UNIVERSIDADE DO EXTREMO SUL CATARINENSE - UNESC

CIÊNCIA DA COMPUTAÇÃO

MAYKO SARTOR

SOLAB - FERRAMENTA DE APOIO AO ENSINO DE SISTEMAS OPERACIONAIS

CRICIÚMA, DEZEMBRO DE 2008

MAYKO SARTOR

SOLAB - FERRAMENTA DE APOIO AO ENSINO DE SISTEMAS OPERACIONAIS

Trabalho de Conclusão de Curso apresentado para obtenção do grau de Bacharel em Ciência da Computação da Universidade do Extremo Sul Catarinense.

Orientador: Prof. MSc. Paulo João Martins

CRICIÚMA, DEZEMBRO DE 2008

MAYKO SARTOR

SOLAB – FERRAMENTA DE APOIO AO ENSINO DE SISTEMAS OPERACIONAIS

Submetido ao corpo docente do Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense como um dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Profa. MSc. Ana Claudia Garcia Barbosa
Coordenadora do Curso de Ciência da Computação

Banca Examinadora:

Prof. MSc. Paulo João Martins (UNESC)
Orientador

Profa. Dra. Ledina Lentz Pereira (UNESC)

Prof. MSc. Ricardo Portes (SERPRO)

Dedico esta pesquisa aos meus pais e a minha irmã por tudo que fizeram e ainda fazem em todos os aspectos da minha vida.

AGRADECIMENTOS

Agradeço a Deus por ter me concedido a família que tenho e por ter me concedido a permissão de existir e evoluir como ser humano.

A minha mãe, Maria de Fátima Sartor, mulher honesta e de muita garra, que sempre me incentivou e torceu muito pela minha conquista. Palavras não bastam para descrever a profunda admiração e o amor que sinto.

Ao meu pai Walmor Sartor (falecido), que em vida sempre me incentivou e lutou para tornar meus sonhos possíveis. Tenho muitas saudades.

Ao meu orientador Sr. Paulo João Martins pela sua disposição, atenção, compreensão, incentivo e experiências transmitidas, além da grande amizade formada.

A todos os professores que tive, pela sua contribuição no desenvolvimento dessa pesquisa.

Agradeço aos meus amigos que sempre me deram força, incentivo e apoio, estando sempre do meu lado.

“O covarde nunca tenta, o fracassado nunca termina e o vencedor nunca desiste." (Norman Vicent Peale)

RESUMO

Sistema operacional é um conjunto de programas que propicia ao usuário utilizar o hardware do computador. Ele é muito complexo, pois serve de suporte para todos os demais programas. Dentro do curso de Ciência da Computação existe uma disciplina específica para ensinar o seu funcionamento, apresentando teorias que são de grande importância para a formação do acadêmico.

Esta pesquisa teve como objetivo principal desenvolver um simulador para apoio ao ensino da disciplina de Sistemas Operacionais. A vantagem da utilização de um simulador é permitir ao acadêmico interagir com os conceitos apresentados em sala de aula. Assim foram utilizadas algumas técnicas para facilitar a visualização de informações e a interação com o acadêmico de forma a dinamizar o processo de aprendizagem. O simulador desenvolvido busca apresentar de forma visual, por meio de técnicas de animação, o funcionamento de um escalonador e um gerenciador de memória, componentes fundamentais dentro de um sistema operacional. Devido à complexidade envolvida, somente as funcionalidades principais foram implementadas para que as animações fossem simples e de fácil entendimento.

Palavras-Chave: Ferramentas de Apoio ao Ensino; SOLAB; Sistemas Operacionais.

ABSTRACT

Operating System is a set of programs that provides the user to use the computer hardware. It is very complex; it serves as a support for all other programs. In the course of Computer Science there is a specific subject to teach its operation, presenting theories that are of great importance to the academic training.

This study aimed to develop a simulator for primary education support of the discipline of Operating Systems. The advantage of using a simulator is allowing the academic to interact with the concepts presented in the classroom. There were some techniques used to facilitate viewing of information and interaction with the academic in order to boost the learning process. The simulator developed presents in a visual display, through techniques of animation, the operation of a scheduler and a memory manager, key components within an operating system. Because of the complexity involved, only the main features were implemented so that the animations were simple and easy to understand.

Keywords: Tools of Support to Education; SOLAB; Operating Systems.

LISTA DE SIGLAS

API	Advanced Programming Interface
BASIC	Beginners All-purpose Symbolic Instruction Code
BIT	Binary Digit
COM	Component Object Model
CPU	Central Processing Unit
FIFO	First In First Out
JDK	Java Development Kit
MMU	Memory Management Unit
MS-DOS	Microsoft Disk Operating System
SJF	Shortest Job First
SO	Sistemas Operacionais
UML	Unified Modeling Language
UNESC	Universidade do Extremo Sul Catarinense

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Sistemas Operacionais é uma disciplina importante no curso de Ciência da Computação que engloba assuntos complexos como, por exemplo, o conceito de gerência de processador e memória, algoritmos de escalonamento e comunicação entre processos.

A disciplina é caracterizada por uma elevada carga teórica, sendo que a experiência de alunos e professores mostra que existem dificuldades em compreender os conceitos abordados. A principal dificuldade está no fato de existirem poucos modelos, que possam simular e demonstrar as teorias apresentadas de forma a facilitar a compreensão por parte dos acadêmicos.

Desta forma, torna-se necessário à existência de ferramentas que possibilitem a apresentação dos algoritmos e mecanismos de um sistema operacional (SO) de forma a permitir a interação do acadêmico com o objeto de estudo.

O objetivo deste trabalho foi desenvolver um simulador para demonstrar não a totalidade das funcionalidades de um sistema operacional, mas sim concentrar-se nas áreas de escalonamento de processos e gerência de memória. O sistema oferece uma interface amigável para os usuários, bem como as devidas funções de ajuda para facilitar o entendimento do mesmo, atendendo aos objetivos de uma ferramenta de apoio ao ensino.

1.1 OBJETIVO GERAL

Desenvolver uma ferramenta de apoio ao ensino de sistemas operacionais.

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos desta pesquisa são:

a) entender e aplicar os conceitos do funcionamento de um sistema operacional;
b) compreender e aplicar os conceitos de ferramentas de apoio ao ensino;
c) modelar e desenvolver o protótipo de um simulador;
d) desenvolver manual do simulador.

1.3 JUSTIFICATIVA

Devido a elevada carga teórica referente à disciplina de sistemas operacionais, os professores encontram dificuldades em fazer os alunos compreenderem os conceitos fundamentais para o aprendizado. Desta forma torna-se necessário a utilização de técnicas de ensino que estão além de aulas expositivas, como, por exemplo, atividades em laboratório com a utilização de simuladores. A motivação para o desenvolvimento do simulador surgiu na disciplina de Sistemas Operacionais II, onde era realizado um trabalho com a finalidade de implementar algumas funcionalidades de um escalonador, de forma a colocar em prática os conceitos apresentados.

A grande vantagem pedagógica oferecida pelo simulador é proporcionar um ambiente de ensino mixto, onde aulas expositivas tradicionais e simulações podem ser combinadas permitindo a experimentação das teorias apresentadas em sala de aula.

Outra questão que motivou esta pesquisa foi o fato de que alguns simuladores de sistemas operacionais, não possuírem o código fonte disponível na Internet para estudo. Existe também o fato de alguns simuladores serem muito

complicados de usar e até mesmo de colocá-los em funcionamento. Então desta forma, o desenvolvimento de um simulador tornou-se interessante pela possibilidade de criar um software que seja de código livre, servindo como base para projetos futuros, e que possa fundamentalmente ser utilizado pelos acadêmicos da universidade (UNESC).

Desta forma, o objetivo principal desta pesquisa foi desenvolver um software de simulação que auxilie no ensino da disciplina de sistemas operacionais. Com ele, o estudante poderá visualizar conceitos fundamentais como controle de processos, gerência de memória, de forma clara e objetiva. Além disso, será possível interagir com o software, definindo as configurações da simulação, permitindo ao acadêmico observar alguns detalhes em um sistema operacional.

1.4 ESTRUTURA DO TRABALHO

A pesquisa está estruturada em 6 capítulos distribuídos da seguinte maneira:

O capítulo 1 é a introdução, onde é descrito o contexto do trabalho, bem como o objetivo geral, os objetivos específicos, a justificativa e a estrutura do trabalho.

O capítulo 2 aborda a questões educacionais da pesquisa, onde são descritos dois modelos de ensino e o papel da informática na educação. Posteriormente, o capítulo define um software educacional, demonstrando suas características e classificando-as.

O capítulo 3 é um estudo sobre sistemas operacionais, onde é feita uma descrição sobre o que ele é, e por quais componentes eles são formados. Em seguida é explicado o que são processos e qual o seu papel dentro do sistema. Posteriormente são abordados gerência de processador e de memória.

No capítulo 4 cita-se e comenta-se a cerca de alguns trabalhos que estão relacionados com a pesquisa.

O capítulo 5 traz o desenvolvimento da ferramenta SOLAB, objeto de estudo desta pesquisa. São descritas os softwares utilizados para o desenvolvimento, as etapas do projeto e a modelagem do sistema. Em seguida o software é descrito em todas as suas funcionalidades e posteriormente são apresentados os resultados obtidos.

O último capítulo traz a conclusão e os possíveis trabalhos futuros.

2 CONSIDERAÇÕES EDUCACIONAIS

Um software de simulação oferece a oportunidade ao acadêmico de interagir com os conhecimentos que lhe são apresentados. Porém de acordo com Maia (2001) um software não pode ser um fim em si mesmo, ele deve estar inserido dentro de um contexto pedagógico, sendo utilizado como um meio para ampliar as possibilidades efetivas de ensino.

O objetivo deste capítulo é apresentar as considerações educacionais que são o suporte do trabalho desenvolvido. Serão apresentados os modelos de ensino, o papel da informática na educação, avaliação de software educacional e os tipos de software educacionais.

2.1 TEORIAS DE APRENDIZAGEM

De acordo com Maia (2001) as duas teorias de aprendizagem mais utilizadas são a tradicional e a construtivista. Compreender estas duas teorias de ensino é fundamental para entender as vantagens pedagógicas adicionadas em um software de simulação.

Na teoria tradicional o foco é centrado no professor. Ele é responsável por passar seus conhecimentos seguindo um programa de curso bem definido, geralmente com base em uma bibliografia específica. Os alunos são avaliados por meio de notas e critérios definidos pelo professor (MERGEL, 1998).

Na teoria construtivista o foco é centrado no aluno. O professor neste modelo age como mediador do conhecimento, motivando nos alunos o espírito de investigação crítico, participativo e cooperativo (LA TAILLE, 1992).

Para Maia (2001) a idéia do simulador é utilizar a teoria construtivista, para incentivar o aluno a explorar as diversas situações e aprender com seus erros. Mesmo o software não apresentando a possibilidade de construção colaborativa do conhecimento, outras características o qualificam como tal:

a) proporciona múltiplas representações da realidade;
b) enfatiza a construção do conhecimento;
c) oferece um ambiente de aprendizado que simula a realidade, utilizando estudos de casos;
d) favorece o pensamento reflexivo em função da experiência com o ambiente de simulação;
e) facilita a identificação, definição e solução de problemas;
f) permite ao aluno um controle do experimento, garantindo uma evolução natural da complexidade das simulações.

Ainda segundo Maia (2001), a grande vantagem do software educacional é unir o ensino tradicional utilizado em sala de aula com a teoria construtivista, que permite ao acadêmico a construção do seu conhecimento.

2.2 INFORMÁTICA NA EDUCAÇÃO

Atualmente a informática vem sendo utilizada diretamente na educação. Sua ação no meio social e sua utilização como instrumento de aprendizagem aumenta a cada dia. Por isso estão ocorrendo mudanças estruturais e funcionais para adequar o processo educacional a esta nova realidade. (LOPES, 2006).

De acordo com Chaves (1985) muitas pessoas não concordam com o potencial educacional do computador, pois pensam que sua única função seria a de

auxiliar o professor na colaboração dos conteúdos a serem ministrados. Mas essa não é a única função do computador. Ele pode ser utilizado como ferramenta do processo de ensino aprendizagem, contribuindo para o desenvolvimento intelectual do acadêmico.

Para Jonassen (1996) aprendizagem e tecnologia relacionam-se de quatro maneiras:

a) aprender a partir da tecnologia. O conhecimento é apresentado ao aluno por meio da tecnologia, cabendo a ele receber esse conhecimento como se ele fosse apresentando pelo professor. Como por exemplo, pode ser citado o ensino assistido por computador, os filmes educativos, tutoriais, aplicações de repetição e prática e o ensino programado;
b) aprender acerca da tecnologia. Neste caso a tecnologia constitui um objeto de aprendizagem, ou seja, ela contém os conhecimentos e competências necessários para que professores e acadêmicos possam utilizá-la;
c) aprender por meio da tecnologia. Nesta categoria estão incluídos os softwares que permitem ao aluno aprender ensinando o computador, como por exemplo, programando o computador utilizando linguagens como BASIC[1] ou o LOGO[2];
d) aprender com a tecnologia. Neste caso, as ferramentas tecnológicas apóiam o acadêmico no processo de reflexão e construção do conhecimento. Desta forma, o mais importante não é a tecnologia em si, mas como ela é aplicada no aprendizado do acadêmico. A ferramenta desenvolvida na pesquisa se enquadra nesta categoria.

[1] Linguagem de programação criada com fins didáticos.
[2] Linguagem de programação utilizada como ferramenta de apoio ao ensino.

Para Marques (1997), o computador representa uma nova forma de socializar o conhecimento, pois a partir dos dados recebidos do acadêmico, o mesmo realiza uma análise e devolve novos elementos, como respostas, desenvolvendo-se assim uma espécie de diálogo entre homem e máquina, tornando o acadêmico e o computador interlocutores um do outro.

A grande vantagem do computador no processo de aprendizagem é a possibilidade de interação existente entre o acadêmico e o seu objeto de estudo.

Para Teixeira (1996), o computador é uma ferramenta de aprendizagem, pois pode ampliar competências, ajudando a desenvolver a capacidade de aprender a aprender e personalizando a transmissão de conhecimentos.

Segundo Haydat (1997), educar pela informática significa utilizar a tecnologia como recurso de auxílio no processo de ensino-aprendizagem. Na sua visão, é dada ênfase ao processo de construção do conhecimento do acadêmico, pois ele é ativo perante a máquina. Desta forma, o computador é usado para socialização e para o desenvolvimento das estruturas do pensamento.

Se observa que o uso do computador na educação tem sido alvo de debates e questionamento. O foco não é o instrumento em si, mas a maneira de empregá-lo, pois depende de uma concepção filosófica e de uma teoria de aprendizagem. Dependendo da concepção de educação adotada, o computador assumirá um determinado papel na relação entre o acadêmico, o conhecimento e o professor (HAYDAT, 1997).

Para implantação da informática no sistema educacional, é necessário que um conjunto de elementos sejam combinados para alcançar todo o potencial esperado na utilização dos recursos tecnológicos (OLIVEIRA apud COSTA, 2002, p. 42).

Segundo Costa (2002) os quatro componentes principais que formam o processo de implantação da informática na educação são:

a) computador ou hardware;

b) software educacional;

c) profissional de educação;

d) usuário.

Dentre os quatro elementos é importante destacar o software educacional. Existe muita discussão em torno do tema, no que diz respeito a como classificar um software educacional. Não existem métricas definidas, somente características que o software deve conter. Por este motivo é de fundamental importância conhecer essas características para avaliar um software educacional.

2.3 CARACTERÍSTICAS DE UM SOFTWARE EDUCACIONAL

De acordo com Campos (2002) o processo de avaliação de um software começa a partir dos seguintes questionamentos:

a) O software agrega valor à capacidade de aprender e de perguntar?

b) Possibilita simulações e interações?

c) Contribui para a análise e teste de hipóteses?

d) Apresenta indicadores para auto-avaliação?

e) Proporciona atividade independente?

Além disso, a avaliação compreende as seguintes etapas:

a) Caracterização do produto:

- Título do software
- Fabricante
- Forma de acesso

- Modalidade: Exercício-e-prática, Jogos, Simulações, Hipermídia, Tutores Inteligentes.
- Requisitos de hardware e de software

b) Avaliação pedagógica

- O software contém aspectos motivadores?
- Quais aspectos o software enfatiza?
  - Memorização de conteúdos;
  - Atenção/concentração;
  - Pensamento lógico;
  - Resolução de problemas;
- O software indica a faixa etária para qual é destinado?
- software direciona-se para atividades extra-classe?

c) Avaliação do software segundo critérios específicos

- A inteface do software são bem diagramadas?
- Os recursos de som são bem utilizados?
- Os recursos de animação são de boa qualidade?
- O tempo de resposta é satisfatório?
- Todas as opções estão implementadas?
- O help é adequado?
- O software adapta-se ao nível do usuário?
- O software dá tratamento aos erros do usuário?
- O software prevê o armazenamento das respostas dos usuários?
- O software é seguro e robusto, resistindo a ações hostis do usuário?

Campos (2002) também define os aspectos que devem ser considerados na avaliação da qualidade de um software, são eles:

1) possibilidade de correção de conteúdo (Alterabilidade);
2) facilidade de leitura da tela (Amenidade ao uso);
3) clareza dos comandos (Amenidade ao uso);
4) independência da linguagem (independência do ambiente);
5) adaptabilidade ao nível do usuário (Eficiência do processamento);
6) adequação do programa ao nível do usuário (Validabilidade);
7) facilidade de leitura do programa (Clareza);
8) ausência de erros no processamento do programa (Correção);
9) adequação do programa ás necessidades curriculares (Rentabilidade);
10) independência de hardware (Independência do ambiente);
11) existência de recursos motivacionais (Amenidade de uso);
12) previsão de atualizações (Validabilidade);
13) ausência de erros de conteúdo (Validabilidade);
14) possibilidade de inclusão de novos elementos (Alterabilidade);
15) resistência do programa a respostas inadequadas (Correção);
16) adequação do vocabulário (Amenidade ao uso);
17) fornecimento de feedback (Amenidade ao uso);
18) apresentação dos escores aos alunos (Validabilidade);
19) uso do tempo do equipamento (Rentabilidade);
20) integração do programa com outros recursos (Rentabilidade)
21) capacidade de armazenamento das respostas (Eficiência do processamento);
22) existência de tratamento de erro (Amenidade ao uso);

23) controle da seqüência do programa (Amenidade ao uso);

24) diagramação das telas (Amenidade ao uso);

25) tempo de resposta (Eficiência do processamento);

26) existência de ramificações para enfoques alternativos (Amenidade ao uso);

27) existência de mensagem de erro (Amenidade ao uso);

28) acesso a helps (Amenidade ao uso);

29) existência de manual do usuário (Amenidade ao uso);

30) uso de ilustrações (Amenidade ao uso);

31) uso de cor (Amenidade ao uso);

32) tempo de exposição de telas (Amenidade ao uso);

33) existência de geração randômica de atividades (Amenidade ao uso);

34) uso de recursos sonoros (Amenidade ao uso).

É importante ressaltar que a aplicação dos questionamentos acima depende primeiramente do tipo de software educacional que esta sob avaliação. De acordo com Maia (2001) a evolução dos softwares educacionais proporcionou o surgimento de vários grupos de softwares, como por exemplo, tutoriais, programas de exercício-e-prática, jogos e simulações.

Portanto para avaliar o software, é necessário conhecer quais os objetivos que ele atende e então fazer a avaliação de acordo. Para explicar melhor, pode-se comparar um jogo educacional com um programa tutorial. O programa tutorial é feito com passos previamente estabelecidos para ensinar a alguém um determinado conteúdo. Por outro lado, no jogo não existe a idéia de passos previamente estabelecidos, seu objetivo é permitir que o usuário decida que caminhos tomar e possa aprender com suas decisões. É possível observar que, embora os dois sejam softwares educacionais,

existem muitas diferenças que separam um do outro. Em decorrência dessas diferenças é necessário adaptar o processo de avaliação focando nas características do software avaliado.

2.3.1 Programas tutoriais

De acordo com Maia (2001) os programas tutoriais são softwares que buscam representar uma aula tradicional por meio do computador. A sua vantagem principal é a utilização de recursos multimídia como imagens, som, animações, entre outras.

Como um programa tutorial busca simular a vida real, ele também traz consigo os mesmos problemas encontrados em sala de aula. Por exemplo, em sala de aula um acadêmico ao encontrar dificuldade em entender um determinado conteúdo pode acabar optando em não questionar o professor e permanecer com a dúvida. No programa tutorial ocorre o mesmo problema, pois o acadêmico pode simplesmente desistir no meio do tutorial, não completando todas as etapas para o aprendizado. Segundo Maia (2001) uma tendência para o softwares tutoriais é implementar técnicas de inteligência artificial para analisar, avaliar e oferecer soluções para as dificuldades do acadêmico.

2.3.2 Programas exercício e prática

Segundo Maia (2001) os programas de exercício e prática são utilizados para revisar conceitos apresentados em sala de aula, por meio de recursos de multimídia. Estes softwares seguem o modelo de perguntas e repostas, como se o

professor aplicasse um exercício aos acadêmicos. Conforme o acadêmico vai respondendo as perguntas o software analisa os resultados e toma a decisão conforme foi previamente programado. Um software de exercício e prática pode ser apresentando também ao acadêmico no formato de um jogo, aumentando em muito as possibilidades de usar recursos multimídia.

2.3.3 Simuladores

De acordo com Maia (2001) as primeiras simulações foram desenvolvida para criar um ambiente seguro para atividades de risco. O objetivo era desenvolver modelos dinâmicos e simplificados do mundo real. Por se mostrar uma ferramenta de grande valor, ela foi aplicada posteriormente em áreas que exigiam grandes investimentos de tempo e dinheiro, como na indústria automobilística e aviação.

Para Valente (1995) a simulação oferece a possibilidade do acadêmico desenvolver hipóteses, testá-las, analisar resultados e refinar conceitos. Este tipo de software é muito mais complexo que um programa tutorial e também tem um potencial educacional muito maior, pois permite ao aluno, testar diferentes hipóteses, tendo assim um contato mais “real” com os conceitos envolvidos no problema em estudo. Existe ainda a situação onde o simulador permite um alto grau de intervenção do aluno. Neste caso o simulador é utilizado mais como ferramenta do que como um mediador de aprendizagem.

Um simulador pode ser uma alternativa educacional muito importante se aplicado no ensino de uma disciplina muito técnica e com elevada carga teórica. A primeira vantagem e o fato do conteúdo teórico estar sintetizado no simulador de maneira amigável, com uso de recursos de multimídia. A segunda vantagem é fazer o

acadêmico ter contato com o conhecimento, representado de forma mais “prática”, podendo interagir por meio dos controles disponíveis no simulador, tendo assim uma idéia de como realmente a teoria é aplicada.

Porém, segundo Valente (1995), construir boas simulações é um processo complicado que requer muitos recursos gráficos e sonoros, para se aproximar ao máximo da realidade. Outra dificuldade é que a simulação não cria a melhor situação para o aprendizado. Ela deve ser utilizada como complemento para apresentações formais, leituras e discussões em sala de aula. Se não for utilizada desta forma, o acadêmico pode pensar que o mundo real pode ser simplificado e controlado da mesma maneira que nos programas de simulação. Isto ocorre porque na simulação os detalhes muito específicos, que são apresentados na teoria, são removidos ou simplificados para tornar o processo mais fácil de ser compreendido. Portanto, é necessário criar condições para o acadêmico poder diferenciar a simulação do mundo real.

3 SISTEMAS OPERACIONAIS

Sistema operacional é um programa que permite aos usuários utilizar o hardware do computador. Seu objetivo é fornecer os mecanismos necessários para que o usuário possa executar programas, tornando o uso do sistema de computação conveniente e aproveitando os recursos do hardware de maneira eficiente (TANENBAUM; WOODHULL, 2000).

Basicamente, um sistema de computação pode ser dividido em quatro partes: o hardware, o sistema operacional, os programas aplicativos e os usuários (SILBERSCHATZ; GALVIN; GAGNE, 2000).

A estrutura básica do hardware engloba os seguintes componentes:

a) unidade central de processamento (CPU): responsável por todo o trabalho de controle e computação do sistema. Na CPU estão contidas todas as instruções que o sistema de computação pode executar, como por exemplo, funções matemáticas, de gerenciamento, de controle e para acesso aos dispositivos;
b) memória: responsável pelo armazenamento temporário das informações;
c) dispositivos de entrada e saída: Este grupo engloba todos os demais dispositivos do sistema de computações, como unidades de armazenamento permanente, dispositivos para comunicação em rede, unidades de CD, entre outros.

Um sistema operacional pode ser considerado um alocador de recursos, pois todos os componentes citados acima oferecem uma série de funcionalidades que podem ser utilizadas para resolver os problemas dos usuários. Assim, ele atua gerenciando

esses recursos, alocando quando necessário para usuários e programas executarem suas tarefas (SILBERSCHATZ; GALVIN; GAGNE, 2000).

Definir um sistema operacional como um alocador de recursos facilita o estudo de todas suas funcionalidades, pois engloba praticamente todas as áreas em que o sistema atua.

Outra forma para definir um sistema operacional é caracterizá-lo como um programa de controle. Essa definição enfatiza a necessidade do sistema operacional atuar controlando a execução dos programas de usuário para evitar erros e o uso indevido do computador. Preocupa-se especialmente com a operação e o controle de dispositivos de entrada e saída (SILBERSCHATZ; GALVIN; GAGNE, 2000).

Para Machado (1997) apesar da complexidade aparente, um sistema operacional é apenas um conjunto de rotinas executadas pelo processador, da mesma forma que qualquer outro programa. Seu principal objetivo é controlar o funcionamento de um sistema computacional, gerenciando seus recursos.

A questão principal, que inclusive é muito discutida, é que não existe uma definição exata para responder o que é um sistema operacional. Para obter uma boa definição é preciso estudar o sistema por completo e verificar todas as suas funcionalidades e componentes e como o sistema responde as mais variadas situações. A partir desse estudo é possível estabelecer uma definição para um sistema em específico e classificá-lo de acordo com as notações aceitas mundialmente.

De acordo com Maia (2001) o que diferencia um sistema operacional de uma aplicação convencional é a forma como ele é executado em função do tempo, pois suas rotinas não são encadeadas de forma seqüencial. A ordem de execução das rotinas de um sistema operacional obedece aos eventos assíncronos aos quais ele responde proveniente principalmente do contexto de hardware.

Para entender melhor como funciona um sistema operacional é necessário dividi-lo em partes ou componentes, agrupando por funcionalidade. De acordo com Silberschatz, Galvin e Gane (2000) um sistema operacional moderno pode ser dividido nos seguintes componentes:

a) gerência de processos: responsável por criar e excluir processos, fornecer mecanismos de sincronização, comunicação entre processos. Também é responsável por controlar as situações de concorrência que ocorrem quando os processos competem por recursos;
b) gerência da memória principal: responsável por manter registros das partes da memória que são utilizadas e por quem. Controlar quais processos deverão ser carregados na memória quando houver espaço disponível. E finalmente alocar e desalocar espaço na memória;
c) gerência de arquivos: responsável por criar e excluir arquivos e diretórios. Fornecer suporte a manipulação de arquivos e diretórios. Responsável também por mapear arquivos no armazenamento secundário e fazer *backup*[3] dos mesmos em meios de armazenamento não voláteis;
d) gerência do sistema de Entrada/Saída: responsável por gerenciar os *drivers*[4] para os dispositivos de hardware. Possui também uma interface genérica para novos *drivers*;
e) gerência de armazenamento secundário: responsável pela gerência dos dispositivos de armazenamento secundários, como os disco rígidos. Controla a utilização de espaço em disco, o espaçamento livre e o escalonamento das solicitações ao disco;

[3] Cópia de segurança de um arquivo.
[4] Software que permite ao sistema operacional controlar um dispositivo.

f) redes: componente responsável por toda a comunicação realizada por meio de dispositivos de rede. Gerencia o acesso aos dispositivos bem como os protocolos de comunicação utilizados;

g) sistema de proteção: define os mecanismos de proteção para controlar o acesso de programas, processos ou usuários aos recursos do sistema. Ele define quais controles serão impostos e como eles serão obedecidos;

h) interpretador de comandos: receber por receber os comandos passados ao sistema, interpretá-los e fazer as chamadas de sistema apropriadas.

Uma propriedade fundamental presente nos sistemas operacionais modernos é a capacidade de permitir que diversos programas sejam executados de forma concorrente. Para implementar esta funcionalidade o sistema operacional utiliza uma abstração denominada processo, que representa um grupo de atividades que precisam ser executadas utilizando os recursos do sistema. Então cabe a ele gerenciar estes processos e garantir que os mesmos acessem todos os recursos disponíveis da maneira mais eficiente possível.

3.1 PROCESSOS

Um processo é basicamente definido como um programa em execução. Ele é constituído pelo código do programa, uma seção de dados onde são armazenadas as variáveis globais, uma pilha para dados temporários e o conteúdo dos registradores (SILBERSCHATZ; GALVIN; GAGNE, 2000).

Segundo Tanenbaum e Woodhull (2000) o conceito de processo fica mais fácil de compreender analisando um sistema operacional multitarefa. Nesses sistemas o processador é compartilhado entre os vários programas e usuários, mas cada usuário

tem a impressão de possuir o uso exclusivo ao processador. Para realizar esta tarefa, o sistema operacional organiza as informações referentes aos programas em execução em estruturas de dados complexas, estruturas estas que recebem o nome de processo.

Conforme Machado e Maia (1998) ele é composto basicamente por três elementos: contexto de hardware, contexto de software e espaço de endereçamento. Estes três elementos juntos mantêm as informações necessárias para a execução de um programa. A figura 1 ilustra a estrutura de um processo.

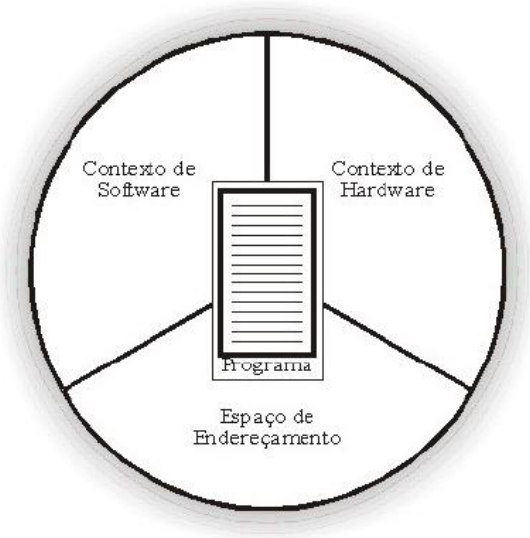


Figura 1. Estrutura de Processo
Fonte: MAIA (2001)

No contexto de hardware são armazenadas as informações referentes aos registradores do processador (registradores de dados, registradores de memória, registradores de entrada/saída, registradores de flag[5] e o principal que é o registrador de instrução que controla o fluxo de execução do programa).

As informações do processo referentes aos recursos oferecidos pelo sistema operacional são armazenadas no contexto de software. Estas informações incluem

[5] Registrador de sinalização do processador.

número de arquivos abertos, canais de entrada e saída, área de memória alocada, identificador do usuário corrente e privilégios.

O espaço de endereçamento é a área de memória definida para o processo e os dados utilizados por ele. Esta área de memória é de uso exclusivo, sendo que o sistema operacional garante que outros processos não têm acesso a ela.

Uma questão importante é a diferença entre um processo e um programa. De acordo com Silberschatz, Galvin e Gagne (2000) um programa é uma unidade passiva como o conteúdo de um arquivo armazenado no disco, enquanto um processo é uma unidade ativa que além do programa possui um contador de instruções é uma série de recursos associados.

Como os processos são independentes um dos outros, o sistema operacional consegue compartilhar o uso do processador entre eles. Desta forma, em um período de tempo muito curto, vários processos diferentes podem usar o processador. Assim o usuário do sistema tem a impressão de que seus programas estão executando todos ao mesmo tempo. Esta característica também é conhecida com pseudoparalelismo. O sistema operacional possui uma série de mecanismos para implementar o pseudoparalelismo, como por exemplo, atribuir estados para os processos. Este recurso será explicado a seguir.

3.2 ESTADOS DO PROCESSO

De acordo com Tanenbaum e Woodhull (2000) num sistema multitarefa o tempo de processamento é dividido entre os diversos processos. Para que isso seja possível o sistema operacional separa os processos por estados. Basicamente existem três estados de processos, que serão listados a seguir:

a) executando: quando está utilizando o processador. Em sistemas com somente um único processador, somente um processo pode ser executado num dado momento;
b) pronto: esta aguardando em uma fila de processos esperando sua oportunidade para utilizar o processador. O sistema operacional determina qual a ordem dos processos para execução. Este mecanismo é chamado de escalonamento;
c) bloqueado: está parado, aguardando por um evento externo, como uma operação de entrada/saída pendente.

Os dois primeiros estados são semelhantes, pois o processo está pronto para executar, mas no segundo a CPU não está disponível e o processo fica aguardando. O terceiro estado é diferente, pois o processo está aguardando um evento externo e assim não pode utilizar a CPU.

De acordo com Machado e Maia (1998) o estado de um processo é modificado de acordo com eventos voluntários gerados por ele, ou por eventos involuntários gerados pelo sistema operacional. Ainda segundo os mesmos autores as trocas de estado de um processo ocorrem da seguinte forma:

a) pronto para executando: após criado, o sistema coloca o processo em uma lista de prontos onde ele aguarda para ser executado. Cada sistema operacional define seus mecanismos e algoritmos para escolher qual processo será executado;
b) executando para bloqueado: ocorre quando o processo gera algum evento que necessita de um recursos externo, como uma operação de Entrada/Saída, ou por intervenção do sistema operacional, quando por

exemplo, a execução de um processo é suspendida por um período de tempo;

c) bloqueado para pronto: quando a solicitação externa do processo é atendida, o processo passa para o estado de pronto. Ele sempre passa para o estado de pronto depois de bloqueado. Ele não pode passar para o estado executando diretamente;

d) executando para pronto: esta mudança ocorre por eventos gerados pelo sistema operacional, como o término do tempo que o processo possui para executar. Assim o processo volta para a fila de prontos e aguarda outra oportunidade.

Desta forma utilizando o mecanismo de mudança de estados o sistema operacional consegue compartilhar os recursos disponíveis.

3.3 GERÊNCIA DE PROCESSADOR

A possibilidade de compartilhar a CPU foi o ponto de partida para o desenvolvimento de sistemas multiprogramáveis. Assim cada sistema operacional passou a utilizar um critério para determinar a ordem dos processos para execução entre os vários processos que concorrem pela utilização do processador (MACHADO; MAIA, 1998).

De acordo com Silberschatz, Galvin e Gagne (2000) o objetivo da multiprogramação é manter sempre algum processo em execução para maximizar a utilização da CPU. A idéia e relativamente simples, um processo é executado até que um evento externo ocorra e o processo seja bloqueado. Em um sistema de computação simples a CPU ficaria ociosa, desperdiçando todo o tempo de espera sem realizar

nenhum processamento. Por meio da multiprogramação esse problema é resolvido, pois quando um processo fica bloqueado, o sistema operacional coloca outro no lugar para utilizar a CPU. Este ciclo se repete sempre que um processo é bloqueado.

A troca entre os processos executado pelo sistema operacional é chamado de escalonamento. Dentro do sistema operacional ele é executado por um componente que recebe o nome de escalonador.

Segundo Maia (2001) o escalonador deve manter a CPU ocupada a maior parte do tempo, balanceando sua utilização e oferecendo tempos de resposta razoáveis para os diversos usuários do sistema. Além disso, os processos não podem ficar esperando indefinidamente pelo processador.

3.4 ESCALONAMENTO NÃO-PREEMPTIVO

Segundo Tanenbaum e Woodhull (2000) nos primeiros sistemas multiprogramáveis o processamento ocorria em lotes, ou seja, vários processos eram enfileirados e ficavam aguardando o processador. Nesses ambientes o tipo de escalonamento mais utilizado era o não preemptivo, ou seja, um processo recebia o processador e o utilizava até completar todo o trabalho.

A implementação desse modelo de escalonamento é relativamente simples. O sistema operacional mantém uma fila, onde os processos são armazenados por ordem de chegada. Quando um assume o processador, ele o utiliza até realizar todo o processamento necessário. Quando ele passa para o estado de bloqueado devido a solicitação de um recurso externo, ele é colocado em uma fila separada até o momento em que possa voltar para a fila principal. Este modelo também é chamado de FIFO (First In, First Out) (MACHADO; MAIA, 1998).

Este modelo de escalonamento é mais adequado para ambientes onde a preempção pode ser prejudicial para o processo que esta utilizando o recurso. Como por exemplo, uma fila de impressão onde é necessário aguardar a conclusão de um documento para imprimir o próximo, ou na gravação de um arquivo, onde a interrupção pode causar a corrupção dos dados. Assim, de acordo com Tanenbaum e Woodhull (2000), algoritmos de escalonamento não-preemptivos deve sem utilizados somente em casos específicos, como os citados acima, não sendo adequados para ambientes com muita concorrência entre processos, onde o compartilhamento eficiente de recursos é fundamental.

3.5 ESCALONAMENTO PREEMPTIVO

O modelo de escalonamento preemptivo é caracterizado pela possibilidade do sistema operacional interromper um processo em execução para que outro utilize o processador. O escalonamento preemptivo permite ao sistema priorizar processos mais importantes, proporcionando melhores tempos de resposta em sistemas de tempo compartilhado. Outra vantagem é o compartilhamento mais justo do processador (MACHADO; MAIA, 1998).

Segundo Silberschatz, Galvin e Gagne (2000) nesse sistema o escalonamento de CPU pode ocorrer em quatro situações:

a) quando um processo passa do estado em execução para o estado em espera (por exemplo, um pedido de entrada/saída ou chamada de espera para término de um dos processos filhos);
b) quando um processo passa do estado em execução para o estado de pronto (exemplo, quando ocorre uma interrupção);

c) quando um processo passa do estado de espera para o estado de pronto (por exemplo, conclusão de entrada/saída);

d) quando um processo termina.

O modelo preemptivo afeta diretamente os casos b e c. De acordo com as regras de escalonamento implementadas pelo sistema operacional, quando um processo se enquadra nestes casos o escalonador do sistema operacional valida as regras efetuando ou não a preempção do processo que esta utilizando o processador.

O problema principal do modelo preemptivo é a sobrecarga gerada no processador pela troca de processos. Pois quando um processo é interrompido para que outro assuma, o processador terá um trabalho adicional para salvar todas as informações do processo atual e substituir pelas informações do próximo. Além disso, podem ocorrer situações em que existe dependência de dados entre processos, sendo que a preempção pode acarretar em perda de desempenho (SILBERSCHATZ; GALVIN; GAGNE, 2000).

3.6 CRITÉRIOS DE ESCALONAMENTO

Existem diferentes algoritmos de escalonamento que se adéquam melhor a determinadas situações. Na escolha do melhor algoritmo devem ser analisadas as suas diferentes propriedades, pois o mesmo é um componente crítico para o desempenho do sistema.

Segundo Tanenbaum e Woodhull (2000), os critérios mais utilizados são:

a) imparcialidade: o sistema operacional deve dividir o tempo de CPU de maneira justa para todos;

b) eficiência: a CPU deve ser mantida ocupada 100% do tempo enquanto existirem processos aguardando na fila;

c) tempo de resposta: como o usuário do sistema deve ter a impressão de que todos seus programa estão executando ao mesmo tempo, o tempo de resposta decorrido entre uma solicitação ser iniciada até ser atendida deve ser o menor possível;

d) turnaround: refere-se ao tempo que um processo leva para ser executado por completo;

e) throughput: refere-se ao numero de processos executados em um intervalo de tempo. Então quanto maior o throughput, maior o número de processos executados por fatia de tempo.

De acordo com Silberschatz, Galvin e Gagne (2000), é importante maximizar a eficiência e o throughput do sistema e minimizar o tempo de resposta e o turnaround. Em muitos casos é otimizado para estabelecer uma média. Porém esta escolha de quais critérios são mais importantes, depende do tipo de sistema operacional que está sendo analisado. Por exemplo, para sistemas operacionais interativos com tempo compartilhado, o melhor é diminuir a variação do tempo de resposta, para proporcionar bons serviços para todos os usuários.

Desta forma foram desenvolvidos vários algoritmos de escalonamento, cada um focando em critérios específicos. Portanto cada sistema operacional utiliza um ou alguns algoritmos combinados, dependendo do tipo de serviço ao qual ele se propõe a prover.

3.7 ALGORÍTMOS DE ESCALONAMENTO

Existem vários algoritmos de escalonamento. Os principais são listados a seguir:

3.7.1 Escalonamento First-In, First-Out (FIFO)

O algoritmo de escalonamento First-In, First-Out é o mais simples. Nesse modelo, o processo que solicitar o processador primeiro, o recebe primeiro. Esse algoritmo é implementando basicamente utilizando uma fila de processos simples. Cada novo processo é alocado no final da fila. Quando um processo libera o processador, o primeiro processo na fila o assume (SILBERSCHATZ; GALVIN; GAGNE, 2000).

Este algoritmo é problemático, pois não promove a distribuição uniforme dos recursos dos sistemas, já que os processos não são previamente avaliados. Desta forma o sistema pode acumular em um momento processos que necessitam muito dos dispositivos de entrada/saída e em outro momento processos que necessitam muito da CPU (MACHADO; MAIA, 1998).

3.7.2 Escalonamento Job Mais Curto Primeiro

O algoritmo de escalonamento Job mais Curto Primeiro (SJF) é baseado em surtos de utilização da CPU. O objetivo do algoritmo é verificar qual processo na fila de execução possui o menor surto de execução e assim atribuir a CPU para este processo (SILBERSCHATZ; GALVIN; GAGNE, 2000).

Este modelo de escalonamento é muito eficiente, pois diminui o tempo médio de espera dos processos. Assim, quando um processo curto recebe o processador antes de um processo longo, o tempo de espera do processo curto diminui mais do que aumenta o tempo de espera do processo longo. Desta forma, o tempo de espera médio diminui (SILBERSCHATZ; GALVIN; GAGNE, 2000).

Para sistemas interativos, onde os processos são adicionados a fila durante a execução do sistema, torna-se difícil para o algoritmo SJF determinar quais processos possuem os menores surtos de CPU. Para tornar viável a utilização desse algoritmo em sistemas interativos, se utiliza de técnicas para prever o próximo surto de CPU de um processo, baseado nos surtos anteriores (TANENBAUM; WOODHULL, 2000).

3.7.3 Escalonamento por Prioridade

Em sistemas interativos, muitos processos podem ser executados de forma concorrente. Alguns processos são mais importantes e precisam ter preferência na execução. Assim, utilizando como base o algoritmo SJF e desenvolvendo um sistema de prioridades, surgiu o algoritmo de escalonamento por prioridades. Sua função é priorizar a execução de processos com prioridade. Este algoritmo também atribui prioridades mais elevadas para processos com menores surtos de CPU (SILBERSCHATZ; GALVIN; GAGNE, 2000).

As prioridades dos processos podem ser internas ou externas. As prioridades internas são atribuídas de acordo com a análise do processo, avaliando seus requisitos de CPU e memória, o número de arquivos abertos e os surtos de CPU do processo. As prioridades externas são definidas por características que não fazem parte do sistema

operacional, como a importância do processo ou as necessidades do usuário. (SILBERSCHATZ; GALVIN; GAGNE, 2000).

Uma desvantagem muito significativa desse algoritmo é a possibilidade de processos ficarem bloqueados indefinidamente. Este problema ocorre, pois processos de baixa prioridade, em sistemas que executam grande quantidade de processos, podem ficar sempre esperando no fim da fila e nunca receber a CPU. Para resolver este problema é utilizada a técnica de envelhecimento (*aging*[6]). Este técnica aumenta gradualmente a prioridade dos processos que ficam esperando por muito tempo. Então depois de um determinado período de tempo, o processo passa a ter uma prioridade alta, até o momento em que recebe a CPU. (TANENBAUM; WOODHULL, 2000).

3.7.4 Escalonamento Round-Robin

O algoritmo de escalonamento Round-Robin foi desenvolvido especialmente para sistemas de tempo compartilhado. Para cada processo o algoritmo atribui um intervalo de tempo chamado de *quantum*[7], o qual é utilizado para que o processo execute seu processamento. Quando o processo utiliza todo o quantum e não termina, o escalonador executa a preempção da CPU e ela é dada a outro processo. Caso o processo não utilize todo o tempo destinado a ele, então é feita a troca de processos naturalmente (TANENBAUM; WOODHULL, 2000).

A fila de processo utilizada é do tipo circular, onde o processo que acabou de utilizar o processador retorna para o fim da fila e espera novamente por sua vez.

[6] Técnica de envelhecimento, onde a prioridade do processo aumenta conforme o tempo de espera.
[7] Fatia de tempo utilizada pelo sistema operacional para controlar o tempo limite de uso do processador por um processo.

Assim a implementação deste algoritmo é muito simples, pois não envolve nenhuma lógica ou estrutura de dados complexa.

O ponto mais importante para utilização desse algoritmo é a escolha do tamanho do quantum. Caso seja estabelecido um valor muito pequeno, o sistema vai desperdiçar muito tempo de CPU efetuando a troca de processo. Mas se for estipulado muito grande, todos os processos terão que esperar muito na fila e, além disso, o tempo médio de resposta do sistema vai aumentar o que não é adequado para um sistema compartilhado. Desta forma, o ideal e avaliar quais os requisitos do sistema e a que tipos de processos ele vai servir, procurando estabelecer assim, um valor que fique na média (SILBERSCHATZ; GALVIN; GAGNE, 2000).

3.7.5 Escalonamento por Múltiplas Filas

Existem situações onde é possível classificar os processos em diferentes grupos. Nessas situações é possível utilizar o sistema de escalonamento por múltiplas filas. O objetivo deste algoritmo é classificar os processos em grupos pré-definidos e atribuí-los a uma fila específica. Essas filas são definidas pelo projetista do sistema operacional, sendo que cada uma delas possui características especificas (SILBERSCHATZ; GALVIN; GAGNE, 2000).

A principal vantagem deste sistema é a possibilidade de utilizar um algoritmo de escalonamento para cada fila. Assim, de acordo com as características do grupo de processos é definido o algoritmo de escalonamento. Também é possível definir diferentes prioridades para cada fila (MACHADO; MAIA, 1998).

Para evitar que processos fiquem aguardando por muito tempo em filas de baixa prioridade, utiliza-se um sistema de realimentação onde em determinados

intervalos de tempo processos que não receberam a CPU são realocados para uma fila superior (SILBERSCHATZ; GALVIN; GAGNE, 2000).

3.8 GERÊNCIA DE MEMÓRIA

De acordo com Machado e Maia (1998) historicamente, a memória principal foi considerada como um recurso escasso. Portanto os projetistas de software sempre se preocuparam em desenvolver sistemas operacionais que não ocupem muito espaço de memória e ao mesmo tempo aperfeiçoem sua utilização. Atualmente, mesmo com a redução de custos e o aumento da capacidade, o gerenciamento de memória continua sendo um fator muito importante no projeto de sistemas operacionais.

A parte do sistema operacional que gerência a hierarquia da memória é chamada gerenciador de memória. Seu objetivo é controlar a memória em uso e livre, alocar memória para processos que solicitarem e liberar quando ela não for mais necessária. O gerenciador de memória também é responsável pela troca entre memória principal e secundária, geralmente o disco rígido, quando o espaço disponível não for suficiente (TANENBAUM; WOODHULL, 2000).

A memória do computador é formada por uma lista muito grande de bytes, dispostos seqüencialmente, cada um com um endereço. A CPU busca as instruções do programa conforme sua execução ocorre. As instruções executadas podem também acessar e armazenar dados em outras partes da memória. Para o hardware do computador a ordem pela qual a memória é acessada não tem nenhum sentido lógico. O controlador de memória, responsável por atender as solicitações do software, simplesmente lê e grava dados, de acordo com os endereços que são informados. Toda a ordem pela qual os pedaços de memória serão lidos ou escritos, bem como os endereços

solicitados, é de responsabilidade do gerenciador de memória (SILBERSCHATZ; GALVIN; GAGNE, 2000).

3.9 MAPEAMENTO DE ENDEREÇOS

Para ser executado um programa necessita estar presente na memória principal. Para que isto ocorra, o sistema operacional aloca o espaço necessário e carrega o programa do armazenamento secundário, geralmente o disco rígido. Cada vez que um programa é carregado, ele é alocado na memória em um endereço diferente, ou seja, não existe uma posição específica. Desta forma é necessária uma rotina de software para ajustar os endereços quando forem solicitados (SILBERSCHATZ; GALVIN; GAGNE, 2000).

Nos primeiros programas de computador desenvolvidos, os endereços de memória presentes no arquivo executável, correspondiam ao endereço real na memória do computador. Desta forma quando dois softwares estavam programados para utilizar a mesma posição de memória, somente um podia ser executado em um determinado momento. A evolução dos compiladores permitiu desenvolver códigos para programas que não dependessem de uma posição específica. Para esta característica ser possível, todos os endereços de memória existentes no programa são relativos ao início do código e não aos endereços reais. Assim quando o programa é carregado para a memória, o sistema operacional ajusta todos os valores de acordo com o endereço inicial (MACHADO; MAIA, 1998).

De acordo com Silberschatz, Galvin e Gagne (2000) a associação dos endereços simbólicos com os endereços reais pode ser executada nas seguintes situações:

a) em tempo de compilação: Quando o endereço de memória onde o processo será alocado for previamente estabelecido, então no processo de compilação é gerado o código absoluto. Nesse caso, para fazer o programa poder ser executado em uma área diferente de memória é preciso recompilar o código. Esta técnica é utilizada nos programas de MS-DOS [8]com a extensão .COM[9];

b) em tempo de carga: Se durante a compilação o endereço de carga não estiver determinado, o compilador deverá gerar um código relocável. Desta forma, a associação de endereços acontece no momento de carregar o programa para a memória. Caso o endereço de inicio mudar, basta recarregar o código do programa com o novo valor;

c) em tempo de execução: Quando o processo puder ser movido durante sua execução de um segmento para outro, a associação de endereços é retardada para o momento da execução. Para utilizar esta técnica o sistema de computação necessita de um hardware especial. A maioria dos sistemas operacionais utiliza esse método.

3.10 SWAPPING

As técnicas de multiprogramação e gerência de memória permitem que o sistema de computação execute uma quantidade maior de processos de maneira concorrente. Porém nem sempre o sistema possui memória disponível para armazenar todos os processos. Desta forma, foi desenvolvida a técnica de swapping, que tem por

[8] Sistema Operacional da Microsoft. Seu nome significa Sistema Operacional em Disco.
[9] Formato de arquivos executáveis para o sistema operacional MS-DOS

objetivo armazenar processos no disco rígido, quando não existe mais espaço disponível na memória principal (MACHADO; MAIA, 1998).

O funcionamento desse sistema é simples, quando um processo termina de usar a CPU, o escalonador executa uma operação denominada *swap out*[10], onde o processo é armazenado na unidade secundária, como um disco rígido. Então a memória que o processo estava utilizando é liberada e o escalonador executa uma operação de swap in[11], onde outro processo é carregado para a memória que foi liberada (SILBERSCHATZ; GALVIN; GAGNE, 2000).

Como os processos possuem tamanhos diferentes e são posicionados em posições diferentes da memória, não é possível que um processo que foi armazenado no disco seja carregado novamente na mesma posição de memória que estava anteriormente. Para que o processo continue executando normalmente, a associação de endereços simbólicos em tempo de execução deve ser utilizada. Quando o processo volta à memória, o gerenciador de memória atualiza o espaço de realocação e a nova posição do processo é mapeada. Desta forma o processo continua utilizando os mesmos endereços lógicos antes e depois da operação de swap. Assim a operação fica transparente para o processo que não precisa implementar nenhuma função extra para reajustar os endereços (MACHADO; MAIA, 1998).

Um processo que sofre a operação de swap out[6] não pode ter nenhuma operação pendente. Se, por exemplo, o processo estiver aguardando por alguma operação de entrada/saída e for armazenado no disco, sendo que um novo processo vai ocupar seu lugar, no momento em que a operação terminar, o gerenciador do dispositivo de entrada/saída vai tentar acessar o endereço do processo para devolver os resultados,

[10] Rotina do sistema operacional que armazena um processo no armazenamento secundário.
[11] Rotina do sistema operacional que recupera um processo do armazenamento secundário.

gerando um erro grave que pode levar ao travamento do sistema (SILBERSCHATZ; GALVIN; GAGNE, 2000).

A área de armazenamento no dispositivo secundário utilizada para swap recebe um tratamento especial do sistema operacional. Geralmente ela é alocada fora do sistema de arquivos para tornar sua utilização o mais rápida possível. Em sistemas operacionais derivados do *Unix*[12], utiliza-se a técnica de criar uma partição específica no disco rígido para utilizar como espaço de swap (TANENBAUM; WOODHULL, 2000).

3.11 MEMÓRIA VIRTUAL

Memória virtual é uma técnica de gerência de memória que combina a memória principal com o armazenamento secundário, oferecendo aos programas um espaço de endereçamento muito maior que a memória principal real. Para implementar esta função o sistema operacional utiliza o conceito de endereço lógico, que representa um sistema de endereçamento paralelo ao endereço físico da memória principal. Desta forma por meio dos endereços lógicos, os programas utilizam o armazenamento secundário como se fosse a memória principal. Além disso, esta técnica permite que um número maior de processos compartilhe a memória, pois somente parte deles realmente permanece alocada. (MACHADO; MAIA, 1998).

A tradução dos endereços lógicos em físicos é feita por um hardware especial chamado de unidade de gerência de memória (*MMU*). Para isso, são utilizados alguns registradores de dados que guardam os valores de redirecionamento, que são associados aos endereços lógicos para assim gerar o endereço físico. Por exemplo, uma

[12] Sistema operacional de alta performance muito utilizado em servidores.

solicitação lógica ao endereço 10, pode acabar sendo redirecionada para o endereço 14010 na memória, devido à intervenção da *MMU* (SILBERSCHATZ; GALVIN; GAGNE, 2000).

O conceito de endereçamento lógico e físico é fundamental para o processo de gerência de memória.

3.12 PAGINAÇÃO

Paginação é uma forma de gerência onde a memória virtual do computador é dividida em páginas, ou seja, é dividida em pedaços de um tamanho pré-estabelecido. Como o espaço de memória virtual é dividido em partes, uma página pode ficar no armazenamento secundário ou na memória real. Quando está no armazenamento secundário é chamada de página virtual e quando esta na memória principal é chamada de página real ou *frame*[13] (MACHADO; MAIA, 1998).

Quando um processo vai ser executado o sistema operacional carrega o arquivo do programa alocando na memória principal dividido em páginas. Estas páginas não são necessariamente dispostas de forma seqüencial na memória, na verdade elas são dispostas de acordo com os pedaços de memória que estão disponíveis. Assim, quando um endereço de memória é solicitado pela CPU, é necessário haver uma forma para localizar em que página de memória ele está. Desta forma, todos os endereços gerados na CPU são divididos em duas partes: uma parte contém o número da página e a outra o deslocamento dentro dela. O número de página serve como índice para a tabela de páginas mantida pelo sistema operacional. Está tabela contém o endereço de início de

[13] Tem o significado de quadro, no sistema operacional representa um quadro ou pedaço da memória principal.

todas as páginas na memória física. Então o endereço obtido na tabela é somado ao número de deslocamento e assim é encontrado o endereço de memória solicitado (SILBERSCHATZ; GALVIN; GAGNE, 2000). A figura 2 ilustra o funcionamento desta técnica.

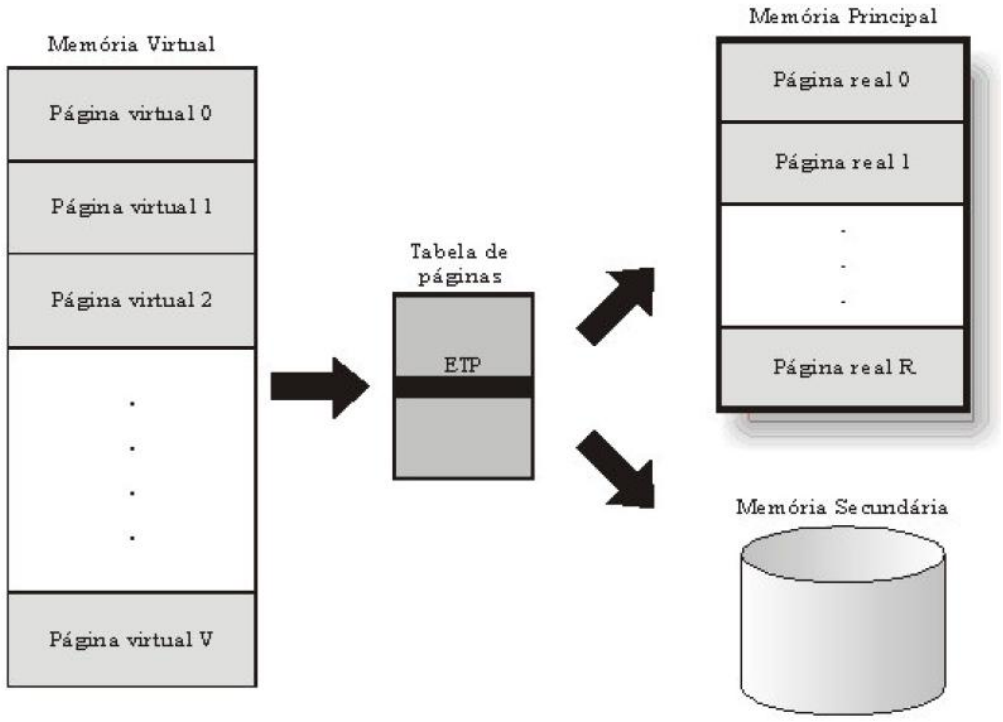


Figura 2. Memória virtual
Fonte: MAIA (2001)

Cada página utilizada no sistema possui um *bit* de informação que indica se ela está presente na memória principal ou no armazenamento secundário. Quando a CPU tenta acessar um endereço de uma página que não está na memória é gerado uma interrupção no sistema operacional chamada falha de página. Para tratar está falha, existe no sistema uma rotina que seleciona uma página pouco utilizada e a grava no armazenamento secundário para liberar espaço na memória. Então a página solicitada é carregada para o espaço que foi aberto, a tabela de páginas é atualizada para informar que a mesma esta disponível e a execução do programa continua do ponto onde foi interrompida (TANENBAUM; WOODHULL, 2000).

As páginas dos processos são transferidas da memória secundária para a principal somente quando solicitado. Esta técnica possibilita uma menor sobrecarga no sistema, pois páginas desnecessárias não são carregadas. Porém a desvantagem dessa abordagem esta no fato de no inicio da execução do sistema ocorrem muitas falhas de página, pois somente algumas páginas estão disponíveis. Alguns sistemas para solucionar este problema, carregam algumas páginas antecipadamente, tentando prever as necessidades do aplicativo. Porém, dependendo do processo, esta solução não traz ganho de desempenho nenhum e ainda causa sobrecarga no sistema operacional (MAIA, 2001).

4 TRABALHOS CORRELATOS

Durantes a pesquisa foram encontrados alguns softwares de apoio ao ensino que contribuíram para o desenvolvimento do SOLAB. Estes softwares são descritos a seguir:

a) **IMática** : é um plataforma de geometria dinâmica utilizada no apoio ao ensino de matemática e geometria. O objetivo da ferramenta é disponibilizar um novo conceito no ensino fundamental, introduzindo experimentação e visualização de propriedades matemáticas. A ferramenta é desenvolvida no instituto de matemática e estatística da Universidade de São Paulo (USP);

b) **SAE-Fra – Software de Apoio ao Ensino de Frações** : software com fins educacionais sobre frações, que tem como público alvo o ensino fundamental e visa tornar o ensino de frações, um pouco mais atrativo ao aluno. Desenvolvido pelo acadêmico Marcos Dias Fagundes na Pontifícia Universidade Católica do Rio Grande do Sul, o software une animação computacional com ensino de frações, buscando motivar o aluno, tornando mais fácil e interessante o aprendizado;

c) **SOSim**: é um simulador de sistema operacional desenvolvido pelo professor Luiz Paulo Maia, como parte de sua tese de mestrado no Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro (NCE/UFRJ). Seu objetivo é facilitar e melhorar as aulas de sistemas operacionais para alunos e professores. O simulador apresenta

os conceitos e mecanismos de um sistema operacional multitarefa, de forma simples e utilizando animações. Com ele é possível visualizar os conceitos de multiprogramação, processo e suas mudanças de estado, gerência de processador (escalonamento) e a gerência de memória virtual. O software oferece vários recursos e configurações, permitindo alterar o seu funcionamento. Assim, o aluno pode visualizar os conceitos técnicos apresentados em aula.

5 DESENVOLVIMENTO DO SIMULADOR SOLAB

Como já foi dito anteriormente a disciplina de Sistemas Operacionais trata conceitos complexos da área computacional. Estes conceitos são de difícil aprendizado, sendo que existem poucos modelos que aproximem a teoria da prática.

O objetivo de desenvolver um simulador é de utilizar um ambiente artificial para ilustrar os conceitos com técnicas de animação, permitindo que os alunos possam interagir com o ambiente para perceber como as teorias apresentadas se comportam na prática.

No sistema desenvolvido o aluno tem como selecionar o tipo de algoritmo de escalonamento, com características especificas. Também é possível criar processos customizados para verificar como eles irão se comportar. O simulador, além de apresentar visualmente o que está acontecendo, também apresenta mensagens dos eventos que estão acontecendo, para que o aluno possa acompanhar todas as situações.

O processo de desenvolvimento do SOLAB seguiu a abordagem clássica de desenvolvimento de software por meio das etapas de Levantamento de Requisitos, Análise, Projeto, Codificação e Testes.

Como resultado da etapa de Levantamento de Requisitos, os principais foram identificados como sendo:

a) simular a execução de processos;
b) permitir ao usuário criar os processos;
c) permitir que processos sejam criados em qualquer momento da simulação;
d) implementar as filas de processo do escalonador de maneira visual para acompanhamento do usuário;

e) oferecer ao usuário formas de visualizar informações de todos os processos criados bem como as mudanças que ocorrerem duramente a simulação;
f) permitir a escolha da política de escalonamento a ser utilizada;
g) permitir definir se o escalonador será preemptivo ou não;
h) oferecer uma forma de o usuário navegar por todos os processos e visualizar informações detalhadas de todos;
i) possibilitar que o usuário possa executar o simulador em modo passo a passo e automático, sendo que no modo automático a velocidade de execução possa ser controlada;
j) exibir informações detalhadas sobre o processo que estiver utilizando a CPU em um determinado momento;
k) exibir informações sobre o gerenciamento de memória do simulador, como número de páginas, tamanho da memória virtual;
l) mostrar como os as páginas de memória dos processos estão alocadas;
m) agrupar todos os eventos que ocorrem no simulador no formato de texto, de forma seqüencial, para o usuário acompanhar o que está acontecendo no simulador;

Ainda nesta etapa do processo de desenvolvimento, os requisitos identificados foram detalhados, servindo como base para a etapa seguinte de Análise, onde foram desenvolvidos diagramas que retratam o problema em questão, numa visão conceitual. O diagrama de classes da Figura 3 apresenta o modelo do SOLAB. São apresentadas somente as classes principais do simulador, o diagrama completo pode ser encontrado nos apêndices da pesquisa.

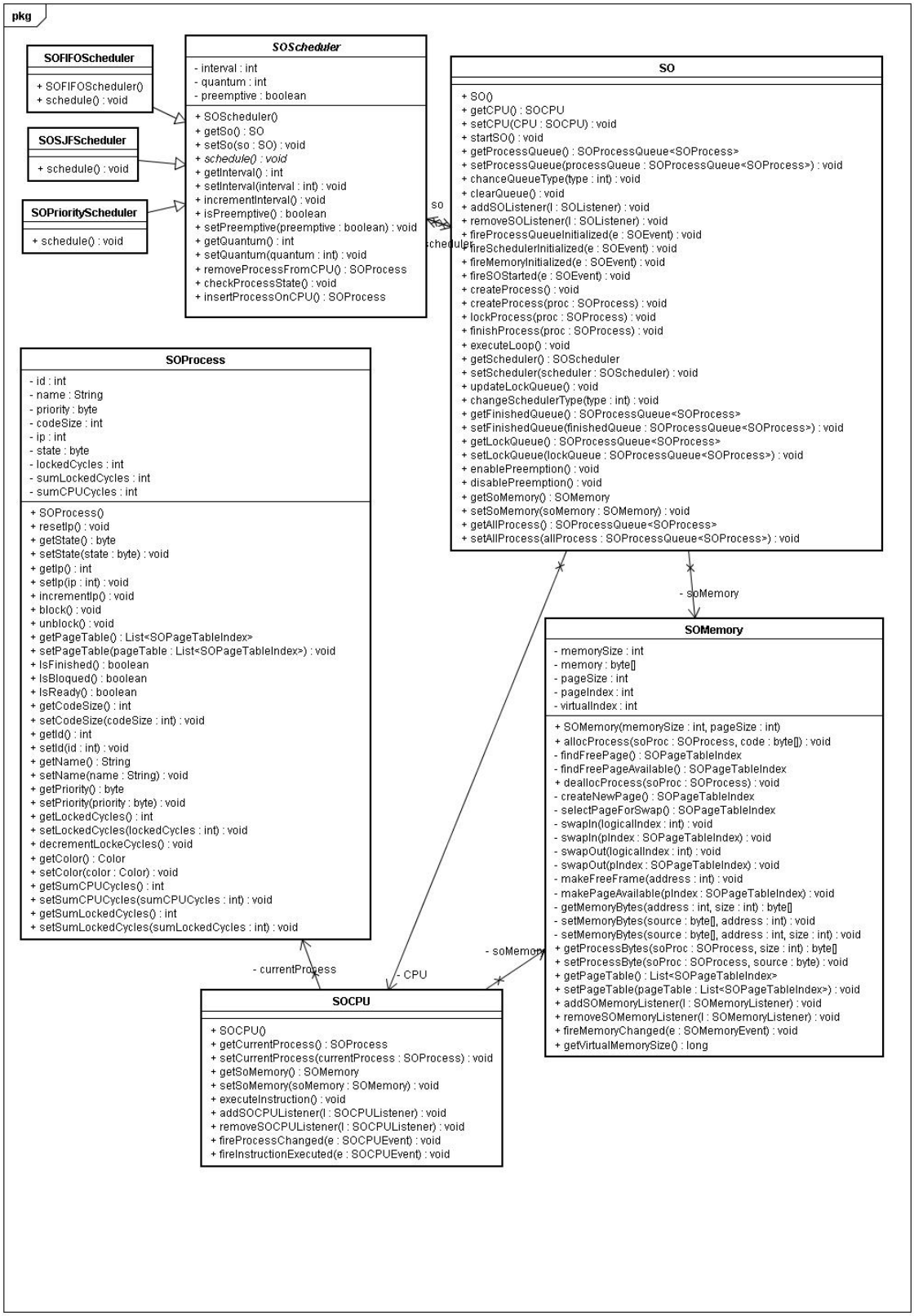


Figura 3. Diagrama de Classes do Simulador

A etapa de Projeto refere-se à transformação dos modelos em especificações que possam ser implementados, ou seja, representam o problema em uma visão computacional. Nesta etapa, além de refinar os diagramas criados anteriormente, foi definida a plataforma a ser utilizada no desenvolvimento do simulador. Nesse projeto uma diretriz seguida foi que somente softwares de código livre foram utilizados no desenvolvimento. O ambiente utilizado será descrito na seção 5.1.

Na etapa de codificação, os modelos definidos no diagrama UML, foram transformados em código da linguagem Java. Todo o código do simulador foi construído com a preocupação de tornar o sistema fácil de ser modificado, para que futuros acadêmicos que tenham interesse em implementar novas funcionalidades ao simulador, não encontrem dificuldades em fazê-lo. Além disso, todo o código está comentado para facilitar o entendimento.

A etapa de testes permite a verificação do correto funcionamento do simulador. Não foi utilizada nesse projeto, nenhuma ferramenta automatizada de testes. Os testes ocorreram no momento da codificação e no final do projeto. Realizaram-se várias simulações verificando que o software está funcionando corretamente.

5.1 RECURSOS UTILIZADOS NA FASE DE DESENVOLVIMENTO

O software foi desenvolvido em Java, pois a linguagem já oferece APIs nativas que facilitam a implementação da aplicação com recursos de animação (DEITEL; DEITEL, 2002). Foi utilizada a versão JDK 1.6 do Java, que é a mais recente disponível, pois foram introduzidas nessa versão muitas funcionalidade que facilitam o desenvolvimento de programas com interface gráfica, principalmente na questão de desempenho. Os programas Java compilados na versão 1.6 consomem menos recursos

do sistema, tem uma melhor integração com o sistema operacional e são mais rápidos do que os compilados na versão 1.5. Além disso, programas desenvolvidos em Java podem ser executados em diversos sistemas operacionais e diversas arquiteturas, assim é atendido um dos requisitos de um software educacional que é a independência de ambiente.

Para desenvolvimento foi utilizado o Netbeans 6, por se tratar de um ambiente de desenvolvimento de fácil usabilidade oferecendo muitos recursos como formatação automática de código, importação automática de bibliotecas, ajudas e complementos no código, que facilitaram o desenvolvimento de software. E também um ponto crucial na escolha dessa ferramenta, foi a facilidade que ela oferece no desenvolvimento de aplicações com interface gráfica. As novas versões do Netbeans oferecem um ambiente para desenvolvimento de interfaces para usuário totalmente visual, onde o programador pode arrastar e soltar os componentes e gerar suas telas sem precisar digitar nenhum código.

O desenho dos diagramas foi feito utilizando a ferramenta JUDE versão 5.2.1, que é gratuita e muito fácil de usar. Essa ferramenta possui uma ótima integração com as bibliotecas da linguagem Java.

5.2 SISTEMA DESENVOLVIDO

O sistema foi proposto para ser uma simulação parcial de um sistema operacional real. Somente as funções de escalonamento e gerência de memória foram implementadas, de acordo com o que foi definido na etapa de projeto. No software desenvolvido, o usuário pode simular o funcionamento de um escalonador e um gerenciador de memória. A Figura 4 mostra o caso de uso do simulador.

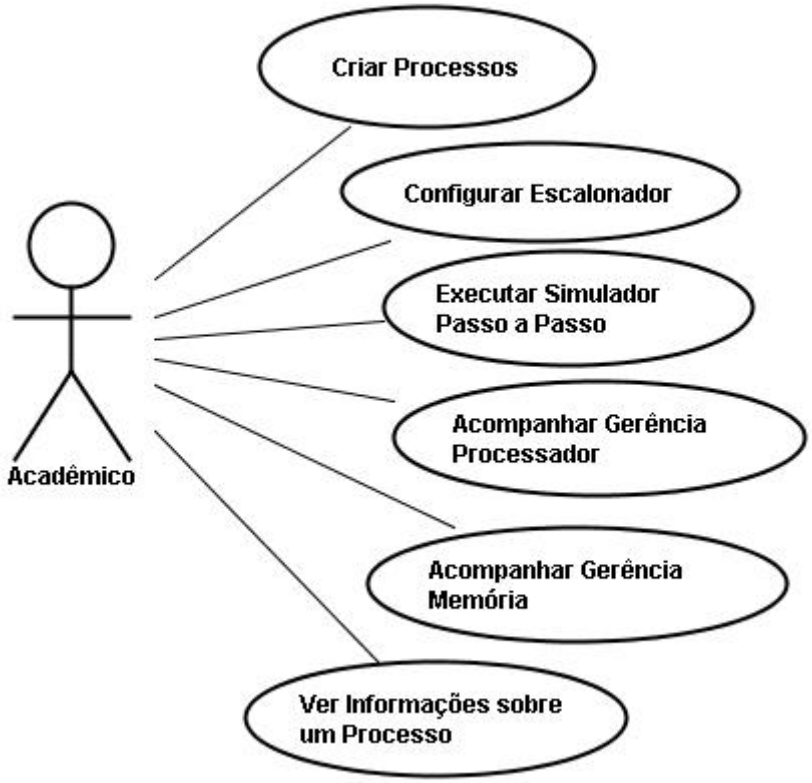


Figura 4. Diagrama de Casos de Uso do simulador

5.3 FUNCIONAMENTO DO SIMULADOR

A interface principal do sistema foi modelada de forma a facilitar o entendimento do usuário.

Os comandos principais do simulador foram disponibilizados na tela principal, agrupados por funções. A Figura 5 mostra a interface principal do sistema.

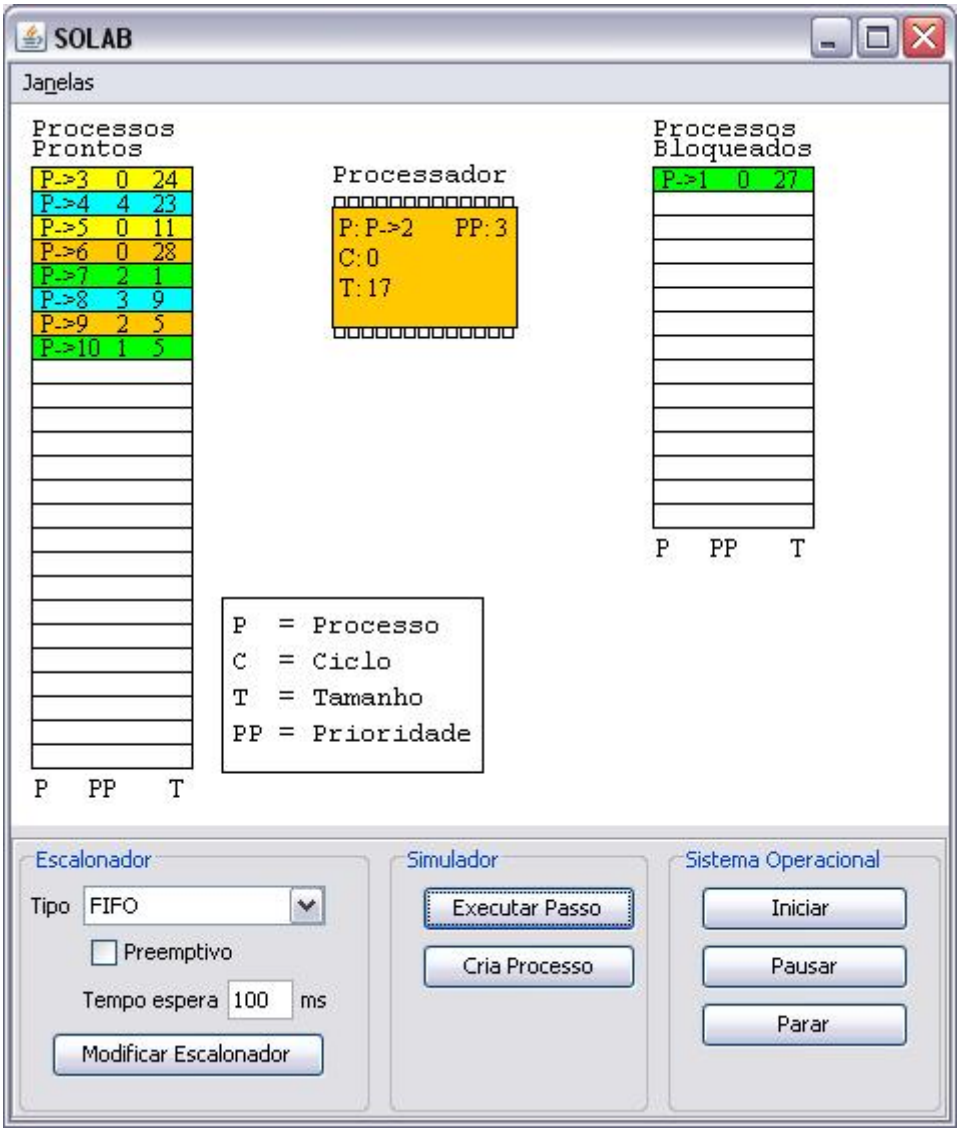


Figura 5. Interface principal do simulador

A janela é composta por três partes principais:

a) barra de menus – a partir dela o usuário pode acessar as janelas secundárias do sistema;

b) painel central – apresenta a parte gráfica do simulador, onde as animações referentes aos componentes do sistema operacional são apresentadas para o usuário;

c) painel inferior – agrupa os comandos principais do simulador. A partir deste painel é possível controlar o comportamento do simulador do sistema operacional.

No painel central é apresentada ao usuário, a imagem de um processador e duas filas de processos. A fila de esquerda é a de processos prontos para o escalonamento. A fila da direita é a de processos bloqueados executando operações de entrada/saída. Nas filas são apresentados o nome do processo, sua prioridade e seu tamanho, informações estas essenciais para o usuário identificar as características de cada algoritmo de escalonamento. As informações sobre o processo que está em execução são exibidas dentro da imagem do processador.

Nas filas de processos são exibidas as seguintes informações:

a) nome: um nome gerado para o processo, produzido pela soma do texto “P->” mais o identificador do processo;

b) prioridade: valor atribuído pelo Sistema Operacional que indica a prioridade de execução do processo. Quando o simulador está configurado para utilizar o algoritmo de escalonamento por prioridades, este valor é utilizado para indicar a ordem de precedência dos processos no acesso ao processador.

c) tamanho: valor que indica o número de instruções do processo. O tamanho de um processo é igual ao número de instruções que ele possui;

No painel inferior o usuário pode controlar o simulador. É possível escolher o tipo de algoritmo de escalonamento, criar um processo, executar instruções passo a passo, ou colocar o simulador em modo automático, onde são executadas as instruções sem intervenção do usuário, respeitando um intervalo de tempo.

Antes de iniciar uma simulação o usuário deve definir o tipo de algoritmo de escalonamento. Isto pode ser feito no painel inferior da janela principal, usando os controles do grupo escalonador. É possível escolher o tipo de algoritmo de escalonamento, definir se o mesmo vai utilizar preempção e escolher o tempo de espera em milisegundos que serve para definir a velocidade da simulação quando está em modo automático. No final do grupo existe o botão “Modificar Escalonador” que serve para aplicar as modificações estabelecidas.

Para controlar o funcionamento do escalonador existem no grupo “Simulador” dois botões. O botão “Executar Passo” permite que seja executada uma operação no processador. Cada instrução executada no processador define um ciclo. Então o controle de ciclos do simulador é definido pelo processador.

O outro botão, “Criar Processo”, serve para que o usuário possa adicionar processos à simulação. Este botão exibe uma janela que será mostrada na Figura 6 a seguir:

Figura 6. Janela do Simulador para Criar Processos

Na janela exibida, intitulada “Criar Processos” o usuário pode definir as seguintes opções:

a) tipo: define o tipo de processo que será criado. O usuário pode optar pela opção “Mais CPU” onde será criado processos com mais instruções de

processador. A próxima opção é “Mais Entrada/Saída” onde serão criados processos que simulam varias instruções de acesso a dispositivos de entrada/saída. A opção ambos combina características das duas versões anteriores;

b) prioridade: valor que indica a prioridade do processo;
c) páginas: valor que indica o tamanho máximo de páginas de memória que o processo vai ocupar. Este valor influência diretamente no tamanho do processo a ser criado;
d) quantidade: indica a quantidade de processos a serem criados;

Após escolher as opções o usuário pode clicar no botão “Criar Processos” para que os processos sejam adicionados a simulação. O botão “Criar Processos Aleatórios” permite que sejam criados processos com características aleatórias, ignorando as opções escolhidas nos campos acima. A única informação que interessa para este botão é a quantidade de processos a criar.

O grupo “Sistema Operacional” também disponível na janela agrupa os controles do sistema operacional para o modo automático. O botão “Iniciar” faz com que o simulador comece a executar ciclos de processador de forma seqüencial respeitando o intervalo em milisegundos definido no grupo “Escalonador”.

Durante a execução do simulador, os processos passam da Fila de Processos para o processador de acordo com a política de escalonamento definida. No processador conforme passam os ciclos, às instruções do processo são simuladas. Quando uma instrução de Entrada/Saída é executada, o processo passa para a fila de Processos Bloqueados. Ele aguarda nessa fila por um tempo aleatório e no final volta para a fila de processos. Se o usuário definiu que o escalonador deve utilizar preempção, o simulador, a cada ciclo, verifica na Fila de Processos se não existe um processo que tenha as

características, definidas pela política de escalonamento, para receber o processador. Caso isso ocorra, o processo que está no processador é movido para a fila de processos e o novo processo recebe o processador. Para evitar que processos ocupem o processador indefinidamente, o escalonador define um número máximo de ciclos que os processos possam ocupar o processador.

A interface principal do simulador, mostrada na Figura 5 tem como objetivo mostrar uma animação do funcionamento de um escalonador. Porém ela não oferece informações completas sobre os processos, pois isso poderia torná-la muito complexa e prejudicaria a animação. Desta forma, foi criada uma janela (Figura 7) separada para esta função.

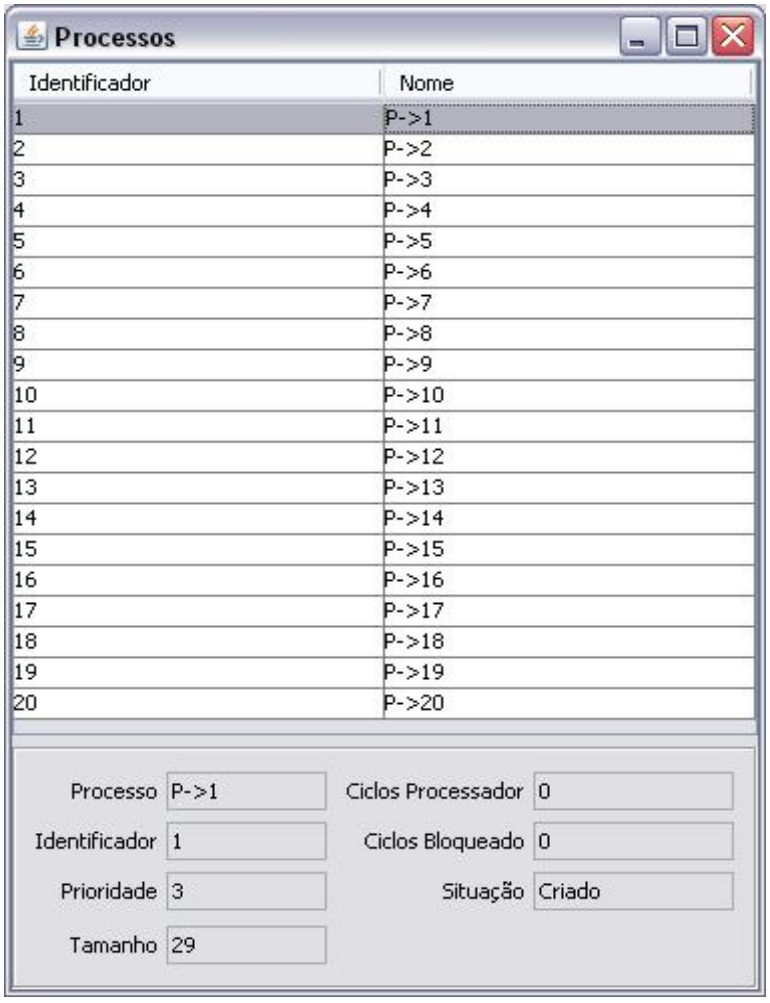


Figura 7. Janela de Processos do Simulador

O objetivo dessa janela é oferecer informações detalhadas sobre todos os processos que foram criados pelo acadêmico durante a simulação. Ela mostra uma tabela com o identificador e nome de cada processo, incluindo também os processos que já foram concluídos, para manter um histórico. No quadro abaixo dessa tabela, são mostradas as seguintes informações:

a) processo: nome do processo em questão;
b) identificador: número único atribuído pelo SO para cada processo criado;
c) tamanho: quantidade de instruções do processo;
d) prioridade: valor que indica a prioridade de execução do processo;
e) ciclos do processador: valor que indica a quantidade total de ciclos de processador utilizado pelo processo;
f) ciclos bloqueado: valor que indica a quantidade total de ciclos que o processo ficou na fila de processos bloqueados. Esse valor é incrementado toda vez que um processo executa uma instrução de Entrada/Saída simulada;
g) situação: estado em que o processo se encontra. Pode apresentar os seguintes valores: Criado, Pronto, Bloqueado e Concluído;

A janela de processos é atualizada a cada ciclo do processador, assim como a animação da janela principal. Com as informações dessa janela, o acadêmico pode fazer comparações e verificar a diferença entre os algorítmos de escalonamento.

A janela principal do simulador permite ao acadêmico acompanhar visualmente o funcionamento do escalonador, porém ocorrem no simulador inúmeros eventos simultâneos, como por exemplo, enquanto um processo recebe o processador um outro é removido da fila de processos bloqueados e passa para a de processos

prontos. Assim foi implementada uma janela que exibe em forma de texto, cada evento que ocorre no simulador. Esta janela é mostrada na figura a seguir:

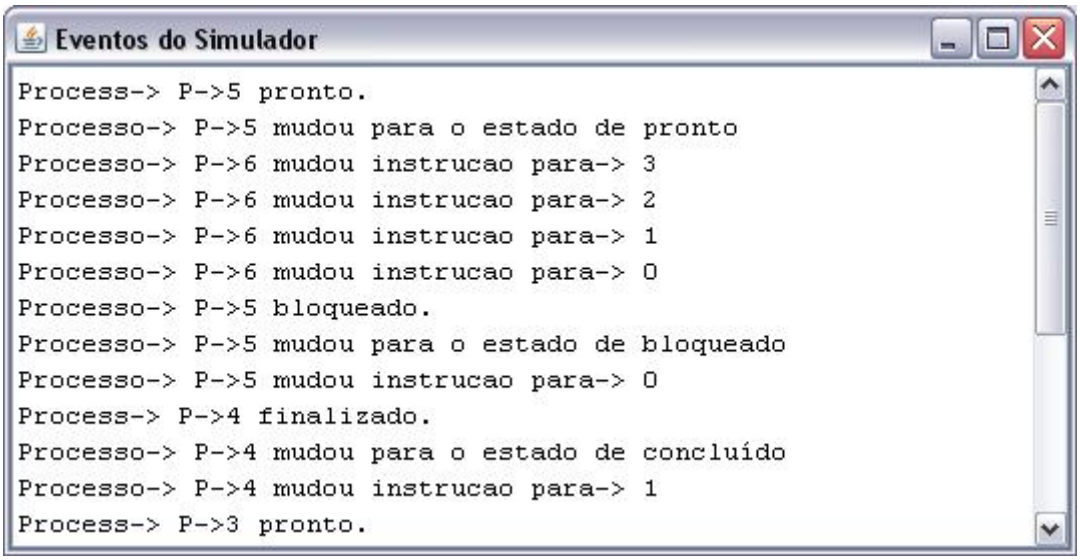


Figura 8. Janela de Eventos do Simulador

Somente os principais eventos do simulador ficam registrados nessa janela. Eles são ordenados de forma que os eventos mais recentes aparecem primeiro na lista. Para facilitar para o usuário todos os eventos de processo trazem antes o nome do processo ao qual ele se refere.

Para simular o gerenciamento de memória, o software desenvolvido possui uma janela separada, como mostra a Figura 9.

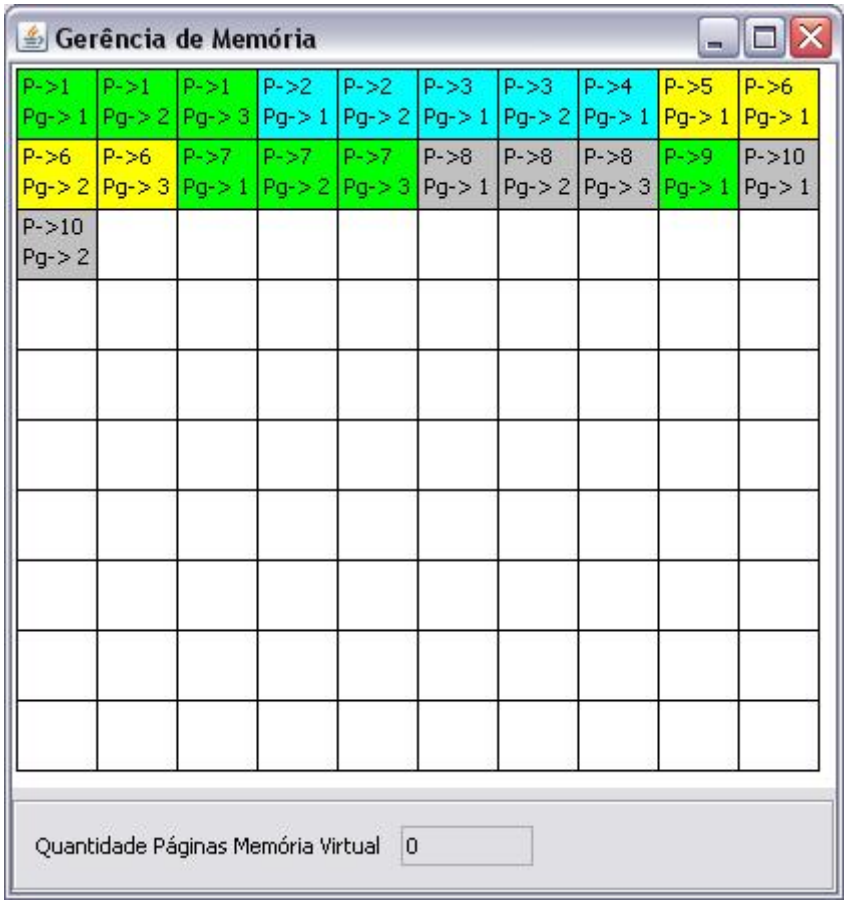


Figura 9. Janela de Gerência de Memória do Simulador

A interface de gerenciamento de memória simula o formato da memória real. Ela apresenta uma imagem formada por cem quadrados menores que representam a memória do computador dividida em cem blocos iguais. Como o simulador implementa o gerenciamento de memória por paginação, esses blocos representam as páginas da memória.

Cada página de memória pode ser usada ou estar livre. Quando uma página está livre, aparece na tela em branco. Se a página estiver ocupada, ela é mostrada com a cor do processo que a esta utilizando, junto com seu nome e o número da página. Assim o acadêmico pode acompanhar como a memória do sistema operacional simulado se comporta.

Quando um processo recebe o processador o sistema operacional simulado verifica qual foi a última instrução executada. Assim ele pode definir em que página está a próxima instrução que será executada pelo processo. Caso esta página esteja na memória virtual o SO verifica se existe memória livre suficiente para armazenar uma página do processo. Se existe memória disponível, o SO lê o processo que esta na memória virtual e o aloca na memória principal. Caso não exista, o sistema operacional seleciona uma página da memória principal, de forma aleatória e a armazena na memória virtual. No espaço livre que restou, o SO aloca a página do processo corrente. Desta forma o simulador implementa o mecanismo de memória virtual (*swap*). É importante ressaltar que os sistemas operacionais modernos realizam mecanismos complexos para escolher uma página de memória para swap. Estes mecanismos não foram implementados no simulador devido sua complexidade. Por isso, a escolha aleatória foi implementada, pois é um mecanismo simples de entender.

Todas as janelas do sistema apresentam legendas para facilitar o entendimento do usuário. Para tornar mais fácil a utilização de todo o simulador, foi criado uma janela de ajuda que agrupa informações sobre as funções do software. Esta janela é exibida na figura a seguir:

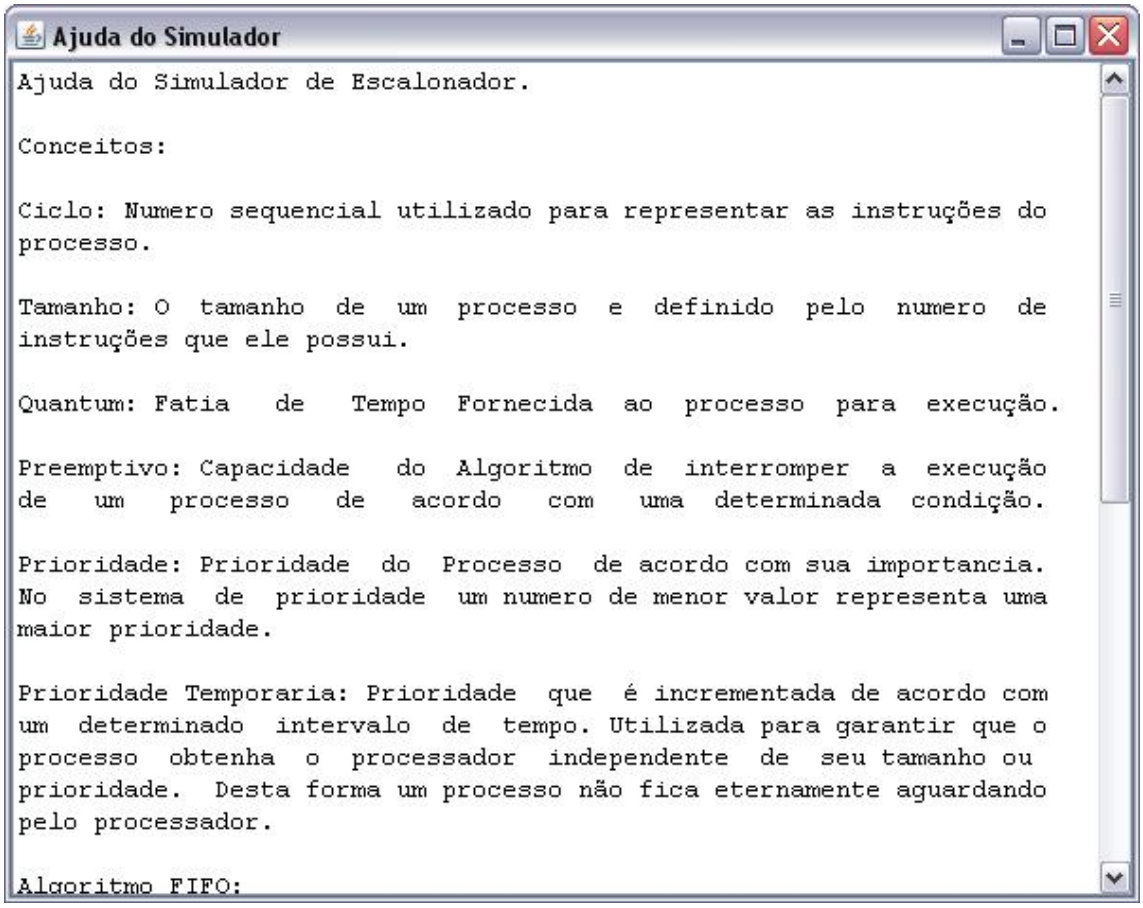


Figura 10. Janela de Ajuda do Simulador

Todos os conceitos apresentados no simulador são explicados nessa janela para que o usuário tenha todas as informações disponíveis no próprio simulador.

5.4 CONSIDERAÇÕES EDUCACIONAIS DO SIMULADOR

Todo o simulador foi desenvolvido buscando estar de acordo com os aspectos educacionais elencados na fundamentação. Desta forma, os seguintes requisitos que foram atendidos são listados a seguir:

a) amenidade ao uso: no desenvolvimento do software todos os comandos e textos foram dispostos de maneira a facilitar a sua utilização pelo

usuário. Os comandos foram agrupados de acordo com suas funções, sempre com uma legenda para indicar seu objetivo. Todas as abreviações que são exibidas na interface possuem uma legenda visível para saber imediatamente o que significam. Além disso, apesar de ser uma simulação onde são muitas as possibilidades de interação, o simulador não conduz o usuário a nenhuma situação onde ele fique sem saber o que o software esta fazendo ou qual o próximo passo. O simulador utiliza inúmeras animações para exibir as informações sendo que estas fazem uso de cores para identificar as partes correspondentes a cada processo. Outro recurso interessante é a possibilidade de serem gerados processos aleatórios, o que faz com que todas as simulações sejam diferentes, permitindo ao usuário, realizar várias iterações, assim desta forma, o mesmo poderá levantar hipóteses e interativamente, saber os seus resultados, comparando com os acontecimentos reais em um sistema operacional. Para finalizar o mesmo possui um manual que contém os conceitos a cerca da disciplina e que foram desenvolvidos no simulador;

b) independência do ambiente: o mesmo foi desenvolvido com o uso da linguagem de programação Java, assim permitindo que este possa ser executado em qualquer ambiente computacional;

c) correção: durante todos os testes realizados o simulador não apresentou nenhum erro durante sua execução. Todos os comandos do sistema possuem tratamento para que caso o usuário forneça uma resposta inadequada ou execute uma operação não permitida, o software exibe uma mensagem adequada e continua sua execução normalmente;

d) rentabilidade: durante a etapa de testes foi verificado que o software exige recursos mínimos no computador, tanto em velocidade de processamento como em capacidade de memória;

e) eficiência do processamento: o simulador oferece feedback automático para as ações do usuário. Por exemplo, se a simulação esta executando e o usuário decide adicionar um novo processo, no mesmo momento em que ele confirma a inclusão, o processo já aparece na fila de processos e passa a fazer parte da simulação. O software oferece poucas restrições na utilização do simulador ao usuário, assim, o mesmo garante a liberdade de interação no uso de recursos, que são de reconhecida importância na área educacional.

5.5 RESULTADOS OBTIDOS

O propósito inicial do projeto foi concluído com êxito no desenvolvimento do simulador para sistemas operacionais, mas o mesmo ainda não foi utilizado em sala de aula para verificar sua aceitação e se facilitará a mediação do ensino da disciplina de sistemas operacionais.

A principal dificuldade encontrada foi compreender os conceitos relacionados a softwares de apoio ao ensino e aplicar no simulador. Durante o desenvolvimento do simulador ocorreram várias discussões sobre tema com o professor orientador que é da área de sistemas operacionais para tentar aproximar ao máximo as características do software com as necessidades dos acadêmicos. Além disso, durante as pesquisas foram encontrados vários trabalhos que abordavam a questão das dificuldades existentes no ensino da disciplina de sistemas operacionais, o que ajudou muito no

desenvolvimento do trabalho, pois foi possível identificar quais as características necessárias para o desenvolvimento do projeto. Ficou constatada a necessidade da utilização de modelos computacionais para ilustrar os conceitos que são estudados em sala de aula.

No desenvolvimento do simulador foram encontradas algumas dificuldades na questão de como desenvolver as animações de um escalonador. Pois como o escalonador executa varias funções ao mesmo tempo, a animação teve que ser projetada para o usuário não ficar confuso durante a simulação. Desta forma, uma das soluções encontradas foi utilizar cores diferentes para cada processo, para o usuário localizar melhor as informações na interface. Além disso, foi criada uma janela de eventos onde tudo que acontece no simulador fica registrado.

CONCLUSÃO

A simulação por computador é uma ferramenta que serve para os mais diversos propósitos. Ela á utilizada em diversas áreas para reproduzir situações do mundo real e analisar suas características, como por exemplo, na indústria automobilística onde são utilizados modelos simulados em super computadores para testar as características de um veículo antes de ele ser produzido. Na área educacional ela é importante, porque possibilita ao acadêmico a construção do conhecimento em relação ao objeto de estudo, permitindo visualizar conceitos que não podem ser estudados pela técnica de observação, como é o caso dos softwares, que somente existem em um ambiente virtual.

Este projeto aplicou as técnicas de simulação para reproduzir de forma visual, parte de um sistema operacional, o software que controla as funções de um computador. Foram empregadas tecnologias de animação para mostrar a lógica de funcionamento de um sistema operacional e a relação que existe entre seus componentes. As animações são bem simples de entender, todas elas com legendas para que o acadêmico possa localizar facilmente os componentes do SO.

Como o SOLAB simula apenas parte de um SO, fica como sugestão para trabalhos futuros o desenvolvimento de outros componentes que sejam relevantes para o ensino acadêmico, como por exemplo, outros algorítmos de escalonamento, chamadas de sistema, gerenciamento de rede, gerenciamento de arquivos. Fica também como sugestão a implementação de mecanismos para análise de desempenho dos diferentes algorítmos de escalonamento, ou seja, coletar os dados gerados pelo simulador e implementar uma interface onde eles sejam exibidos em forma de gráficos e tabelas, para que o acadêmico possa realizar análises. No seu estado atual, o simulador trata os

processos bloqueados de forma uniforme, deixando eles na fila de espera por um tempo pré-definido. Uma sugestão seria adicionar diferentes chamadas de Entrada/Saída no sistema e tratar os diversos dispositivos de forma diferenciada. Assim o aluno poderia, por exemplo, verificar o tempo médio de espera de uma requisição ao disco rígido, configurar esse tempo no simulador para que quando o simulador executar uma chamada simulada de acesso ao disco, o processo fique bloqueado pelo tempo definido pelo acadêmico, adicionando assim mais detalhes a simulação, enriquecendo o aprendizado do acadêmico.

O software será publicado na Internet no site da Universidade (UNESC) para ficar disponível aos acadêmicos da instituição bem como para qualquer profissional que deseje compreender o funcionamento de um sistema operacional.

REFERÊNCIAS

BOOCH, Grady; JACOBSON, Ivar. **UML: Guia do Usuário.** 2. ed. São Paulo: Campus, 2005.

CAMPOS, Gilda Helena B. de. **Como avaliar um software educacional?** Disponível em: <http://www.timaster.com.br/revista/colunistas/ler_colunas_emp.asp?cod=331&pag=1>. Acesso em: 22 ago. 2008.

CAMPOS, Gilda Helena B. de. **O que determina a qualidade de um software educacional?** Disponível em: <http://www.timaster.com.br/revista/colunistas/ler_colunas_emp.asp?cod=310>. Acesso em: 22 ago. 2008.

COSTA, Carla Patrícia Ferreira. **O Software "Show do Milhão" como estratégia pedagógica.** 2002. Teste (Mestrado) - Universidade Federal de Santa Catarina, Florianópolis, 2002.

CHAVES, Eduardo O C. **O Computador na Educação.** Disponível em: <http://edutec.net/Textos/Self/EDTECH/funteve.htm>. Acesso em: 19 ago. 2008.

DEITEL, H. M.; DEITEL, P. J.; CHOFFNES, D. R.. **Sistemas Operacionais.** 3. ed. São Paulo: Pearson, 2005.

DEITEL, Paul J.. **Java como Programar.** 6. ed. São Paulo: Prentice-hall, 2005.

GUEDES, Gilleanes T. A.. **UML: uma Abordagem Prática.** São Paulo: Novatec, 2004.

HANLEY, Susan. **On Constructivism.** Disponível em: <http://www.inform.umd.edu/UMS+State/UMDProjects/MCTP/Essays/Constructivism.txt>. Acesso em: 21 abr. 2008.

HAYDAT, Regina Célia Cazaux. **Curso de Didática Geral.** São Paulo: Ática, 1997.

HERROD, Stephen Alan. **Using Complete Machine Simulation to Understand Computer.** 1998. 121 f. Tese (Phd) - Stanford University, Stanford, 1998.

JONASSEN, D.. **Computers in the Classrom.** New Jersey: Prentice Hall, 1996.

LA TAILLE, Yves De. **Teorias Psicogenéticas em Discussão.** 13. ed. São Paulo: Summus, 1992.

LOPES, José Junio. **A Introdução da informática no ambiente escolar.** Disponível em: <http://www.clubedoprofessor.com.br/artigos/artigojunio.htm>. Acesso em: 19 ago. 2008.

MACHADO, Francis Berenger; MAIA, Luiz Paulo. **Arquitetura de Sistemas Operacionais.** 4. ed. São Paulo: Ltc, 2007.

MAIA, Luiz Paulo. **SOsim: Simulador para o ensino de sistemas operacionais.** 2001. 85 f. Teste (Mestrado) - Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2001.

MARQUES, Cristina P. C. et al. **Computador e Ensino.** São Paulo: Ática, 1997.

MERGEL, Brenda. **Instructional Design and Learning Theory.** Disponível em: <http://www.usask.ca/education/coursework/802papers/mergel/brenda.htm>. Acesso em: 21 abr. 2008.

MOREIRA NETO, Oziel. **Entendendo e Dominando Java.** 2. ed. São Paulo: Digerati Books, 2007. 416 p.

RAMOS, Ricardo Argenton. **Treinamento Prático em UML.** São Paulo: Digerati Books, 2006. 143 p.

SHAY, Willian A.. **Sistemas Operacionais.** São Paulo: Makron Books, 1996.

SILBERSCHATZ, Abraham; GAGNE, Greg; GALVIN, Peter Baer. **Sistemas Operacionais - Conceitos e Aplicações.** São Paulo: Campus, 2001.

TANENBAUM, Andrew S.. **Sistemas Operacionais Modernos.** 2. ed. São Paulo: Prentice-hall, 2003.

TANENBAUM, Andrew S.; WOODHULL, Albert S.. **Sistemas operacionais : projeto e implementação.** 2. ed. Porto Alegre: Bookman, 2000.

TEIXEIRA, Jacqueline de Fátima. **Uma discussão sobre a classificação de software educacional de acordo com o paradigma educacional predominante.** Disponível em: <http://www.ccuec.unicamp.br/revista/infotec/artigos/jacqueline.html>. Acesso em: 22 ago. 2008.

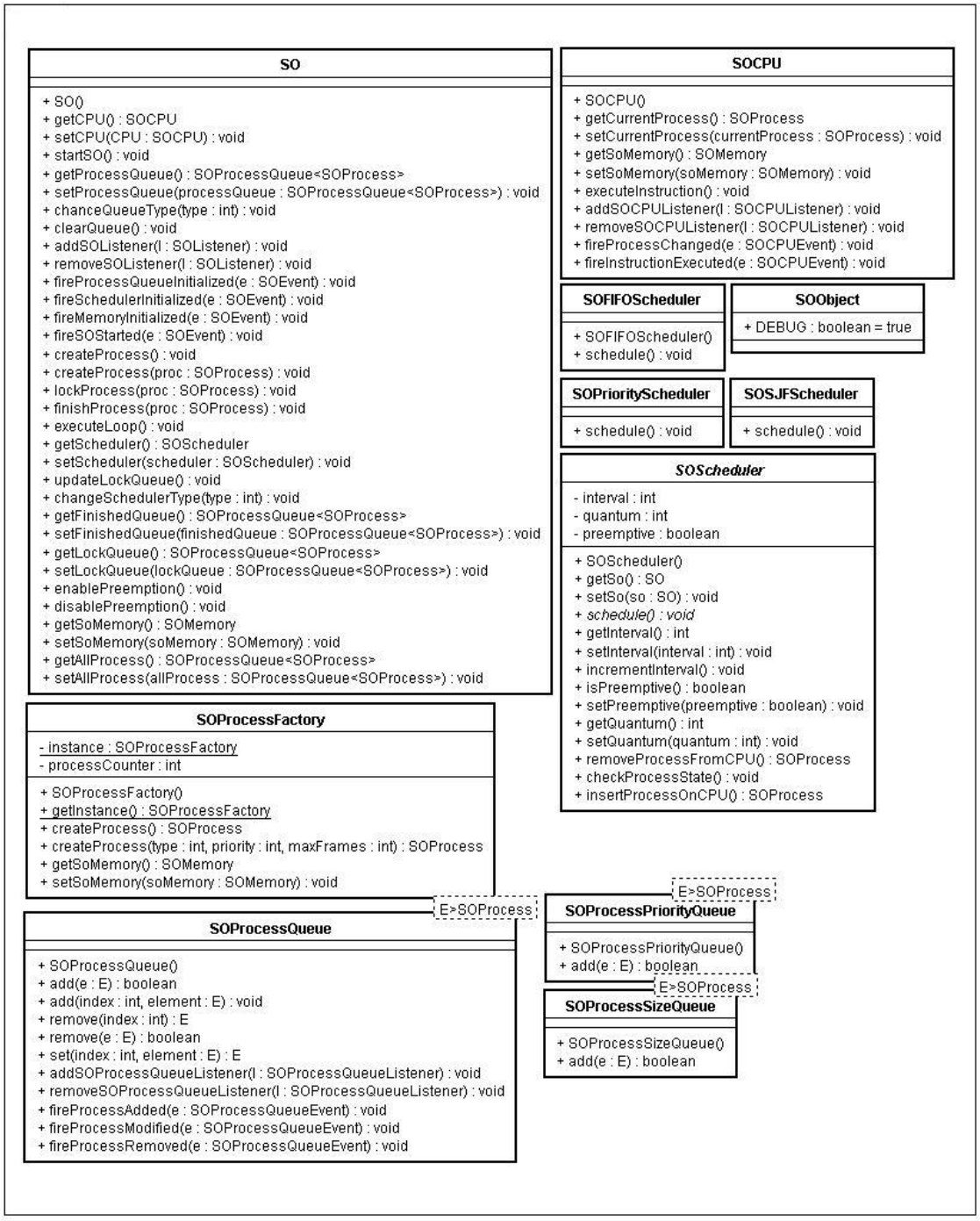
TOSCANI, Simao Sirineo; OLIVEIRA, Romulo Silva De; CARISSIMI, Alexandre Da Silva. **Sistemas Operacionais.** 3. ed. São Paulo: Sagra Luzzatto, 2004.

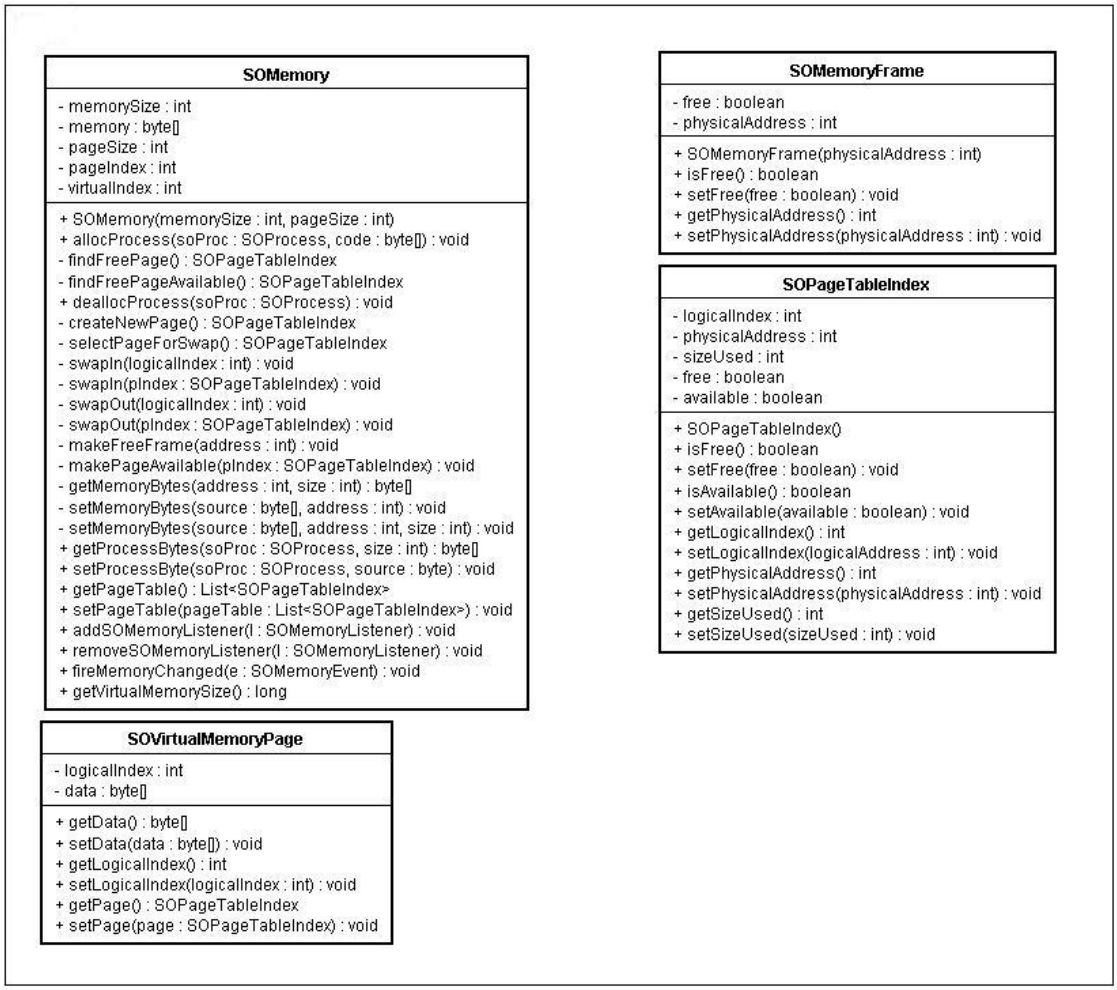
VALENTE, José Armando. **Diferentes usos do Computador na Educação.** Disponível em: <http://www.nied.unicamp.br/publicacoes/separatas/Sep1.pdf>. Acesso em: 22 ago. 2008

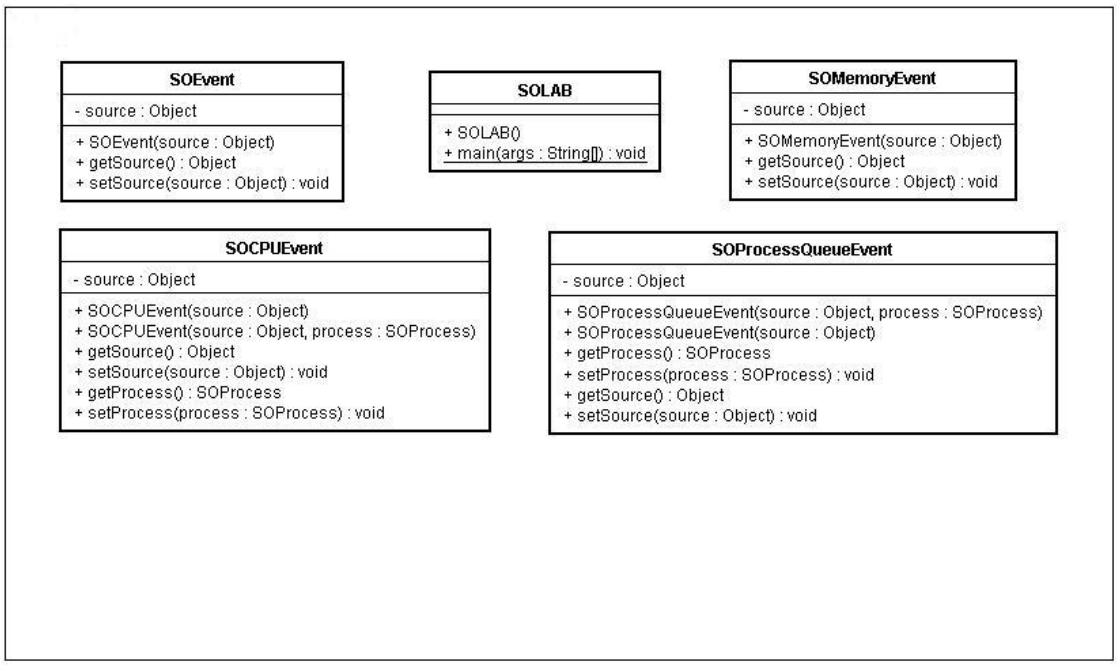
VALENTE, José Armando. **Por quê o Computador na Educação ?** Disponível em: <http://www.nied.unicamp.br/publicacoes/separatas/Sep2.pdf>. Acesso em: 22 ago. 2008.

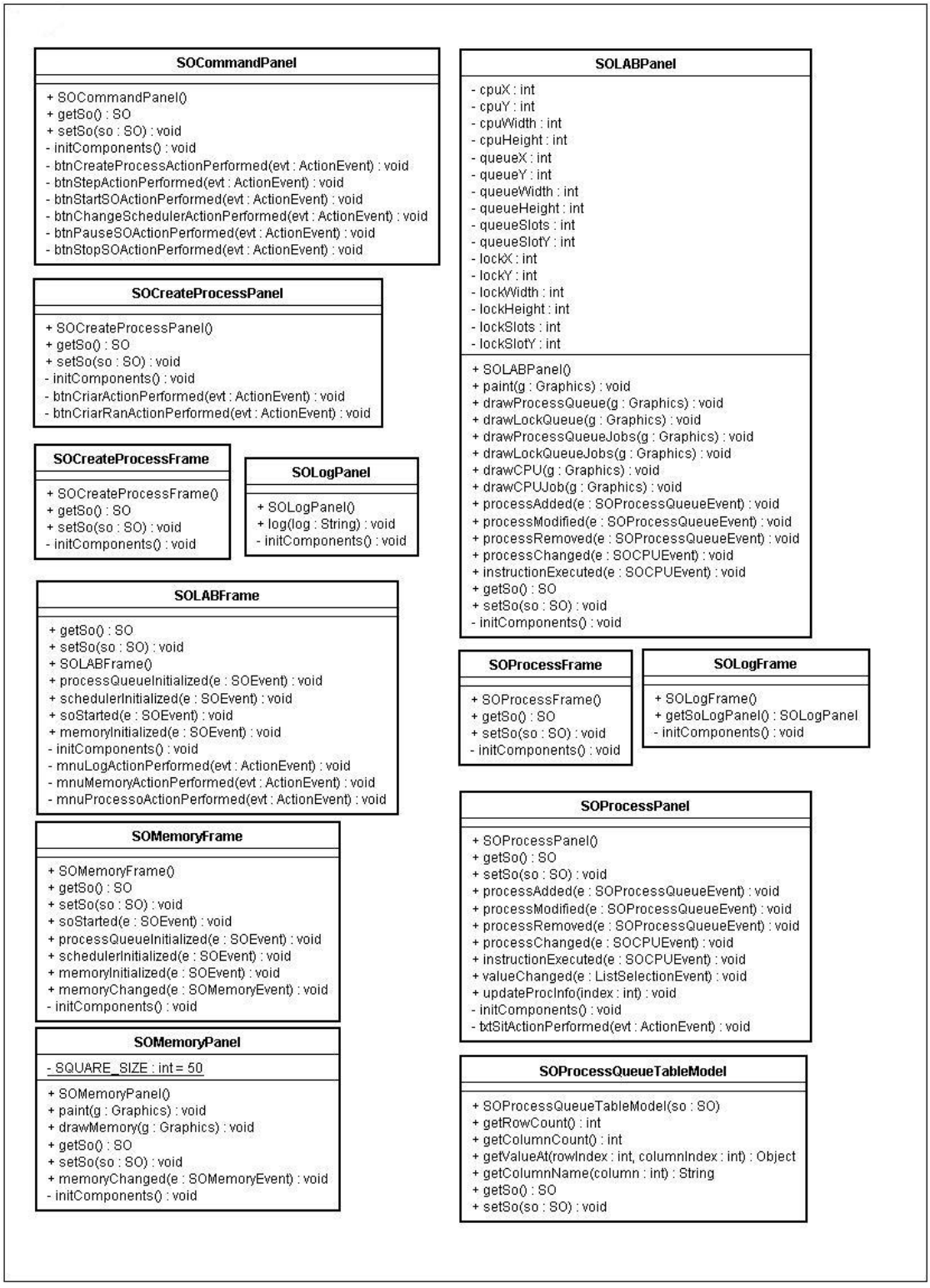
APÊNDICE

Em anexo os diagramas de classe do simulador SOLAB.




SOMemory
- memorySize : int
- memory : byte[]
- pageSize : int
- pageIndex : int
- virtualIndex : int
+ SOMemory(memorySize : int, pageSize : int)
+ allocProcess(soProc : SOProcess, code : byte[]) : void
- findFreePage() : SOPageTableIndex
- findFreePageAvailable() : SOPageTableIndex
+ deallocProcess(soProc : SOProcess) : void
- createNewPage() : SOPageTableIndex
- selectPageForSwap() : SOPageTableIndex
- swapIn(logicalIndex : int) : void
- swapIn(pIndex : SOPageTableIndex) : void
- swapOut(logicalIndex : int) : void
- swapOut(pIndex : SOPageTableIndex) : void
- makeFreeFrame(address : int) : void
- makePageAvailable(pIndex : SOPageTableIndex) : void
- getMemoryBytes(address : int, size : int) : byte[]
- setMemoryBytes(source : byte[], address : int) : void
- setMemoryBytes(source : byte[], address : int, size : int) : void
+ getProcessBytes(soProc : SOProcess, size : int) : byte[]
+ setProcessByte(soProc : SOProcess, source : byte) : void
+ getPageTable() : List<SOPageTableIndex>
+ setPageTable(pageTable : List<SOPageTableIndex>) : void
+ addSOMemoryListener(l : SOMemoryListener) : void
+ removeSOMemoryListener(l : SOMemoryListener) : void
+ fireMemoryChanged(e : SOMemoryEvent) : void
+ getVirtualMemorySize() : long
SOMemoryFrame
- free : boolean
- physicalAddress : int
+ SOMemoryFrame(physicalAddress : int)
+ isFree() : boolean
+ setFree(free : boolean) : void
+ getPhysicalAddress() : int
+ setPhysicalAddress(physicalAddress : int) : void
SOPageTableIndex
- logicalIndex : int
- physicalAddress : int
- sizeUsed : int
- free : boolean
- available : boolean
+ SOPageTableIndex()
+ isFree() : boolean
+ setFree(free : boolean) : void
+ isAvailable() : boolean
+ setAvailable(available : boolean) : void
+ getLogicalIndex() : int
+ setLogicalIndex(logicalAddress : int) : void
+ getPhysicalAddress() : int
+ setPhysicalAddress(physicalAddress : int) : void
+ getSizeUsed() : int
+ setSizeUsed(sizeUsed : int) : void
SOVirtualMemoryPage
- logicalIndex : int
- data : byte[]
+ getData() : byte[]
+ setData(data : byte[]) : void
+ getLogicalIndex() : int
+ setLogicalIndex(logicalIndex : int) : void
+ getPage() : SOPageTableIndex
+ setPage(page : SOPageTableIndex) : void


SOEvent
- source : Object
+ SOEvent(source : Object)
+ getSource() : Object
+ setSource(source : Object) : void
SOLAB
+ SOLAB()
+ main(args : String[]) : void
SOMemoryEvent
- source : Object
+ SOMemoryEvent(source : Object)
+ getSource() : Object
+ setSource(source : Object) : void
SOCPUEvent
- source : Object
+ SOCPUEvent(source : Object)
+ SOCPUEvent(source : Object, process : SOProcess)
+ getSource() : Object
+ setSource(source : Object) : void
+ getProcess() : SOProcess
+ setProcess(process : SOProcess) : void
SOProcessQueueEvent
- source : Object
+ SOProcessQueueEvent(source : Object, process : SOProcess)
+ SOProcessQueueEvent(source : Object)
+ getProcess() : SOProcess
+ setProcess(process : SOProcess) : void
+ getSource() : Object
+ setSource(source : Object) : void


SOCommandPanel
+ SOCommandPanel()
+ getSo() : SO
+ setSo(so : SO) : void
- initComponents() : void
- btnCreateProcessActionPerformed(evt : ActionEvent) : void
- btnStepActionPerformed(evt : ActionEvent) : void
- btnStartSOActionPerformed(evt : ActionEvent) : void
- btnChangeSchedulerActionPerformed(evt : ActionEvent) : void
- btnPauseSOActionPerformed(evt : ActionEvent) : void
- btnStopSOActionPerformed(evt : ActionEvent) : void
SOCreateProcessPanel
+ SOCreateProcessPanel()
+ getSo() : SO
+ setSo(so : SO) : void
- initComponents() : void
- btnCriarActionPerformed(evt : ActionEvent) : void
- btnCriarRanActionPerformed(evt : ActionEvent) : void
SOCreateProcessFrame
+ SOCreateProcessFrame()
+ getSo() : SO
+ setSo(so : SO) : void
- initComponents() : void
SOLogPanel
+ SOLogPanel()
+ log(log : String) : void
- initComponents() : void
SOLABFrame
+ getSo() : SO
+ setSo(so : SO) : void
+ SOLABFrame()
+ processQueueInitialized(e : SOEvent) : void
+ schedulerInitialized(e : SOEvent) : void
+ soStarted(e : SOEvent) : void
+ memoryInitialized(e : SOEvent) : void
- initComponents() : void
- mnuLogActionPerformed(evt : ActionEvent) : void
- mnuMemoryActionPerformed(evt : ActionEvent) : void
- mnuProcessoActionPerformed(evt : ActionEvent) : void
SOMemoryFrame
+ SOMemoryFrame()
+ getSo() : SO
+ setSo(so : SO) : void
+ soStarted(e : SOEvent) : void
+ processQueueInitialized(e : SOEvent) : void
+ schedulerInitialized(e : SOEvent) : void
+ memoryInitialized(e : SOEvent) : void
+ memoryChanged(e : SOMemoryEvent) : void
- initComponents() : void
SOMemoryPanel
- SQUARE_SIZE : int = 50
+ SOMemoryPanel()
+ paint(g : Graphics) : void
+ drawMemory(g : Graphics) : void
+ getSo() : SO
+ setSo(so : SO) : void
+ memoryChanged(e : SOMemoryEvent) : void
- initComponents() : void
SOLABPanel
- cpuX : int
- cpuY : int
- cpuWidth : int
- cpuHeight : int
- queueX : int
- queueY : int
- queueWidth : int
- queueHeight : int
- queueSlots : int
- queueSlotY : int
- lockX : int
- lockY : int
- lockWidth : int
- lockHeight : int
- lockSlots : int
- lockSlotY : int
+ SOLABPanel()
+ paint(g : Graphics) : void
+ drawProcessQueue(g : Graphics) : void
+ drawLockQueue(g : Graphics) : void
+ drawProcessQueueJobs(g : Graphics) : void
+ drawLockQueueJobs(g : Graphics) : void
+ drawCPU(g : Graphics) : void
+ drawCPUJob(g : Graphics) : void
+ processAdded(e : SOProcessQueueEvent) : void
+ processModified(e : SOProcessQueueEvent) : void
+ processRemoved(e : SOProcessQueueEvent) : void
+ processChanged(e : SOCPUEvent) : void
+ instructionExecuted(e : SOCPUEvent) : void
+ getSo() : SO
+ setSo(so : SO) : void
- initComponents() : void
SOProcessFrame
+ SOProcessFrame()
+ getSo() : SO
+ setSo(so : SO) : void
- initComponents() : void
SOLogFrame
+ SOLogFrame()
+ getSoLogPanel() : SOLogPanel
- initComponents() : void
SOProcessPanel
+ SOProcessPanel()
+ getSo() : SO
+ setSo(so : SO) : void
+ processAdded(e : SOProcessQueueEvent) : void
+ processModified(e : SOProcessQueueEvent) : void
+ processRemoved(e : SOProcessQueueEvent) : void
+ processChanged(e : SOCPUEvent) : void
+ instructionExecuted(e : SOCPUEvent) : void
+ valueChanged(e : ListSelectionEvent) : void
+ updateProcInfo(index : int) : void
- initComponents() : void
- txtSitActionPerformed(evt : ActionEvent) : void
SOProcessQueueTableModel
+ SOProcessQueueTableModel(so : SO)
+ getRowCount() : int
+ getColumnCount() : int
+ getValueAt(rowIndex : int, columnIndex : int) : Object
+ getColumnName(column : int) : String
+ getSo() : SO
+ setSo(so : SO) : void