturadas do próprio sistema operacional Windows utilizando bibliotecas já existentes.

O objetivo do leitor de processos é de capturar as informações dos processos existentes no sistema operacional. Para isso, ele irá utilizar as funções da biblioteca do próprio Windows: *Process32Next* que percorre toda a lista de processo, nesta lista existe, o PID e o nome do processo; *GetProcessTimes* que é utilizada para capturar as informações referentes ao tempo, como: Data Início, Data Fim, Tempo de Kernel, Tempo de Usuário; *OpenProcessToken* que é utilizada para verificar se um determinado processo é de usuário, pois a função *Process32Next* percorre toda a lista de processos, inclusive os processos de sistema.

O leitor de processos faz uma verificação, no sistema operacional em intervalo de tempo fixo para detectar a existência ou não de um novo processo. Caso haja um novo, ele é incluído na lista de processos. E a cada intervalo de tempo é feita uma atualização das informações de todos os processos dessa lista. A lista de processos de usuário é armazenada em uma estrutura de árvore binária para otimizar a performance, pois o leitor de processos não deve interferir nos demais processos.

Antes de enviar os dados dos processos para a aplicação Java, é feita uma transformação/tratamento dos dados, pois o sistema captura as informações data início, data fim, tempo de *kernel* e tempo de usuário, porém a aplicação Java necessita do tempo de duração do processo e de utilização da CPU.

A aplicação Java está dividida em camadas, a de comunicação que interage com o leitor de processos, a de simulação e a de exibição que interage com as camadas de comunicação e de simulação.

O JNI é responsável por estabelecer uma comunicação entre o aplicativo Java e o leitor de processos. O momento em que a aplicação Java inicia o processo de captura é feito através da chamada de uma função dentro do leitor de processos.

Com relação à camada de comunicação, existe um fluxo que pode ser definido da seguinte forma: a) acionamento do leitor de processos que inicia o processo de captura das informações dos processos; b) recepção/tratamento dos dados onde o leitor de processos envia a cada intervalo de tempo uma imagem de todos os processos. Estes

são armazenados pela aplicação Java; c) desligamento do leitor de processos que interrompe a comunicação. A Figura 1 mostra o diagrama de seqüência para o processo de comunicação. O processo de execução é uma chamada assíncrona, pois enquanto o componente executa o monitoramento, a interface do usuário deve ser atualizada pelos dados recebidos, e também deve permitir que o processo seja interrompido pelo usuário a qualquer momento.

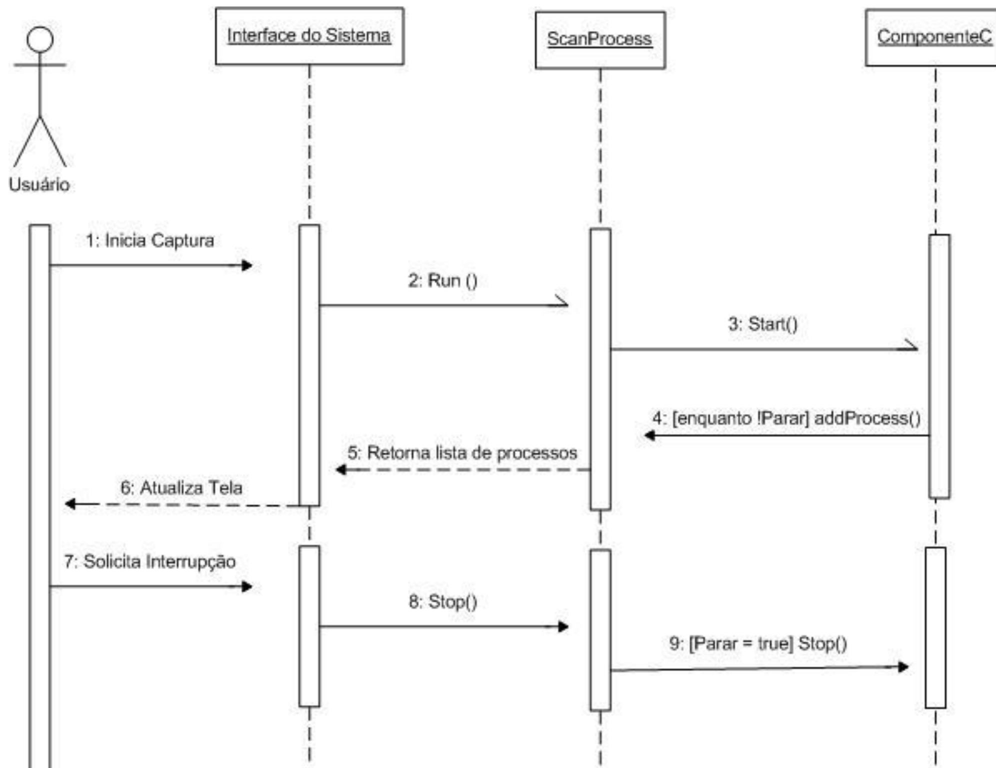

Figura 1. Diagrama de seqüência da camada de comunicação

O tratamento de dados feito na camada de comunicação utiliza a estrutura de árvore binária para agilizar o processo de pesquisa, uma vez que, a cada processo recebido, é necessário verificar se o mesmo já pertence à lista de processos existente. Também é importante mencionar que para cada processo, haverá uma lista encadeada de consumo de CPU. Em seguida os dados serão enviados para a camada de exibição, para que possam ser mostrados na tela, ficando totalmente independente da camada de comunicação.

A camada de simulação é responsável por simular o escalonamento de acordo com o algoritmo selecionado. Os dados de entrada são os dados capturados pela camada de comunicação (PID, nome e tempo total de CPU), e como saída são gerados os dados com o mesmo formato gerado na captura, facilitando a comparação dos resultados obtidos. No simulador os seguintes parâmetros podem ser alterados: *clock*: a freqüência de interrupção do relógio que define quantas vezes por segundo o *kernel* irá entrar em execução, sendo que o padrão é 60 hertz; troca de contexto: representa o tempo que o *kernel* irá gastar para trocar o processo na CPU sendo o valor padrão zero; quantum

(apenas para RR): sendo o valor padrão 6. O simulador pode gerar as simulações com os seguintes algoritmos: RR, FCFS e SJF.

A camada de exibição recebe os dados enviados pela camada de comunicação ou pela de simulação, e irá exibi-los em um *grid*. Com relação à camada de comunicação, a atualização da tela acontecerá a cada um segundo, pois assim o usuário poderá acompanhar a situação de cada processo em tempo real. No *grid* serão exibidas sempre as 20 últimas leituras para facilitar a visualização. A visualização do resultado, também poderá ser feita através de gráficos, para isso foi desenvolvido dois tipos de gráficos.

Um gráfico com barras verticais mostra o resultado da captura. O eixo X representa os dados gerados pela captura e o eixo Y representa a quantidade de CPU consumida dentro de cada captura. Este gráfico não permite determinar a ordem de execução se tivermos mais de um processo. Porém, com este gráfico é possível se ter uma idéia do comportamento de cada processo, permitindo compará-los visualmente.

Um gráfico com barras horizontais mostra o comportamento dos algoritmos. O eixo X representa a linha do tempo, onde a menor unidade é um *tick (1000 milisegundos)*, assim é possível determinar com precisão, qual processo executou em um determinado tempo. Porém, não é possível gerar este gráfico para os processos em tempo real uma vez que nem sempre o *tick* coincide com a freqüência do *clock*. Sendo assim, este gráfico será gerado apenas para os dados do simulador.

4. Resultados

Realizamos os testes utilizado um *notebook* com processador Intel Celeron M 1.60 Ghz, 504 MB RAM com Windows XP Professional SP2. O intervalo de tempo utilizado entre as coletas de dados foi de 100 ms. Os processos que usamos foram o *Quick Sort, Heap Sort* e *Bubble Sort*. O *Quick Sort* foi aplicado no ordenamento de 10.000.000 números, o *heap sort* foi aplicado na ordenação de 3.000.000 números e o *Bubble Sort* ordenou 40.000 números. Os tempos de início dos processos foram de zero, 500, 1000 ms, respectivamente para o *Bubble Sort, Heap Sort e Quick Sort,* para facilitar a visualização gráfica do escalonamento. A simulação foi feita com todos os processos capturados, e a partir desta simulação foi gerado o gráfico somente destes processos. Na simulação não consideramos fatores como alocação de memória, acesso a disco e I/O (E/S).

A Figura 2 mostra o gráfico vertical gerado a partir dos dados capturados do Windows durante o teste. Ele mostra o comportamento dos processos utilizados. Neste gráfico é possível identificar que ocorrem interrupções na execução dos processos, possivelmente por sobrecarga do sistema.

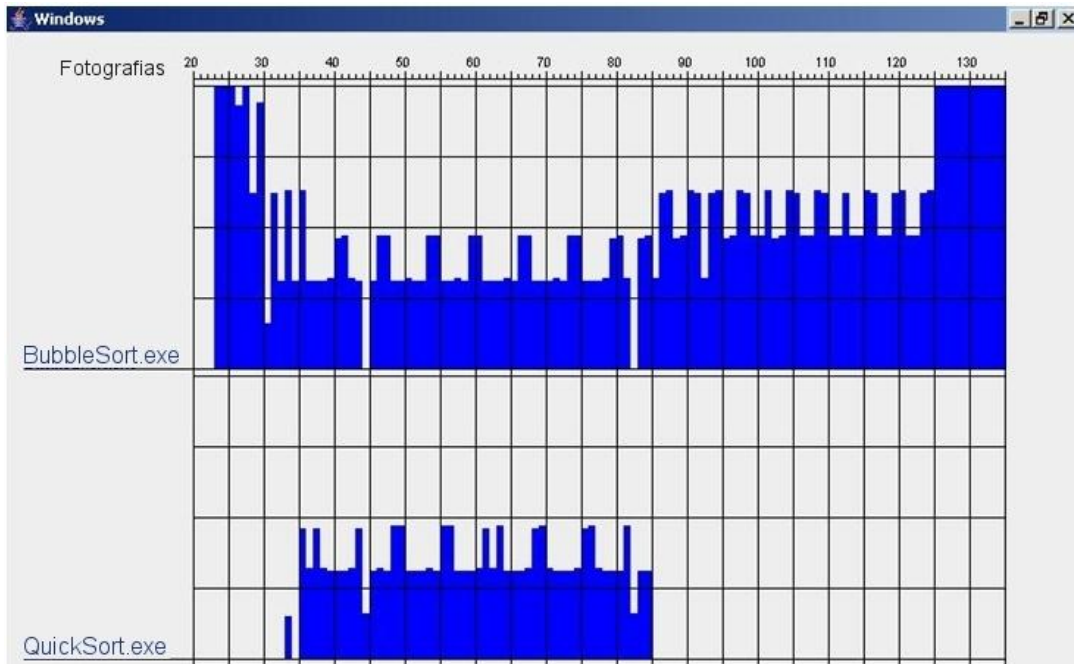

Figura 2. Gráfico vertical do Windows mostrando os processos em real-time dos processos Bubble Sort e Quick Sort.

A Figura 3 mostra o gráfico vertical do resultado da simulação utilizando o algoritmo RR com o quantum igual a dois. Note que, esta simulação gerou um resultado muito parecido com o do Windows mostrado na Figura 2.

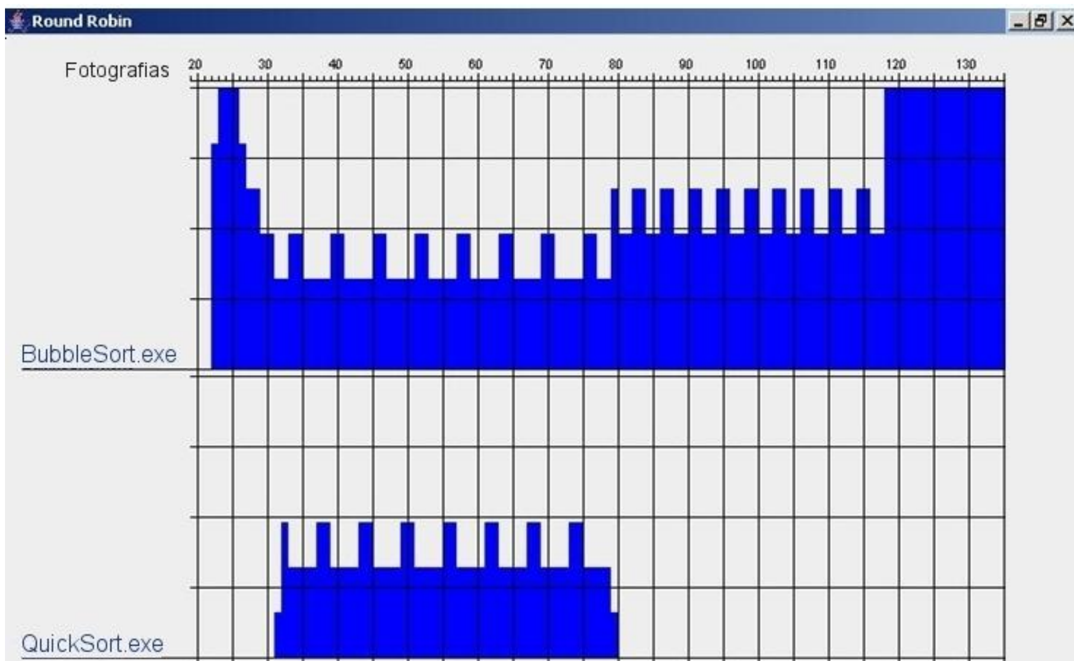

Figura 3. Gráfico vertical mostrando a simulação dos processos Bubble Sort e Quick Sort usando o RR.

A Figura 4 mostra o gráfico horizontal da simulação utilizando o algoritmo RR. Note que, com este gráfico é possível determinar com precisão qual processo executou em um determinado tempo, podemos ver que todos os processos estão tendo as suas fatias de tempo.

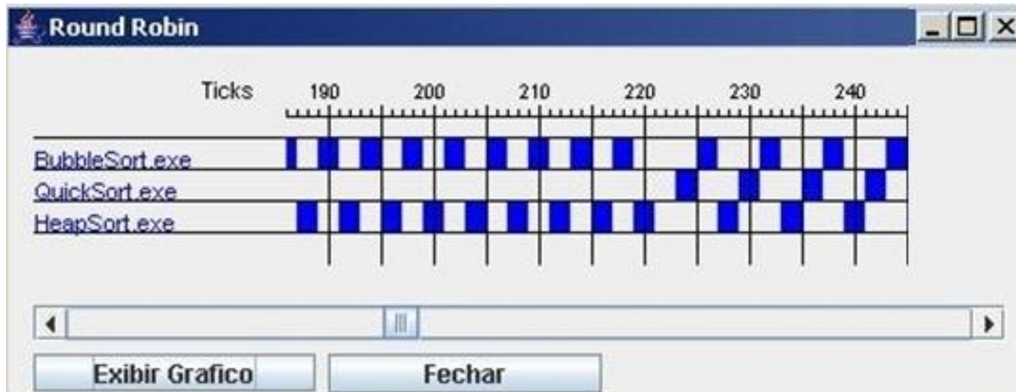

Figura 4. Gráfico horizontal mostrando a simulação dos processos Bubble Sort, Quick Sort e Heap Sort usando RR.

A Figura 5 mostra o mesmo teste utilizando o algoritmo SJF. Note que, apesar de o *Bubble Sort* ser o mais longo, quando ele foi iniciado ainda não existiam os demais, porém, assim que o *Heap Sort* iniciou, ele foi colocado em execução, pois ele era mais curto que o *Bubble Sort*. A mesma coisa aconteceu quando o *Quick Sort* foi criado. Isso mostra que o algoritmo está priorizando o processo mais curto.

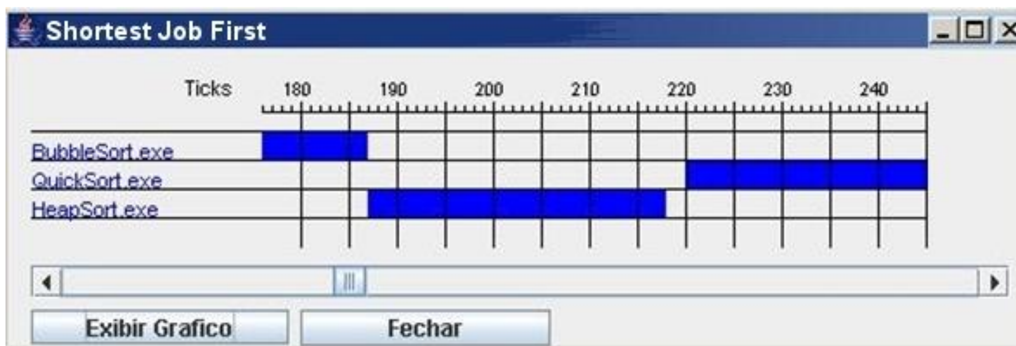

Figura 5. Gráfico horizontal mostrando a simulação dos processos Bubble Sort, Quick Sort e Heap Sort usando SJF.

A Figura 6 mostra o resultado da simulação com o algoritmo FCFS. Na parte de cima está o início do gráfico, e logo abaixo a sua continuação, note que, o *Bubble Sort* executou até terminar, e em seguida o *Heap Sort* começou, mostrando que o algoritmo está executando por ordem de chegada.

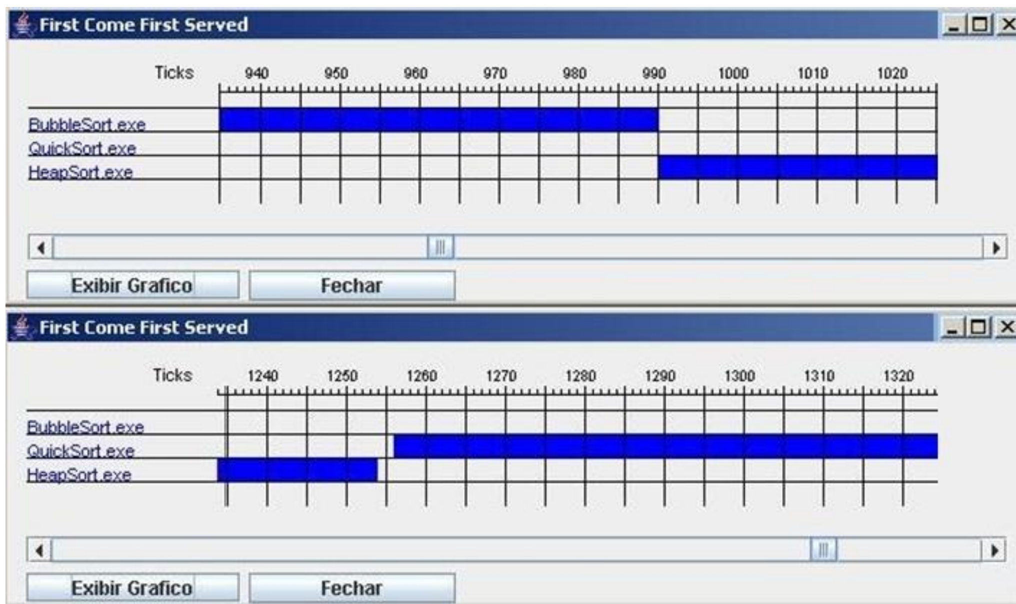

Figura 6. Gráfico horizontal mostrando a simulação dos processos Bubble Sort, Quick Sort e Heap Sort usando FCFS.

Desta forma, se o aluno tiver coletado os dados em tempo real no Windows é possível efetuar simulações com diferentes algoritmos de escalonamento.

5. Conclusões

Desenvolvemos uma ferramenta que permite capturar os dados dos processos em execução na máquina, e a partir desses dados gerar simulações com os algoritmos RR, FCFS e SJF. Para as simulações, a ferramenta permite que sejam modificados os valores de *clock*, tempo de captura, troca de contexto e quantum. Além das simulações, também é possível monitorar o consumo de CPU de cada processo em tempo real. Tanto os dados das simulações como os em tempo real, são exibidos em forma de tabela e gráficos.

Com os resultados obtidos nos testes com a utilização de dados em tempo real capturados do sistema operacional, é possível gerar simulações com os algoritmos, permitindo que seja feita uma comparação entre os dados capturados e os gerados pelo simulador. Também é possível fazer comparações entre os algoritmos, permitindo que se identifique qual o melhor algoritmo para uma determinada situação.

Podemos concluir que cada algoritmo foi desenvolvido para atender uma situação distinta. No caso dos algoritmos utilizados na ferramenta de simulação, sabemos que o algoritmo FCFS pode ser utilizado em execuções de processos que tenham a ordem definida, não estando sujeitos a preempção. O algoritmo SJF pode ser utilizado para execução de processos em lotes, pois o mesmo já tem os seus tamanhos definidos. O algoritmo RR é ideal para utilização em sistemas iterativos (computadores pessoais, por exemplo), pois dá a cada processo a oportunidade de utilizar o processador sem ter que esperar o término do processo anterior. Sendo assim, não se pode afirmar qual algoritmo é o melhor ou mais eficiente, tudo vai depender do tipo de sistema em que ele será utilizado.

Conseguimos mostrar que a utilização desta ferramenta contribui para o aprendizado dos algoritmos de escalonamento, pois torna visíveis suas diferenças através da geração de gráficos. Com os gráficos temos a oportunidade de visualizar o comportamento de cada algoritmo e também do Windows, permitindo que se faça uma comparação visual entre cada um, como foi mostrado no teste de simulação.

Dando continuidade ao presente trabalho planejamos desenvolver uma ferramenta semelhante para a captura da alocação de memória em cada processo, e em seguida gerar simulações com esses dados. Dessa forma teríamos uma ferramenta para auxiliar no aprendizado não só dos algoritmos de escalonamento como também na alocação de memória.

6. Referências

Carvalho, D. S., Balthazar, G. R., Dias, C. R., Araújo, M. A. P. e Monteiro, P. H. R. (2006) "Simulador para a Prática de Sistemas Operacionais", XXVI Congresso da Sociedade Brasileira de Computação (CSBC), Campo Grande, MS.

Cañas, D. A. (1987) "Graphos: a graphic operating system", ACM SIGCSE, Bulletin, Volume 19 (1) 201 – 205.

Cruz, E. H. M., Silva, V. P. e Gonçalves, R. A. L. (2007) "Sistema Operacional Integrado Simulado: Módulo de Entrada e Saída", XIV Escola Regional de Informática da SBC, Guarapuava, PR.

Machado, F. B. e Maia, L. P. (2007) "Arquitetura de Sistemas Operacionais", Editora LTC.

Maia, P. L. (2001) "SOsim: Simulador para o Ensino de Sistemas Operacionais", Dissertação de Mestrado, NCE/UFRJ, http://www.training.com.br/sosim, acessado em 04/08/2008.

Rocha, A. R., Schneider, A., Alves, J. C., Silva, R. M. A. (2004) "wxProc: Um Simulador de Políticas de Escalonamento Multiplataforma", INFOCOMP Journal of Computer Science, Volume 3 (1) 43- 47.

Silberschatz, A., Galvin, P. B., Gagne, G. (2004) "Fundamentos de Sistemas Operacionais", tradução de Elisabete do Rego Lins, sexta Edição, Editora LTC.

Tanembaum, A. S. (2000) "Sistemas Operacionais: Projeto e Implementação", tradução de Edson Furmankiewicz, Editora Bookman.