

Luiz Paulo Maia

**SOsim: SIMULADOR PARA O ENSINO DE
SISTEMAS OPERACIONAIS**

Orientador: Ageu C. Pacheco Jr., Ph.D.

Núcleo de Computação Eletrônica – NCE

Instituto de Matemática – IM

Universidade Federal do Rio de Janeiro – UFRJ

Rio de Janeiro, Março de 2001

**SOSim: SIMULADOR PARA O ENSINO DE
SISTEMAS OPERACIONAIS**

Luiz Paulo Maia

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE MATEMÁTICA/NÚCLEO DE COMPUTAÇÃO ELETRÔNICA (NCE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO, COMO PARTE DOS REQUISITOS NECESSÁRIOS À OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM INFORMÁTICA.

Aprovada por:

Prof. Ageu C. Pacheco Jr., Ph.D.

Prof. Oswaldo Vernet de Souza Pires, D.Sc.

Prof^a. Margareth Simões Penello Meirelles, D.Sc.

RIO DE JANEIRO, RJ – BRASIL

MARÇO DE 2001

FICHA CATALOGRÁFICA

MAIA, LUIZ PAULO.

SOsim: Simulador para o Ensino de Sistemas Operacionais. [Rio de Janeiro] 2001

IX, 85 p. 29,7 cm (IM/NCE/UFRJ, M.Sc., Informática, 2001)

Dissertação – Universidade Federal do Rio de Janeiro, IM/NCE

1. Sistemas Operacionais. 2. Educação. 3. Ensino.

I. Tese (Mestr. IM/NCE/UFRJ). II Título.

AGRADECIMENTOS

Gostaria de agradecer inicialmente a Sérgio Guedes (NCE/UFRJ), Eduardo Luiz Pareto (NCE/UFRJ), Osvaldo Vernet (NCE/UFRJ), Fábio Ferrentine (NCE/UFRJ), Francis Berenger Machado (PUC-Rio), José Lucas Mourão Rangel Netto (PUC-Rio), Sergio Lifschitz (PUC-Rio) e Denis Cople (UERJ) pelo apoio dado na realização deste trabalho.

Gostaria também de agradecer especialmente ao meu orientador, professor Ageu Pacheco (NCE/UFRJ), por sua paciência e habilidade em lidar com as dificuldades de um mestrando em tempo parcial.

Gostaria de agradecer à Deise Lobo Cavalcante (Dona Deise), Regina Célia Gomes Lima e aos demais profissionais da Secretaria Acadêmica do IM/UFRJ.

Gostaria de agradecer aos amigos e familiares, presentes e ausentes, que tanto me apoiaram para a realização deste trabalho.

Finalmente, gostaria de agradecer à Maria Clara e Maria Luiza por abrirem mão do tempo que deveria dedicar a elas para a confecção deste trabalho de tese.

DEDICATÓRIA

Gostaria de dedicar este projeto ao mestre dos mestres, Prof. Júlio Salek Aude (NCE/UFRJ), falecido poucas semanas antes da conclusão deste trabalho.

Serei sempre grato a ele, por me mostrar o que é ser realmente um professor. Dedicção, humildade, paciência, bom humor, conhecimento, simplicidade, didatismo eram apenas alguns dos adjetivos para uma aula sua.

Fora da sala de aula, o Prof. Salek era um implementador de idéias, uma usina de realizações. Alguns têm as idéias, outros implementam, mas poucos idealizam e realizam. Qualquer um com estes predicados poderia estar trabalhando em um dos mais modernos centros de pesquisa do mundo. Mas não, o Prof. Salek escolheu o caminho mais difícil para realizar seus desafios.

Professor, esteja o senhor onde estiver, saiba que seu trabalho e suas idéias continuarão a influenciar muitas pessoas, especialmente seus ex-alunos como eu. Espero, algum dia, voltar a frequentar suas classes. Muito obrigado por tudo.

Resumo da Tese apresentada ao IM/NCE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.).

SOsim: Simulador para o Ensino de Sistemas Operacionais

Luiz Paulo Maia

Março/2001

Orientador: Ageu C. Pacheco Jr.

Programa: Informática

Um problema comum a professores e alunos de disciplinas na área da Ciência da Computação, tais como arquitetura de computadores e sistemas operacionais, é a relativa dificuldade em caracterizar a real dinâmica dos eventos computacionais. Por melhor que sejam o conhecimento e a comunicação do mestre, o raciocínio e a atenção dispensados pelos alunos, o perfeito entendimento dos conceitos abordados fica comprometido pela estática inerente da abordagem das disciplinas.

No caso específico de sistemas operacionais, após vários anos envolvido com o seu ensino, passei a procurar uma forma de dinamizar os conceitos e técnicas apresentados. O presente trabalho, consequência desta experiência, visa construir um simulador gráfico (SOsim) que sirva de ferramenta visual de suporte efetivo para o ensino e aprendizado dos conceitos e técnicas implementados nos sistemas operacionais atuais, de forma a tornar este processo mais fácil e eficaz.

Após finalizado, o simulador será colocado disponível na Internet para que professores e alunos tenham acesso gratuito ao software de apoio educacional. Desta forma, o programa poderá ser utilizado como ferramenta de apoio ao ensino presencial ou à distância.

Abstract of Thesis presented to IM/NCE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

SOsim: A Simulator Supporting Lectures on Operating Systems

Luiz Paulo Maia

March/2001

Advisors: Ageu C. Pacheco Jr.

Department: Informatics

A common problem faced by teachers and students in classes of Computer Science matters, such as computer architecture and operating systems, is the relative difficulty in well reflecting the real dynamic nature of the computing events involved. No matter how solid might be the knowledge and the communicating ability of the instructor, as also the concentration and close attention paid by the students, the well understanding of the concepts presented is impaired by the implicit static nature of the class or lecture presentation.

In the specific case of operating systems, after many years lecturing on the subject, I started searching for a way on improving the dynamics of the concepts & techniques approached. The present work is the outcome of that search, and implements a simulator (SOsim) with visual graphic facilities to serve as an effective supporting tool for a better “teaching & learning” of the concepts & techniques implemented in current operating systems, as a way to turn the whole process more efficient.

On completion, the simulator will be made available on the Internet, so that free access to its educational contents will be assured to students and instructors.

SUMÁRIO

1 VISÃO GERAL

1.1 Introdução -----	1
1.2 Motivação-----	2
1.3 Trabalho Proposto -----	3
1.4 Estrutura do Trabalho -----	5

2 CONSIDERAÇÕES EDUCACIONAIS

2.1 Introdução -----	6
2.2 Modelos de Ensino -----	6
2.3 Ferramentas de Ensino-----	8
2.4 Trabalhos Relacionados -----	10
2.4.1 BASI -----	11
2.4.2 NACHOS -----	12
2.4.3 OSP -----	13
2.4.4 MINIX-----	13
2.4.5 FreeBSD-----	14
2.4.6 TROPIX -----	14
2.5 SOsim -----	15

3 MODELO PROPOSTO

3.1 Introdução -----	19
3.2 Processo-----	21
3.2.1 Estados do Processo -----	23
3.2.2 Mudanças de Estado do Processo-----	24
3.3 Gerência do Processador -----	26
3.3.1 Escalonamento Não-Preemptivo-----	26
3.3.2 Escalonamento Preemptivo-----	27
3.3.3 Critérios de Escalonamento -----	31

3.4 Gerência de Memória -----	32
3.4.1 Memória Virtual -----	33
3.4.2 Mapeamento -----	35
3.4.3 Paginação -----	36
3.4.4 Política de Busca de Páginas -----	38
3.4.5 Política de Alocação de Páginas -----	39
3.4.6 Working Set -----	39
3.4.7 Política de Substituição de Páginas -----	41
3.4.8 Swapping -----	43
3.4.9 Thrashing -----	44

4 ARQUITETURA E IMPLEMENTAÇÃO

4.1 Introdução -----	45
4.2 Orientação por Objetos -----	46
4.3 Multithread -----	47
4.4 Ambiente do Simulador -----	49
4.5 Objetos do Simulador -----	54
4.6 Objeto Processo -----	57
4.6.1 Criação do Processo -----	58
4.6.2 Visualização dos Processos -----	61
4.6.3 Visualização dos PCBs -----	62
4.7 Gerência do Processador -----	63
4.8 Gerência de Memória -----	65
4.8.1 Mapeamento -----	68
4.8.2 Tamanho de Página -----	69
4.8.3 Lista de Frames -----	69
4.8.4 Política de Substituição de Páginas -----	70
4.8.5 Arquivo de Paginação -----	73
4.9 Estatísticas de Simulação -----	75

5 CONCLUSÕES E TRABALHOS FUTUROS

5.1 Introdução ----- 77

5.2 Conclusões----- 77

5.3 Trabalhos Futuros ----- 78

REFERÊNCIAS BIBLIOGRÁFICAS----- 81

LISTA DE FIGURAS

Fig. 1.1 – Visão da interface gráfica do simulador-----	4
Fig. 2.1 – Interação simulador, professor e aluno -----	15
Fig. 3.1 – Sistema operacional genérico -----	20
Fig. 3.2 – Processo-----	22
Fig. 3.3 – Mudanças de estado do processo -----	25
Fig. 3.4 – Espaço de endereçamento virtual-----	34
Fig. 3.5 – Tabela de mapeamento-----	35
Fig. 3.6 – Tabela de páginas-----	37
Fig. 4.1 – Ambiente monothread -----	48
Fig. 4.2 – Ambiente multithread-----	49
Fig. 4.3 – Console-----	50
Fig. 4.4 – Gerência do Processador-----	51
Fig. 4.5 – Gerência de Memória -----	52
Fig. 4.6 – Log do simulador -----	53
Fig. 4.7 – Estatísticas do simulador -----	53
Fig. 4.8 – Criação de processo(s) -----	59
Fig. 4.9 – Visualização dos processos -----	61
Fig. 4.10 – Process Control Block-----	62
Fig. 4.11 – Estados dos processos no simulador-----	63
Fig. 4.12 – Opções de escalonamento-----	64
Fig. 4.13 – Memória principal-----	66
Fig. 4.14 – Opções de memória-----	67
Fig. 4.15 – Tabela de páginas -----	68
Fig 4.16 – Política de substituição de páginas -----	71
Fig. 4.17 – Arquivo de paginação e swapping-----	74

LISTA DE TABELAS

Tab. 4.1 – Formulários -----	57
Tab. 4.2 – Códigos pré-definidos -----	60
Tab. 4.3 – Tipos de instruções -----	60
Tab. 4.4 – Operações sobre processos -----	61
Tab. 4.5 – Aumento de prioridade -----	65
Tab. 4.6 – Indicadores estatísticos -----	75

CAPÍTULO 1 - VISÃO GERAL

1.1 Introdução

Na maioria dos cursos de computação, quer sejam graduação, mestrado, pós-graduação ou extensão é oferecida uma disciplina ou mais voltadas especificamente para o ensino da arquitetura de sistemas operacionais. Estas disciplinas são a base de conhecimento conceitual para que os alunos tenham a compreensão do funcionamento de sistemas operacionais em uso atualmente, como as diversas versões do Unix (Linux, Solaris, AIX, FreeBSD etc.) e a família Windows da Microsoft.

O currículo da disciplina Sistemas Operacionais envolve diversos aspectos um tanto complexos, tais como o conceito de processos, escalonamento, gerência de memória virtual etc. A experiência de alunos e professores nesta área tem mostrado como é grande a dificuldade em ensinar e compreender os modelos e técnicas apresentados.

O modelo tradicional de aula em que o professor segue uma bibliografia e um cronograma rígido de aulas, não é suficiente para que a maioria dos alunos tenha uma compreensão precisa do que está sendo ensinado. O problema não está no modelo de ensino, baseado no comportamentalismo, mas na falta de ferramentas capazes de traduzir os conceitos teóricos apresentados em conceitos práticos.

Envolvido profissionalmente com a formação de recursos humanos técnicos, especificamente nesta área de sistemas operacionais, pude observar durante estes anos que as dificuldades do ensino e aprendizado do assunto não estão restritas somente ao âmbito acadêmico. As mesmas limitações aparecem em cursos específicos para profissionais responsáveis pela manutenção e gerência de sistemas comerciais.

O simulador aqui desenvolvido tem como objetivo auxiliar o ensino e aprendizado de sistemas operacionais. O SOsim servirá como ferramenta de suporte visual para aulas de sistemas operacionais, inclusive aquelas ministradas à distância. A distribuição do software será gratuita, estando disponível na Internet para download.

1.2 Motivação

No início da década de 1990, a bibliografia de referência existente de boa qualidade no país sobre o tema Sistemas Operacionais era unicamente em língua inglesa, dificultando em muito sua aplicação nos cursos relacionados ao assunto. Ciente desta carência, o livro "Introdução à Arquitetura de Sistemas Operacionais" foi escrito e publicado, na tentativa de facilitar o ensino e aprendizado de sistemas operacionais [MACHADO, 1992]. A primeira edição foi amplamente aceita pela comunidade acadêmica, principalmente nos cursos de graduação e extensão. Uma segunda edição revisada e ampliada foi lançada posteriormente com o novo nome de "Arquitetura de Sistemas Operacionais" [MACHADO, 1997]. Neste período, o livro foi adotado em inúmeras instituições de ensino, inclusive em outros estados. Em 1998, colocamos na Internet uma página com informações adicionais sobre o livro e o tema [MAIA, 1998].

Apesar de todo o trabalho desenvolvido para facilitar o ensino e aprendizado de sistemas operacionais, existe um problema que o material impresso não resolve. Por melhores que sejam o professores e alunos, a apresentação da dinâmica dos algoritmos e mecanismos implementados limita-se à visualização de uma pequena seqüência de eventos que tenta representar um grande número de situações. Por exemplo, o mecanismo de gerência de memória virtual, que envolve tabelas de mapeamento e tradução de endereços, é extremamente difícil de ser apresentado de forma clara, devido à sua dinâmica e complexidade.

A melhor forma de resolver o problema da apresentação da dinâmica de um sistema operacional seria implementar uma ferramenta que reduzisse a distância entre o modelo real e o modelo conceitual, tornando este último mais próximo da realidade. Com base nesta idéia, foi desenvolvido um simulador gráfico (SOSim) que serve de ferramenta visual de suporte para o ensino e aprendizado dos conceitos e técnicas implementados nos sistemas operacionais atuais, de forma a tornar este processo mais efetivo.

O simulador será colocado disponível na Internet para que professores e alunos tenham acesso gratuito ao software de apoio educacional. Juntamente com o simulador, será

lançada a terceira edição do livro “Arquitetura de Sistemas Operacionais”. Nesta nova edição, além da revisão e ampliação do material, estaremos atuando mais fortemente na Internet, oferecendo exercícios e propostas de trabalhos baseados no SOSim.

1.3 Trabalho Proposto

A idéia do presente projeto de tese em questão surgiu a partir de um trabalho de final de curso de graduação em Informática na UFRJ, desenvolvido para a disciplina de Arquitetura de Computadores I, orientado na ocasião pelo Prof. Ageu C. Pacheco Jr., denominado de CPUsim [JUNQUEIRA, 1997].

O CPUsim permite visualizar dinamicamente o funcionamento de uma unidade central de processamento (CPU), através do acompanhamento das ações de execução e de transferência em/entre seus componentes clássicos: unidade lógica e aritmética (ULA), unidade de controle (UC) e memória principal. O CPUsim tornou-se uma ótima ferramenta de auxílio ao ensino de disciplinas ligadas à arquitetura e organização de computadores.

O SOSim tem objetivos semelhantes ao simulador CPUsim, ou seja, servir de ferramenta visual que mostre de forma precisa o funcionamento e os conceitos envolvidos em um sistema operacional multiprogramável. Através de uma interface simples, o aluno e/ou professor podem utilizar o software educacional para consolidar conceitos apresentados em sala de aula. Conceitos como processo, gerência de processador e gerência de memória podem ser apresentados e analisados com o auxílio do simulador (Fig. 1.1).

O simulador é de fácil instalação, não exige qualquer conhecimento de linguagem de programação para ser utilizado e pode ser executado em um computador pessoal (PC) que tenha o sistema operacional Microsoft Windows. O software foi desenvolvido em Borland Delphi, utilizando-se o conceito de orientação a objetos, o que permite o fácil acesso ao código fonte e sua alteração.

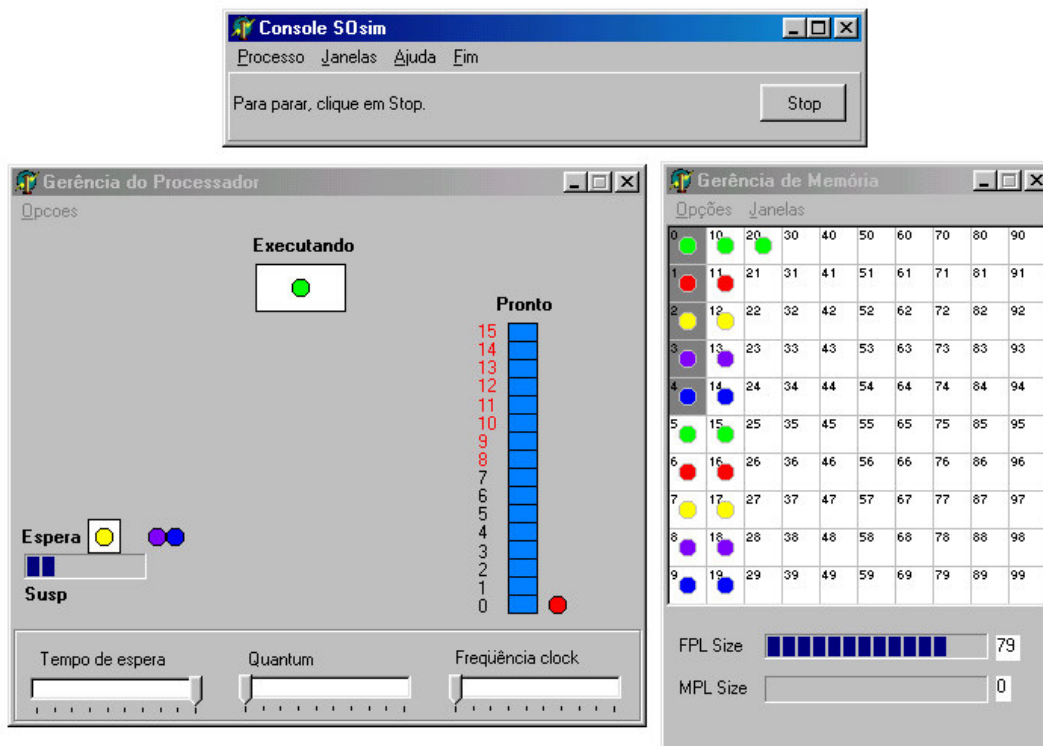


Fig. 1.1 - Visão da interface gráfica do simulador

A seguir, são apresentadas as principais contribuições esperadas para o ensino e aprendizado de sistemas operacionais, a partir da efetiva utilização do simulador SOsim:

- Ampliar a eficiência no processo de ensino/aprendizado, a partir de sua interface gráfica e dos recursos de animação;
- Melhorar a comunicação professor-aluno, ampliando o poder de compreensão e entendimento dos conceitos por parte dos alunos;
- Oferecer apoio para aulas presenciais e à distância;
- Tornar a transferência e aquisição de conhecimento mais participativa;
- Atingir vários tipos de cursos de sistemas operacionais e uma ampla diversidade de professores e alunos.

1.4 Estrutura do Trabalho

Este trabalho foi organizado em capítulos, sendo que neste primeiro capítulo foi feita uma abordagem geral das motivações para o desenvolvimento desta dissertação de tese.

O capítulo 2 apresenta os fundamentos educacionais que norteiam o projeto do simulador. O capítulo 3 enfoca as diversas características do modelo proposto, base para a compreensão dos demais capítulos.

O capítulo 4 descreve a arquitetura e implementação do simulador, com suas estruturas internas, algoritmos e interfaces gráficas, dando ênfase à gerência do processador e gerência de memória, com atenção especial para a gerência de memória virtual.

Finalmente, o capítulo 5 trata das conclusões sobre a tese apresentada e oferece sugestões para trabalhos futuros.

CAPÍTULO 2 – CONSIDERAÇÕES EDUCACIONAIS

2.1 Introdução

O simulador SOsim é um software educacional voltado ao suporte do ensino de conceitos ligados à arquitetura de sistemas operacionais modernos, no entanto, o simulador precisa ser utilizado dentro de um contexto pedagógico. Isto significa que o software não é um fim em si mesmo, mas um meio de se ampliar as possibilidades e efetividades do ensino.

Este capítulo apresenta as considerações educacionais que servem de suporte para o simulador. Inicialmente são apresentados os modelos de ensino comportamentalista e construtivista, e os tipos de ferramentas de ensino utilizadas atualmente. Em seguida, são analisados os trabalhos relacionados disponíveis. Finalmente são apresentadas as vantagens educacionais da utilização do simulador SOsim para o ensino de sistemas operacionais.

2.2 Modelos de Ensino

De uma forma bastante resumida, serão apresentados os dois modelos de ensino/aprendizado mais adotados atualmente: o comportamentalista (behaviorista) e o construtivista [MELO, 1999]. A compreensão destas duas teorias de ensino é necessária para entender as vantagens pedagógicas trazidas pelo projeto SOsim.

No modelo comportamentalista, os alunos ficam diante de um professor que passa seus conhecimentos seguindo um programa de curso bem definido e seqüencial, geralmente com base em uma bibliografia específica. Nesta teoria, os alunos são avaliados através de notas, a partir de critérios previamente definidos pelo professor. O modelo comportamentalista é fundamentado principalmente na figura do professor, ou seja, o foco é centrado no professor e não no aluno [BRENDA, 1998]. O comportamentalismo tem sido o modelo de ensino/aprendizado dominante no país.

No modelo construtivista, o professor tem a função de facilitador (ou mediador) do conhecimento e não um transmissor deste. Sua função é motivar nos alunos o espírito de investigação crítico, participativo e cooperativo. Neste modelo, o foco principal é no aluno e o professor passa a ter o papel de tutor e incentivador do conhecimento [HANLEY, 1994].

A concepção do simulador é utilizar a teoria construtivista para incentivar o aluno a explorar as diversas situações e aprender com seus próprios erros, o que é evitado no modelo comportamentalista. Apesar do software não oferecer a possibilidade de construção colaborativa de conhecimento, são apresentadas algumas outras características que o qualificam como tal:

- Proporciona múltiplas representações da realidade;
- Enfatiza a construção do conhecimento;
- Oferece um ambiente de aprendizado que simula a realidade, utilizando estudo de casos;
- Favorece o pensamento reflexivo em função da experiência com o ambiente de simulação;
- Facilita a identificação, definição e solução de problemas;
- Permite ao aluno o controle do experimento, garantindo uma evolução natural da complexidade das simulações.

A grande vantagem deste projeto de software educacional é permitir a criação de um ambiente híbrido de ensino/aprendizado, onde o comportamentalismo é aplicado em sala de aula e o construtivismo como apoio à formação do conhecimento, permitindo a experimentação das teorias apresentadas em sala de aula.

É previsto em versões futuras tornar o SOsim em uma ferramenta educacional mais colaborativa, permitindo trabalhos em grupos na forma de projetos e a construção colaborativa do conhecimento. Seriam introduzidas interfaces pré-definidas com os principais módulos do simulador, de forma que novos módulos pudessem ser implementados/alterados conforme a realidade a ser simulada.

2.3 Ferramentas de Ensino

As primeiras ferramentas de ensino que utilizavam o computador foram baseadas na máquina de B.F. Skinner, que implementava o conceito de instrução programada. Este conceito baseia-se em dividir o conteúdo do curso em módulos pequenos, lógicos e seqüencialmente encadeados. Cada módulo é finalizado com uma ou mais questões que devem ser respondidas pelo aluno. Se o resultado esperado for alcançado o aluno passa para o próximo módulo, caso contrário o aluno deve retornar e rever um ou mais módulos [VALENTE, 1993].

Durante a década de 1960, diversos programas de instrução programada foram implementados, dando início ao conceito de Computer Aided Instruction (CAI) ou Programas Educacionais por Computador (PEC). A disseminação do CAI somente aconteceu com a massificação do uso dos microcomputadores, permitindo a criação de vários tipos de cursos, como tutoriais, exercício-e-prática, avaliações, jogos e simulações.

Existe um consenso que o computador não deve ser utilizado para ensinar diretamente, substituindo o professor, mas sim promover o aprendizado, passando a ser uma ferramenta educacional. Isto significa que o professor deixa de ser o repassador de conhecimento e torna-se o criador de ambientes de ensino e facilitador do processo de aprendizado.

- **Programas Tutoriais**

Os programas tutoriais são uma versão computadorizada das aulas tradicionais. Suas vantagens em relação ao modelo convencional é a utilização de animações, som, controle do desempenho do aluno, aferição do aprendizado etc.

A maioria dos programas tutoriais são versões eletrônicas do mundo real, ou seja, da sala de aula, guardando todos os problemas deste modelo de ensino. A tendência dos

programas tutoriais é a implementação de técnicas de inteligência artificial para analisar, avaliar e oferecer soluções para as dificuldades do aluno.

- **Programas Exercício-e-Prática**

Os programas exercício-e-prática (drill-and-practice) são utilizados para revisar os conceitos apresentados em aula, utilizando as características de multimídia dos sistemas computadorizados. Estes programas requerem respostas freqüentes do aluno, propiciando o feedback imediato, exploram o lado de multimídia e interatividade do software educacional, sendo apresentados geralmente em forma de jogos.

- **Simulações**

Uma simulação envolve a criação de modelos dinâmicos e simplificados do mundo real. As primeiras simulações foram desenvolvidas para criar um ambiente seguro para atividades que oferecessem risco ao ser humano, como as simulações de viagens espaciais e mergulhos profundos. Posteriormente, as simulações foram aplicadas a processos que exigiam grande investimento de tempo e/ou dinheiro, como na indústria automobilística e aviação.

No mundo acadêmico, a simulação permite ao aluno desenvolver hipóteses, testá-las, analisar os resultados e sedimentar seus conhecimentos. O potencial educacional deste tipo de ferramenta é muito superior que os programas tradicionais, como tutoriais e exercício-e-prática.

Existem diversas áreas do conhecimento que fazem uso de simulações, como engenharia, física, química, biologia, economia etc. Na área da Ciência da Computação existem simuladores que auxiliam no ensino de várias disciplinas, como redes de computadores, técnicas de programação, arquitetura de computadores [HERROD, 1998] [JUNQUEIRA, 1997], sistemas operacionais, dentre outras.

A grande vantagem deste tipo de software educacional é sua simplicidade de utilização, dispensando o aluno de detalhes de instalação e interação dos ambientes reais. A simulação deve ser utilizada como uma complementação das aulas tradicionais, de forma a sedimentar os conceitos e técnicas apresentados.

Apesar de suas vantagens, a construção de simuladores não é simples, pois envolve um grande esforço de programação e recursos multimídia (som, vídeo, imagem) para tornar a simulação próxima da realidade. Estes fatos tornam os projetos de simuladores longos e dispendiosos financeiramente.

Este tipo de ferramenta é indicada para cursos onde o aluno possui pouco conhecimento prévio de sistemas operacionais, sendo indicado especialmente para cursos de extensão e graduações de curta duração.

2.4 Trabalhos Relacionados

Existem, atualmente, duas categorias de ferramentas que são utilizadas para o ensino de sistemas operacionais: sistemas reais e simuladores.

Os sistemas reais oferecem o código fonte para ser estudado e analisado por alunos e professores. Inicialmente, o sistema deve ser instalado em um hardware apropriado, exatamente como um sistema comercial. A partir de uma interface de comandos, o aluno ou professor poderá interagir com o sistema operacional e verificar sua implementação no código fonte do sistema.

A vantagem deste tipo de abordagem é permitir ao aluno estudar em detalhes a implementação de um sistema operacional, principalmente as interfaces com a arquitetura de hardware, possibilitando ao aluno realizar alterações no código fonte e verificar suas conseqüências na prática. Nesta categoria podemos citar os sistemas MINIX, FreeBSD, Linux e TROPIX.

O maior problema deste tipo de abordagem é justamente a necessidade de entrar em detalhes de implementação do sistema. A grande maioria dos currículos de sistemas operacionais não tem esse objetivo e, mesmo que tivesse, a carga horária disponível não seria suficiente.

Este tipo de ferramenta de ensino é mais indicado para estudantes de mestrado que estejam interessados em especialização na área de sistemas operacionais, que já tenham bons conhecimentos de arquitetura de computadores e sólidos conceitos de sistemas operacionais.

Simuladores, por outro lado, oferecem uma forma mais acessível a professores e alunos de estudar os mecanismos básicos de um sistema operacional, sem entrar em detalhes de instalação do software, da arquitetura de hardware, programação assembly etc. Nesta categoria podemos citar o BASI, NACHOS, OSP e o SOsim, objeto desta tese.

No caso específico de simuladores para o estudo de sistemas operacionais existem os genéricos, que abordam a maioria dos módulos de gerência necessários a um sistema operacional, e os específicos, focados em algum módulo do sistema, como de escalonamento ou comunicação entre processos. O SOsim é um simulador genérico que cobre os principais módulos de um sistema operacional moderno.

A seguir, são apresentadas as principais ferramentas voltadas para auxiliar o ensino de sistemas operacionais:

2.4.1 BASI

O Ben-Ari Concurrent Interpreter (BASI) é um simulador que permite desenvolver programas concorrentes utilizando diversos mecanismos de sincronização, como semáforos, mutexes e monitores. Como a maioria dos sistemas operacionais multiprogramáveis precisa implementar primitivas de sincronização, geralmente as disciplinas de sistemas operacionais apresentam estes conceitos, porém, de forma superficial [BYNUM, 1996].

Sem a utilização de um simulador deste tipo, a implementação prática das técnicas de programação concorrente fica dependente de uma ou mais linguagens de programação, como Pascal Concorrente, Modula e Ada. Outra opção seria a utilização de system calls do próprio sistema operacional, o que obrigaria ao estudante entrar em detalhes específicos do sistema.

O BASI foi desenvolvido com base na versão do Pascal definido por M. Ben-Ari. Esta versão é um subconjunto do Pascal Padrão (Pascal-S), definido por Niklaus Wirth, com extensões para programação paralela, como construções cobegin/coend, wait e signal. Comparado com outras linguagens concorrentes, o BASI oferece uma variedade de técnicas de sincronização com uma sintaxe simples, permitindo que qualquer programador C ou Pascal tenha acesso ao simulador.

O ambiente do BACI pode ser compilado e utilizado em ambientes Linux, RS/6000 AIX, SunOS, SGI IRIX e DOS, com pequenas modificações nos arquivos de configuração [BYNUM, 1999].

2.4.2 NACHOS

O NACHOS (Not Another Completely Heuristic Operating System) foi desenvolvido por Tom Anderson, Wayne Christopher e Stephen Procter na Universidade de Berkeley Califórnia, com o propósito específico de ser uma ferramenta para o auxílio ao ensino de sistemas operacionais em cursos de graduação, podendo ser utilizado gratuitamente [ANDERSON, 1999].

O ambiente foi concebido para que o aluno possa estudar, compreender e adicionar componentes ao sistema, escrevendo código para a gerência de threads, gerência de arquivos, multiprogramação, memória virtual e redes. Além do simulador para o sistema operacional, o NACHOS oferece um ambiente de hardware simulado, onde o sistema operacional é suportado.

O NACHOS foi desenvolvido em C++ e está disponível para a maioria das plataformas Unix, como HP-UX, IBM-AIX, Sun Solaris, Linux e FreeBSD, além do MS-DOS. Até o momento não existe uma versão para o sistema Windows da Microsoft.

2.4.3 OSP

O OSP (Operating System Project) foi desenvolvido por Michael Kifer e Scott Smolka, no Departamento de Ciência da Computação da Universidade Suny Stony Brook, Nova York [KIFER, 1991].

O simulador consiste de vários módulos, cada um responsável por um serviço do sistema operacional, como escalonamento, tratamento de interrupções, gerência de arquivos, gerência de memória e comunicação entre processos. A partir de um gerador de projetos (OSP Project Generator) é possível montar um sistema operacional com módulos já existentes e módulos desenvolvidos pelo próprio aluno.

O OSP foi desenvolvido em Linguagem C, sendo suportado pela maioria das plataformas Unix, como SunOS, HP-UX, IBM-AIX, Linux e FreeBSD. Desta forma, para que o aluno possa interagir com o ambiente é necessário ter domínio de programação em Linguagem C em ambiente Unix, além do conhecimento de estrutura de dados.

2.4.4 MINIX

O MINIX é um sistema operacional compatível com o Unix que pode ser utilizado gratuitamente para o ensino de sistemas operacionais, pois permite que um aluno instale o sistema em um microcomputador e estude o seu código fonte para compreender seu funcionamento [TANENBAUM, 1996].

O MINIX foi desenvolvido pelo professor Andrew Tanenbaum da Vrije Universiteit em Amsterdam, Holanda. Existem versões do sistema compatíveis com a família de processadores Intel, Macintosh, Amiga, Atari e SPARC da Sun Microsystems. O

MINIX foi desenvolvido em Linguagem C, oferece suporte a multiprogramação, permite acesso multiusuário, disponibiliza diversos utilitários e bibliotecas, oferece um compilador C e suporte ao protocolo TCP/IP, dentre outras facilidades.

2.4.5 FreeBSD

O FreeBSD surgiu em 1993, na Universidade da Califórnia Berkeley, com base no sistema 4.4BSD-Lite2, sendo suportado nas plataformas Intel e Compaq Alpha. O sistema é oferecido gratuitamente e pode ser utilizado para praticamente qualquer aplicação, inclusive pedagógica [FREEBSD, 2000].

Suas principais características são:

- Multitarefa preemptiva e suporte a múltiplos usuários;
- Suporte aos protocolos da família TCP/IP;
- Esquema de gerência de memória virtual;
- Suporte a múltiplos processadores simétricos (SMP);
- Várias linguagens para desenvolvimento, como GNU C/C++, FORTRAN e Perl;
- Código fonte aberto, que permite seu estudo e/ou alteração.

2.4.6 TROPIX

O TROPIX é um sistema operacional de filosofia Unix, desenvolvido no NCE/UFRJ (Núcleo de Computação Eletrônica/Universidade Federal do Rio de Janeiro) e mantido pelos professores Pedro Salenbauch e Oswaldo Vernet [TROPIX, 2000].

Foi inicialmente concebido em 1982, com o nome de PLURIX para um computador também desenvolvido no NCE, o PEGASUS, baseado no processador Motorola 68000. A partir de 1994, foi iniciada a conversão do sistema para a plataforma Intel (386, 486 e Pentium).

O TROPIX possui diversas aplicações, tais como o ensino/aprendizado de sistemas operacionais, desenvolvimento de software e implementação de servidores Internet. A distribuição do sistema é gratuita e seus manuais estão em português.

Suas principais características são:

- Sistema multiusuário e multitarefa, com suporte a threads;
- Memória compartilhada;
- Utilitários básicos do Unix;
- Ambiente de desenvolvimento em ANSI C;
- Suporte para redes TCP/IP, SLIP e PPP;
- Serviços de rede do tipo telnet, rlogin, FTP, WWW, POP3 etc.

2.5 SOsim

O SOsim foi desenvolvido para servir de ferramenta de auxílio para o ensino de sistemas operacionais, tornando o trabalho do professor mais eficaz. Aproximando a teoria à prática, o simulador permite facilitar o entendimento e o aprendizado por parte dos alunos. O software oferece a possibilidade de ampliar a interação entre alunos e professores, em função das experiências que cada um tenha desenvolvido (Fig. 2.1).

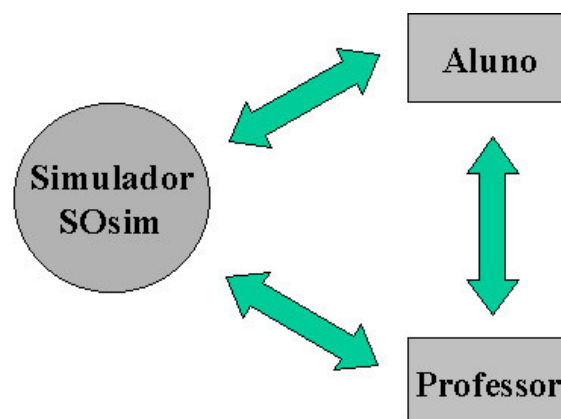


Fig. 2.1 - Interação simulador, professor e aluno

A seguir, são apresentadas as principais características e vantagens do simulador SOsim:

- Maior eficiência

O simulador permite aumentar a efetividade do processo de ensino/aprendizado, principalmente em cursos de curta duração e baixos pré-requisitos. A partir de sua interface gráfica e dos recursos de animação é possível visualizar facilmente os diversos conceitos e técnicas encontrados em um sistema operacional multiprogramável.

Além disto, o software oferece opções de configuração que permitem que diferentes algoritmos possam ser analisados e avaliados, como por exemplo o algoritmo de escalonamento utilizado. Além disso, o simulador oferece as opções de visualização de eventos através de logs e estatísticas dos diversos módulos.

Como o uso do simulador, o professor tem a chance de apresentar os conceitos e técnicas de forma mais clara a partir de animações, melhorando a comunicação com os alunos e ampliando, assim, o poder de compreensão e entendimento dos conceitos apresentados.

- Facilidade de uso

A maioria dos software utilizados para o apoio ao ensino de sistemas operacionais cria um ambiente idêntico ao mundo real. Em função do alto grau de complexidade de um sistema operacional real, há um correspondente grau de dificuldade em utilizar este tipo de software como ferramenta de suporte ao ensino.

O SOsim permite que o professor apresente a dinâmica de um sistema operacional sem necessariamente entrar em detalhes de hardware e/ou software, como é exigido pelos simuladores reais. O software não exige qualquer conhecimento de programação para ser utilizado. Sua interface gráfica é intuitiva, eliminando as barreiras tradicionais encontradas nos programas existentes. Este atributo é especialmente importante nos

cursos com ênfase em conceitos e não na implementação detalhada dos componentes de um sistema operacional.

O aluno pode interagir com o simulador de forma a criar uma situação real e verificar suas conseqüências. Por exemplo, ao criar um processo CPU-bound de alta prioridade, o sistema fica dedicado a esta tarefa até o seu término.

- Apoio presencial e não-presencial

Os professores podem utilizar o simulador em sala de aula como ferramenta de apoio às aulas presenciais ou recomendar seu uso em outros momentos, como em laboratórios nas escolas ou mesmo ser utilizado nos computadores pessoais dos próprios alunos. Neste caso, o software educacional torna-se uma extensão da sala de aula, funcionando como uma ferramenta de apoio não-presencial.

- Diversidade de público

O simulador pode ser utilizado em cursos básicos, intermediários e avançados de sistemas operacionais. Neste último caso, o professor poderá criar projetos que façam alterações nos módulos já existentes e/ou inserções de facilidades ainda não implementadas. Uma série de sugestões para projetos pode ser obtida no Capítulo 5 - Conclusões e Trabalhos Futuros.

Para que os alunos possam alterar o código fonte do simulador é necessário o conhecimento de orientação por objetos, Pascal e conhecimentos básicos de Borland Delphi [CALVERT, 1996]. Apesar da aparente dificuldade, estes conhecimentos fazem parte da maioria dos cursos correntes de Ciência da Computação, sendo um bom momento para integrar e consolidar diferentes disciplinas e áreas de conhecimento.

- Ambiente padronizado

O simulador foi concebido para ser executado no ambiente Windows da Microsoft. A escolha deste ambiente operacional deve-se ao fato de ser amplamente utilizado tanto no mundo acadêmico quanto no comercial, além de ser suportado por uma plataforma padrão de hardware. Além disto, o software pode ser executado com poucos recursos de hardware e não exige qualquer tipo de dispositivo especial.

- Facilidade de modificação e ampliação

O projeto do simulador foi desenvolvido em um ambiente orientado a objetos, simplificando em muito a compreensão do código fonte e facilitando suas futuras ampliações. O código do SOsim foi desenvolvido na linguagem Pascal, utilizando o software Borland Delphi. Este ambiente de desenvolvimento permite aos interessados em ler e/ou alterar o código fonte do simulador, uma forma mais fácil e rápida de implementar modificações, se comparado ao ambiente Unix que geralmente utiliza Linguagem C/C++.

CAPÍTULO 3 – MODELO PROPOSTO

3.1 Introdução

Um sistema operacional, por mais complexo que possa parecer, é apenas um conjunto de rotinas executado pelo processador, da mesma forma que qualquer outro programa. Sua principal função é controlar o funcionamento de um sistema computacional, gerenciando seus diversos recursos como processadores, memórias e dispositivos de entrada e saída [MACHADO, 1997].

A grande diferença entre um sistema operacional e aplicações convencionais é a maneira como suas rotinas são executadas em função do tempo. Um sistema operacional não é executado de forma linear como na maioria das aplicações, com início, meio e fim. Suas rotinas são executadas em função de eventos assíncronos, ou seja, eventos que ocorrem a qualquer momento.

Os sistemas multiprogramáveis ou multitarefa permitem que diversos programas dividam os mesmos recursos computacionais concorrentemente, como memória principal, processador e dispositivos de entrada/saída. Para cada programa, o sistema operacional aloca uma fatia de tempo (time-slice) do processador. Caso o programa não esteja concluído nesse intervalo de tempo, ele é substituído por um outro, e fica esperando por uma nova fatia de tempo. Não só o processador é compartilhado nesse sistema, mas também a memória e os periféricos. O sistema cria para cada usuário um ambiente de trabalho próprio (máquina virtual), dando a impressão de que todo o sistema está dedicado exclusivamente a ele.

As vantagens do uso de sistemas multiprogramáveis são o aumento da produtividade dos seus usuários e a redução de custos, a partir do compartilhamento dos diversos recursos do sistema. Por exemplo, enquanto um programa espera por uma operação de leitura ou gravação em disco, outros programas podem estar sendo processados neste mesmo intervalo de tempo. O sistema operacional preocupa-se em gerenciar o acesso

concorrente aos seus diversos recursos, como memória, processador e periféricos, de forma ordenada e protegida, entre os diversos programas (Fig. 3.1).

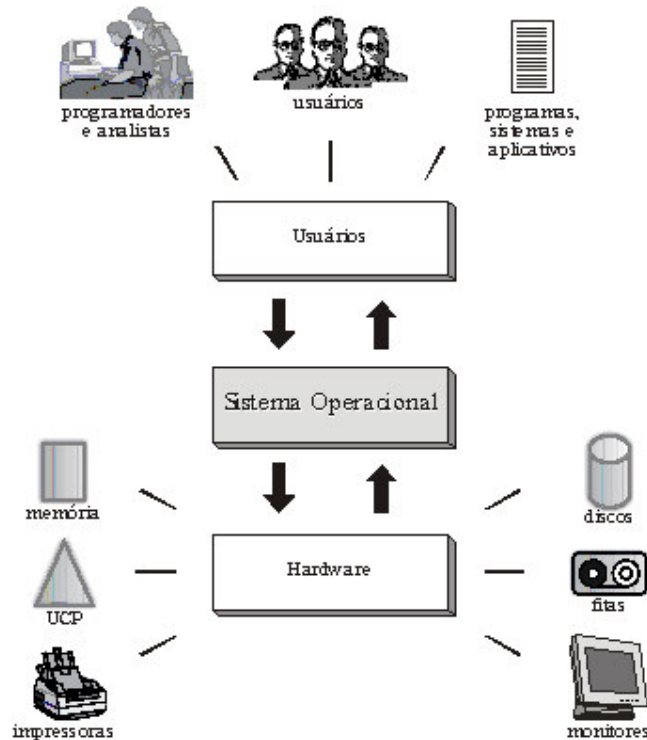


Fig. 3.1 - Sistema operacional genérico

Sistemas multiprogramáveis são de implementação complexa e para que seu projeto tenha sucesso é necessário abordar cinco áreas principais: processo, gerência de memória, proteção, escalonamento, gerência de recursos e estruturação do sistema [DENNING, 1980]. O SOsim tem como principal objetivo apresentar os conceitos e técnicas encontrados nos sistemas operacionais multiprogramados. O projeto do simulador implementa, em diferentes níveis de detalhamento, as áreas acima citadas por Denning, utilizando como base os algoritmos encontrados nos sistemas operacionais OpenVMS da Compaq [LEVY, 1980] [MILLER, 1992] e Windows 2000 da Microsoft [SOLOMON, 1998].

3.2 Processo

Um processo pode ser entendido inicialmente como um programa em execução, só que seu conceito é mais abrangente. Este conceito torna-se mais claro quando pensamos de que forma os sistemas multiprogramáveis (multitarefa) atendem os diversos usuários (tarefas) e mantêm informações a respeito dos vários programas que estão sendo executados concorrentemente [MACHADO, 1997].

Um sistema multiprogramável simula um ambiente de monoprogramação para cada usuário, isto é, cada usuário do sistema tem a impressão de possuir o processador exclusivamente para ele. Nesses sistemas, o processador executa a tarefa de um usuário durante um intervalo de tempo e, no instante seguinte, está processando outra tarefa. A cada troca é necessário que o sistema preserve todas as informações da tarefa que foi interrompida, para quando voltar a ser executada não lhe faltar nenhuma informação para a continuação do processamento. À estrutura computacional interna, responsável pela manutenção de todas as informações necessárias à execução de um programa, como conteúdo de registradores e espaço de memória, dá-se o nome processo.

O conceito de processo pode ser refinado como sendo o ambiente onde se executa um programa. A execução de um mesmo programa pode ser afetada dependendo do processo no qual ele é processado. O processo pode ser dividido em três elementos básicos: contexto de hardware, contexto de software e espaço de endereçamento, que juntos mantêm todas as informações necessárias à execução do programa (Fig. 3.2).

O contexto de hardware constitui-se basicamente no conteúdo de registradores: program counter (PC), stack pointer (SP) e bits de estado. Quando um processo está em execução, o contexto de hardware está armazenado nos registradores do processador. No momento em que o processo perde a utilização da CPU, o sistema salva o seu contexto.

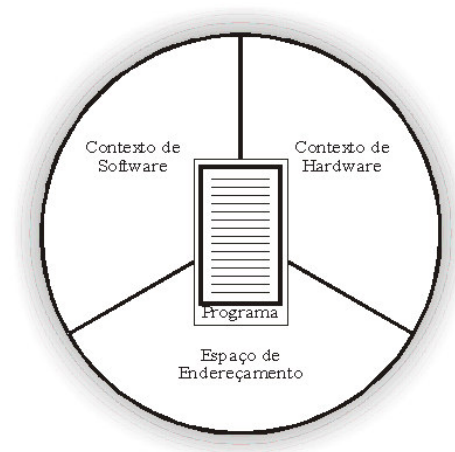


Fig. 3.2 - Processo

O contexto de hardware é fundamental para a implementação dos sistemas time-sharing, onde os processos se revezam na utilização do processador, podendo ser interrompidos e, posteriormente, restaurados como se nada tivesse acontecido. A troca de um processo por outro na CPU, realizada pelo sistema operacional, é denominada mudança de contexto (context switching). A mudança de contexto consiste em salvar o conteúdo dos registradores da CPU e carregá-los com os valores referentes ao do processo que esteja ganhando a utilização do processador. Essa operação resume-se, então, em substituir o contexto de hardware de um processo pelo de outro.

O contexto de software especifica características do processo que vão influir na execução de um programa, como o número máximo de arquivos abertos simultaneamente ou o tamanho do buffer para operações de E/S. Essas características são determinadas no momento da criação do processo, mas algumas podem ser alteradas durante sua existência. O contexto de software define basicamente três grupos de informações de um processo: sua identificação, suas quotas e seus privilégios.

O espaço de endereçamento é a área de memória do processo onde um programa poderá ser executado, além do espaço para os dados utilizados por ele. Cada processo possui seu próprio espaço de endereçamento, que deve ser protegido do acesso dos demais processos.

3.2.1 Estados do Processo

Um processo não é executado todo o tempo pelo processador. Durante sua existência, ele passa por uma série de estados. O simulador implementa três estados em que um processo pode se encontrar: execução, pronto e espera.

- Estado de execução

Um processo é dito no estado de execução (running) quando está sendo processado pela CPU. Como o simulador implementa apenas um processador, somente um processo pode estar sendo executado em um dado instante de tempo. Os processos se revezam na utilização do processador seguindo uma política estabelecida pelo escalonador.

- Estado de pronto

Um processo está no estado de pronto (ready) quando apenas aguarda uma oportunidade para executar, ou seja, espera que o sistema operacional aloque a CPU para sua execução. O escalonador é responsável por determinar a ordem pela qual os processos em estado de pronto devem ganhar a CPU. Normalmente existem vários processos no sistema no estado de pronto.

- Estado de espera

Um processo está no estado de espera (wait) quando aguarda algum evento externo ou algum recurso para poder prosseguir seu processamento. O simulador implementa três tipos de estado de espera: um processo pode estar esperando o término de uma operação de entrada/saída, aguardando ser retirado do estado de suspenso (resume) ou a espera pelo término de um page fault.

3.2.2 Mudanças de Estados do Processo

Um processo muda de estado diversas vezes durante seu processamento, em função de eventos originados por ele próprio, como uma operação de entrada/saída, ou pelo sistema operacional, como término de quantum e preempção. O simulador implementa as quatro principais mudanças de estado que podem ocorrer a um processo em um sistema operacional multiprogramável:

- Pronto → Execução

Quando um processo é criado, o sistema o coloca em uma lista de processos no estado de pronto, onde aguarda uma oportunidade para ser executado (Fig. 3.3a). O simulador oferece três opções de escalonamento, apresentadas posteriormente neste capítulo.

- Execução → Espera

Um processo em execução passa para o estado de espera por eventos gerados pelo próprio processo (Fig. 3.3b). Nesse caso, o processo ficará neste estado esperando pela conclusão do evento solicitado. O simulador coloca um processo no estado de espera em três situações: quando o processo faz uma operação de entrada/saída, gera um page fault ou é suspenso pelo usuário.

- Espera → Pronto

Um processo no estado de espera passa para o estado de pronto quando a operação solicitada é atendida ou o recurso esperado é concedido. Um processo no estado de espera sempre terá de passar pelo estado de pronto antes de poder ser novamente selecionado para execução (Fig. 3.3c).

- Execução → Pronto

Um processo em execução passa para o estado de pronto por eventos gerados pelo sistema. O simulador implementa dois eventos deste tipo: o fim da fatia de tempo (time-slice) que o processo possui para sua execução (quantum-end) e preempção (Fig. 3.3d). Nestes casos, o processo volta para a fila de pronto, onde aguardará por uma nova oportunidade para continuar seu processamento.

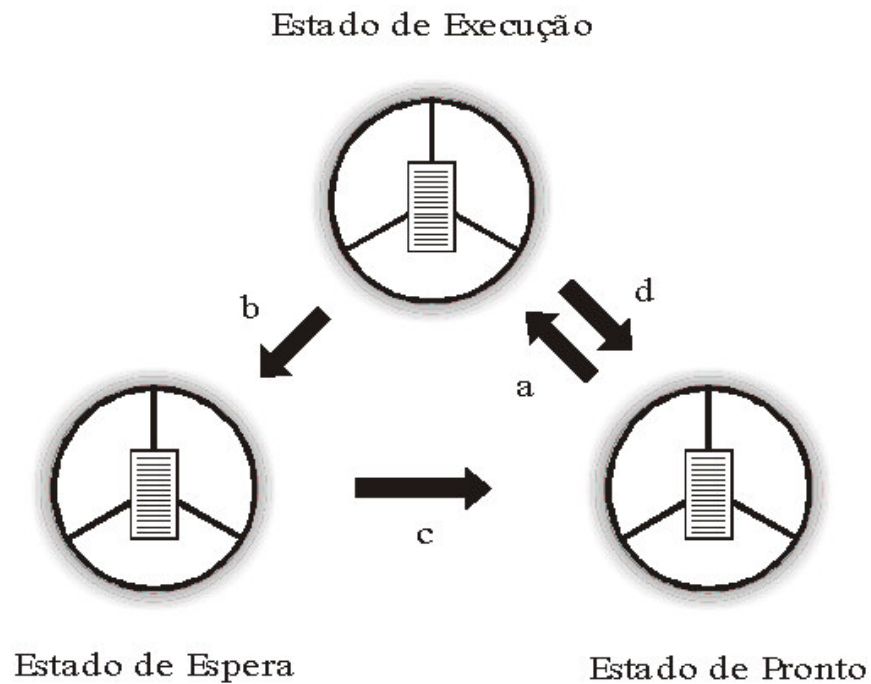


Fig. 3.3 - Mudanças de estado do processo

3.2.3 Sinais

Um sinal está para um processo, assim como as interrupções estão para o sistema operacional. Os sinais permitem notificar os processos de eventos síncronos e assíncronos, sendo fundamentais para a gerência, comunicação e sincronização de processos [MAIA, 1999].

A geração de um sinal ocorre quando o sistema operacional notifica o processo através de bits de sinalização localizados no seu PCB. Por exemplo, quando um processo é eliminado, o sistema ativa o bit correspondente. No momento em que o processo é escalonado, o bit é testado e o processo eliminado.

O SOsim utiliza sinais para a eliminação e suspensão de processos, de forma semelhante implementada pelos sistemas operacionais.

3.3 Gerência do Processador

O conceito básico que gerou a implementação dos sistemas multiprogramáveis foi a possibilidade de a CPU ser compartilhada entre diversos processos. Para isso, todo sistema multiprogramável possui um critério para determinar qual a ordem na escolha dos processos para execução entre os vários que concorrem pela utilização do processador [MACHADO, 1997].

O procedimento de seleção é uma das principais funções realizadas por um sistema operacional, sendo conhecido como escalonamento (scheduling). A parte do código do sistema operacional responsável pelo escalonamento é chamada de escalonador (scheduler).

Os principais objetivos do escalonamento são, basicamente, manter a CPU ocupada a maior parte do tempo, balancear a utilização do processador entre os diversos processos, maximizar o throughput do sistema e oferecer tempos de resposta razoáveis para os usuários interativos. Esses objetivos devem ser atendidos de forma que o sistema trate todos os processos igualmente, evitando assim que um processo fique indefinidamente esperando pela utilização do processador (starvation). Para atender alguns desses objetivos, muitas vezes conflitantes, os sistemas operacionais devem levar em consideração características dos processos, ou seja, se um processo é do tipo batch, interativo, CPU-bound ou I/O-bound.

O algoritmo principal implementado pelo simulador é muito semelhante ao dos sistemas operacionais OpenVMS da Compaq e Windows 2000 da Microsoft, e foi escolhido por permitir a implementação de diversas opções de escalonamento.

3.3.1 Escalonamento Não-Preemptivo

Nos primeiros sistemas multiprogramáveis, onde predominava tipicamente o processamento batch, o escalonamento implementado era do tipo não-preemptivo. Nesse tipo de escalonamento, quando um processo (ou job) ganha o direito de utilizar a CPU, nenhum outro processo pode lhe tirar esse recurso.

Nesse esquema, o processo que chegar primeiro (first-in) é o primeiro a ser selecionado para execução (first-out) e quando um processo ganha o processador, ele utilizará a CPU sem ser interrompido. Seu algoritmo de implementação é bastante simples, sendo necessária apenas uma fila, onde os processos que passam para o estado de pronto entram no seu final e são escalonados quando chegarem ao seu início.

O problema do escalonamento FIFO é a impossibilidade de se prever quando um processo terá sua execução iniciada, já que isso varia em função do tempo de execução dos demais processos que se encontram na sua frente. Outro problema é a possibilidade de processos CPU-bound de menor importância prejudicarem processos I/O-bound mais prioritários.

Essa política de escalonamento foi inicialmente implementada em sistemas batch, sendo ineficiente, se aplicada da forma original em sistemas de tempo compartilhado. Atualmente, sistemas de tempo compartilhado utilizam o escalonamento FIFO com variações, permitindo, assim, sua implementação.

3.3.2 Escalonamento Preemptivo

Um algoritmo de escalonamento é dito preemptivo quando o sistema pode interromper um processo em execução para que outro processo utilize o processador. O

escalonamento preemptivo permite que o sistema dê atenção imediata a processos mais prioritários, além de proporcionar melhores tempos de resposta em sistemas de tempo compartilhado. Outro benefício decorrente deste tipo de escalonamento é o compartilhamento do processador de uma maneira mais uniforme entre os processos.

Vejamos, a seguir, os dois algoritmos de escalonamento preemptivos implementados pelo simulador:

- Escalonamento por tempo

No escalonamento por tempo sempre que um processo passa para o estado de execução, ou seja, ganha a CPU, existe um tempo-limite para sua utilização de forma contínua, denominado time-slice ou quantum. Quando esse tempo, expira, sem que antes a CPU seja liberada pelo processo, este volta ao estado de pronto, dando a vez para outro processo. Esse mecanismo é definido como preempção por tempo.

A fila de processos em estado de pronto é tratada como uma fila circular. O escalonamento é realizado, alocando a CPU para cada processo da fila no intervalo de tempo determinado pelo quantum. O valor do quantum de tempo pode ser alterado dinamicamente pelo usuário durante a execução da simulação.

Através do escalonamento por tempo, nenhum processo poderá monopolizar a CPU, sendo o tempo máximo alocado continuamente para um determinado processo igual ao quantum. No caso de sistemas de tempo compartilhado, onde vários usuários utilizam o sistema concorrentemente, esse algoritmo é bastante adequado.

- Escalonamento por prioridades

O escalonamento por tempo consegue melhorar a distribuição do tempo de CPU em relação ao escalonamento não-preemptivo, porém ainda não consegue implementar um compartilhamento equitativo entre os diferentes tipos de processos. Isso acontece em

razão de o escalonamento por tempo tratar todos os processos de uma maneira igual, o que nem sempre é desejável.

Um processo CPU-bound leva vantagem, na utilização da CPU, sobre o processo I/O-bound. Como, no escalonamento por tempo, um processo I/O-bound compete pelo processador da mesma forma que um processo CPU-bound, e o processo I/O-bound passa a maior parte do tempo no estado de espera, o processo CPU-bound tem mais chance de ser executado.

Para solucionar esse problema, os processos I/O-bound devem levar alguma vantagem no escalonamento, a fim de compensar o excessivo tempo gasto no estado de espera. Como alguns processos devem ser tratados de maneira diferente dos outros, é preciso associar a cada um deles uma prioridade de execução. Nesse esquema, processos de maior prioridade são escalonados preferencialmente. Toda vez que um processo for para a fila de pronto com prioridade superior ao do processo em execução, o sistema deverá interromper o processo corrente, colocá-lo no estado de pronto e selecionar o de maior prioridade para ser executado. Esse mecanismo é definido como preempção por prioridade.

Todos os sistemas de tempo compartilhado implementam algum esquema de prioridade, de forma a dar maior importância a um processo no momento do escalonamento. A prioridade é uma característica do contexto de software de um processo, podendo ser estática ou dinâmica. O simulador oferece a possibilidade de prioridade estática ou dinâmica, através das opções da gerência do processador.

A prioridade é dita estática quando não é modificada durante a existência do processo. Apesar da simplicidade de implementação, a prioridade estática pode ocasionar tempos de resposta elevados. Na prioridade dinâmica, a prioridade do processo pode ser ajustada de acordo com o tipo de processamento realizado pelo processo e/ou a carga do sistema. Todo o processo, ao sair do estado de espera, recebe um acréscimo à sua prioridade. Dessa forma, os processos I/O-bound terão mais chance de ser escalonados e, assim, compensar o tempo que passam no estado de espera. É importante perceber

que os processos CPU-bound não são prejudicados, pois podem ser executados enquanto os processos I/O-bound esperam por algum evento.

- Escalonamento por múltiplas filas com realimentação

O escalonamento por múltiplas filas com realimentação (multi-level feedback queues) implementa diversas filas, onde cada qual tem associada uma prioridade de execução, porém os processos não permanecem em uma mesma fila até o término do processamento. Neste escalonamento, o sistema tenta identificar dinamicamente o comportamento de cada processo, ajustando assim suas prioridades de execução. O escalonamento por múltiplas filas com realimentação é um algoritmo de escalonamento generalista, podendo ser implementado em qualquer tipo de sistema operacional.

Esse esquema permite que os processos sejam redirecionados entre as filas do sistema, fazendo com que o sistema operacional implemente um mecanismo de ajuste dinâmico, denominado mecanismo adaptativo, que tem como objetivo ajustar os processos em função do comportamento do sistema. Os processos não são previamente associados às filas de pronto, e sim direcionados pelo sistema entre as diversas filas com base no seu comportamento.

Um processo, ao ser criado, entra no final da fila de mais alta prioridade. Cada fila implementa o mecanismo de FIFO para escalonamento. Quando um processo em execução deixa a CPU, seja por preempção por prioridade ou por solicitação a um recurso do sistema, ele é reescalonado dentro da mesma fila. Caso o processo esgote seu quantum de tempo, ele é redirecionado para uma fila de menor prioridade (preempção por tempo). O escalonamento de um processo em uma fila só acontece quando todas as outras filas de prioridades mais altas estão vazias. A fila de mais baixa prioridade implementa o mecanismo do escalonamento circular.

Essa política de escalonamento atende as necessidades dos diversos tipos de processos. No caso de processos I/O-bound, ela oferece um bom tempo de resposta, já que esses processos têm prioridades altas por permanecerem a maior parte do tempo nas filas de

mais alta ordem. No caso de processos CPU-bound, a tendência é que, ao entrar na fila de mais alta prioridade, o processo ganhe o processador, gaste seu quantum de tempo e seja direcionado para uma fila de menor prioridade. Dessa forma, quanto mais tempo um processo utiliza do processador, mais ele vai caindo para filas de menor prioridade.

3.3.3 Critérios de Escalonamento

Um algoritmo de escalonamento tem como principal função decidir qual dos processos prontos para execução deve ser alocado à CPU. Cada sistema operacional necessita de um algoritmo de escalonamento adequado a seu tipo de processamento. A seguir, apresentaremos os principais critérios de escalonamento:

- Utilização da CPU

Na maioria dos sistemas é desejável que o processador permaneça a maior parte do seu tempo ocupado. Uma utilização na faixa de 30% indica um sistema com uma carga de processamento baixa, enquanto que na faixa de 90% indica um sistema bastante carregado, próximo da sua capacidade total.

- Throughput

O throughput representa o número de processos (tarefas) executados em um determinado intervalo de tempo. Quanto maior o throughput, maior o número de tarefas executadas em função do tempo. A maximização do throughput é desejada na maioria dos sistemas.

- Tempo de turnaround

Tempo que um processo leva desde sua admissão no sistema até ao seu término, levando em consideração o tempo de espera para alocação de memória, espera na fila de processos prontos para execução, processamento na CPU e operações de E/S.

- Tempo de resposta

Em sistemas interativos, o tempo de resposta é o tempo decorrido do momento da submissão de um pedido ao sistema até a primeira resposta produzida. O tempo de resposta não é o tempo utilizado no processamento total de uma tarefa, e sim o tempo decorrido até que uma resposta seja apresentada. Este tempo, geralmente, é limitado pela velocidade do dispositivo de saída.

De uma maneira geral, qualquer algoritmo de escalonamento busca otimizar a utilização da CPU e o throughput, enquanto tenta diminuir os tempos de turnaround e de resposta. Dependendo do tipo do sistema, um critério pode ser mais enfatizado do que outros, como, por exemplo, nos sistemas interativos, onde o tempo de resposta deve ser mais considerado.

O algoritmo de escalonamento não é o único responsável pelo tempo de execução de um processo. Outros fatores, como o tempo de processamento e de espera em operações de E/S, devem ser considerados no tempo total da execução. O escalonamento somente afeta o tempo de espera de processos na fila de pronto.

3.4 Gerência de Memória

Na memória principal residem todos os programas e dados que serão executados ou referenciados pelo processador. Um programa residente na memória secundária para ser executado deve ser, de alguma forma, carregado para a memória principal. A organização e gerência da memória principal têm sido fatores importantes no projeto de sistemas operacionais. Enquanto nos sistemas monoprogramáveis a gerência da memória não é muito complexa, nos sistemas multiprogramáveis ela torna-se crítica. Isso ocorre devido à necessidade de se manter o maior número de processos possível utilizando a memória eficientemente, tornando sua gerência muito mais difícil [MACHADO, 1997].

Existem inúmeros mecanismos de gerência de memória, como alocação contígua, esquemas de overlay, alocação particionada estática e dinâmica, e memória virtual. Mesmo a gerência de memória virtual pode ser implementada utilizando-se paginação, segmentação ou uma mistura de ambos. No projeto do SOSim implementou-se apenas a gerência de memória virtual com paginação, que será detalhada neste item.

Os mecanismos anteriores à memória virtual não foram contemplados neste projeto por servirem apenas para apresentar a evolução histórica da gerência de memória, pois, na prática, não são mais implementados na maioria dos sistemas operacionais comerciais, com exceção de alguns supercomputadores, por questões de desempenho. A gerência de memória virtual com segmentação pura é muito pouco utilizada. Geralmente, os sistemas que utilizam segmentação também implementam paginação, em um modelo híbrido. Este último modelo é recomendado para trabalhos futuros.

O algoritmo de memória virtual implementado é muito semelhante ao dos sistemas operacionais OpenVMS da Compaq e Windows 2000 da Microsoft, e foi escolhido por permitir a implementação de diversos conceitos importantes, como working set.

3.4.1 Memória Virtual

Memória virtual (virtual memory) é uma técnica sofisticada e poderosa de gerência de memória, onde as memórias principal e secundária são combinadas, dando ao usuário a ilusão de existir uma memória muito maior que a memória principal [DENNING, 1970].

O conceito de memória virtual está baseado em desvincular o endereçamento feito pelo programa dos endereços físicos da memória principal. Assim, os programas e suas estruturas de dados deixam de estar limitados ao tamanho da memória primária disponível. Para permitir que apenas partes realmente necessárias à execução do processo estejam na memória, o código deve ser dividido em blocos e mapeados na memória principal, a partir do espaço de endereçamento virtual. O espaço de endereçamento virtual representa o conjunto de endereços virtuais que os processos

podem endereçar. Analogamente, o conjunto de endereços reais é chamado espaço de endereçamento real.

O espaço de endereçamento virtual não tem nenhuma relação direta com os endereços no espaço real. Um programa pode fazer referência a endereços virtuais que estejam fora dos limites do espaço real, ou seja, os programas e suas estruturas de dados não estão mais limitados ao tamanho da memória física disponível. Como os programas podem ser muito maiores que a memória física, apenas parte deles pode estar residente na memória em um determinado instante. O sistema operacional utiliza a memória secundária como extensão da memória principal e o transporte de programas entre uma e outra dá-se de maneira dinâmica e transparente ao usuário. Quando um programa é executado, só uma parte do código fica residente na memória principal, permanecendo o restante na memória secundária até o momento de ser referenciado (Fig. 3.4).

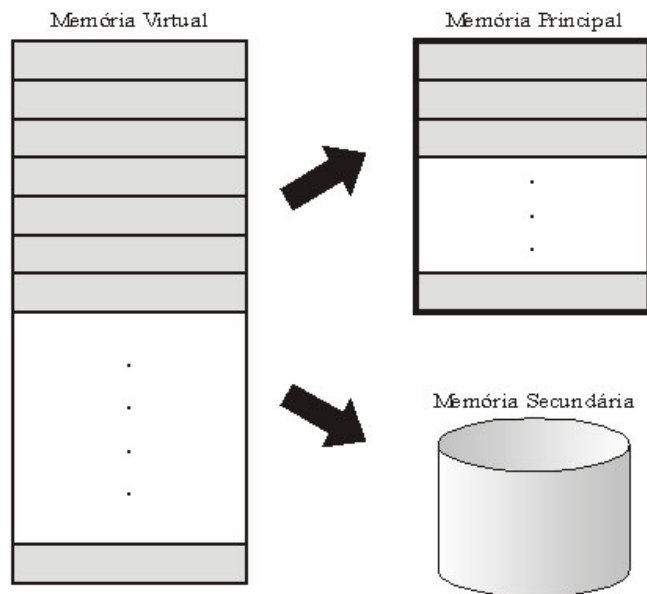


Fig. 3.4 - Espaço de endereçamento virtual

Outra vantagem da memória virtual é permitir um número maior de processos compartilhando a memória, já que apenas algumas partes de cada processo estarão residentes. Isto leva a uma utilização mais eficiente também do processador, permitindo um maior número de processos no estado de pronto.

3.4.2 Mapeamento

O mapeamento permite ao sistema operacional traduzir um endereço localizado no espaço de endereçamento virtual do processo para um endereço no espaço real. Como consequência do mapeamento, um programa não precisa estar necessariamente contíguo na memória principal para ser executado.

Cada processo tem o mesmo espaço de endereçamento virtual, como se possuísse sua própria memória virtual. O mecanismo de tradução se encarrega de manter tabelas de mapeamento exclusivas para cada processo, relacionando os endereços virtuais do processo às suas posições na memória física (Fig. 3.5). Quando um programa está sendo executado, o sistema, para realizar a tradução, utiliza a tabela de mapeamento do processo no qual o programa executa. Se um outro programa vai ser executado no contexto de outro processo, o sistema deve passar a referenciar a tabela do novo processo.

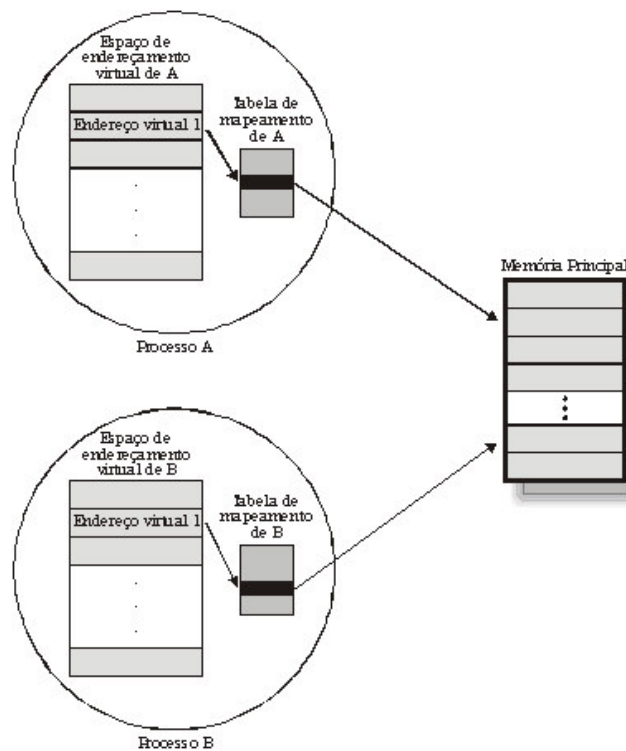


Fig. 3.5 - Tabela de mapeamento

Neste esquema, como cada processo tem a sua própria tabela de mapeamento e a tradução dos endereços é realizada pelo sistema, é garantida a proteção dos espaços de endereçamento dos processos, a menos que haja compartilhamento explícito de memória (shared memory).

Caso o mapeamento fosse realizado para cada célula na memória principal, o espaço ocupado pelas tabelas na memória real seria tão grande quanto o espaço de endereçamento virtual de cada processo, o que inviabilizaria a implementação do mecanismo de memória virtual. Em função disso, as tabelas mapeiam blocos de informações, cujo tamanho determina o número de entradas existentes nas tabelas de mapeamento. Quanto maior o bloco, menos entradas nas tabelas de mapeamento e, conseqüentemente, tabelas de mapeamento que ocupam um espaço de memória menor.

Existem sistemas que trabalham apenas com blocos do mesmo tamanho (paginação), outros que utilizam blocos de tamanhos diferentes (segmentação) e, ainda, há sistemas que trabalham com os dois tipos de blocos (segmentação com paginação). O simulador implementa apenas a gerência de memória virtual com paginação.

3.4.3 Paginação

Paginação (paging) é a técnica de gerência de memória onde o espaço de endereçamento virtual e o espaço de endereçamento real são divididos em blocos do mesmo tamanho, chamados páginas. As páginas no espaço virtual são denominadas páginas virtuais (virtual pages), enquanto as páginas no espaço real são chamadas de páginas reais (frames).

Todo o mapeamento é realizado em nível de página, através de tabelas de páginas. Cada página virtual do processo possui uma entrada na tabela de páginas (Page Table Entry - PTE), com informações de mapeamento que permitem ao sistema localizar a página real correspondente na memória principal (Fig. 3.6).

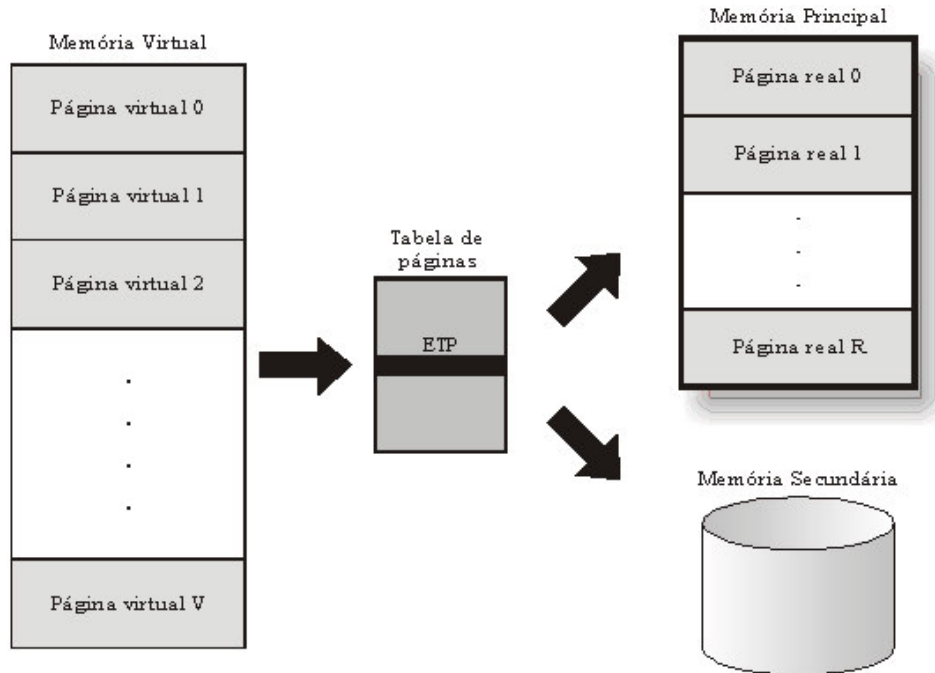


Fig. 3.6 - Tabela de páginas

Quando um programa é executado, as páginas virtuais são transferidas da memória secundária para a memória principal e colocadas em frames. Sempre que o programa fizer referência a um endereço virtual, o mecanismo de mapeamento localizará, no PTE da tabela do processo, o endereço físico do frame.

Além da informação sobre a localização da página virtual, o PTE possui outras informações, entre elas o bit de validade (valid bit), que indica se uma página está ou não na memória física. Se o bit tem o valor 0, indica que a página virtual não está na memória principal, enquanto, se for igual a 1, a página está localizada na memória.

Sempre que o processo faz referência a um endereço virtual, o simulador verifica, através do bit de validade, se a página que contém o endereço referenciado está ou não na memória principal. Caso não esteja, dizemos que ocorreu um page fault e, neste caso, o sistema deve transferir a página da memória secundária para a memória física.

Quando um processo faz referência a um endereço e ocorre um page fault, o processo é retirado do processador e colocado em estado de espera, até que a página seja lida do disco. Depois da leitura da página em disco, o processo é recolocado na fila de processos pronto e quando for reescalonado poderá continuar seu processamento.

3.4.4 Políticas de Busca de Páginas

O mecanismo de memória virtual permite a execução de um programa sem que esteja completamente residente na memória. A política de busca de páginas (fetch policy) determina quando uma página deve ser trazida para a memória principal. Existem, basicamente, duas alternativas: paginação por demanda e paginação antecipada.

Na paginação por demanda (demand paging), as páginas dos processos são transferidas da memória secundária para a principal apenas quando são referenciadas. Este mecanismo é conveniente, na medida em que leva para a memória principal apenas as páginas realmente necessárias à execução do programa. Desse modo, é possível que partes do programa, como rotinas de tratamento de erros, nunca sejam carregadas para a memória.

Na paginação antecipada (anticipatory paging ou prepaging), o sistema traz para a memória, além das páginas referenciadas, outras páginas que podem ou não ser necessárias ao processo no futuro. No caso do processo não precisar das páginas trazidas antecipadamente, o sistema terá perdido tempo de processador e ocupado memória principal desnecessariamente.

O simulador implementa o mecanismo de paginação por demanda e o mecanismo de paginação antecipada. A paginação antecipada é opcional e implementada apenas na criação de um processo.

3.4.5 Política de Alocação de Páginas

A política de alocação de páginas determina quantos frames cada processo pode alocar na memória principal. Existem, basicamente, duas alternativas: a alocação fixa e alocação variável. O simulador implementa a política de alocação de páginas fixa, mas permite que este valor seja definido na criação do processo.

Na política de alocação fixa, cada processo recebe um número máximo de páginas que pode ser utilizado. Se o número de páginas for insuficiente, o processo gera um page fault e pede uma página para obter uma nova. O número máximo de páginas pode ser igual para todos os processos ou ser definido individualmente. Alocar o mesmo número de páginas para todos os processos, apesar de justo, em princípio não funciona, caso os processos tenham necessidades diferentes de memória, como geralmente acontece. Se cada processo pode ter um número máximo de páginas, o limite pode ser definido com base no tipo da aplicação (interativa ou batch), no início da sua execução.

Na política de alocação variável, o número máximo de páginas alocadas ao processo pode variar durante sua execução, em função de sua taxa de paginação, por exemplo. A taxa de paginação é o número de page faults por unidade de tempo de um processo. Processos com elevadas taxas de paginação podem receber frames adicionais a fim de reduzi-las, ao mesmo tempo que processos com taxas baixas de paginação podem cedê-las. Este mecanismo, apesar de mais flexível, exige que o sistema operacional monitore o comportamento dos processos, provocando maior overhead.

3.4.6 Working Set

O mecanismo de memória virtual apesar de suas vantagens, introduz um grande problema. Sempre que um processo faz referência a uma de suas páginas e esta não se encontra na memória (page fault), exige do sistema operacional pelo menos uma operação de E/S, que, quando possível, deve ser evitada.

Qualquer sistema que implementa paginação deve se preocupar em manter na memória principal um certo número de páginas que reduza ao máximo a taxa de paginação dos processos, ao mesmo tempo que não prejudique os demais processos que desejam ter acesso à memória. O conceito de working set surgiu a partir da análise da taxa de paginação dos processos. Quando um programa começa a ser executado, percebe-se uma elevada taxa de page faults, que se estabiliza com o decorrer da execução. Esse fato está ligado ao princípio da localidade.

Localidade pode ser definido como a tendência que existe em um programa de fazer referências a posições de memória de forma quase uniforme, ou seja, a instruções e dados próximos. Isso significa que um processo tenderá a concentrar suas referências em um mesmo conjunto de instruções e dados na memória principal, durante um determinado período de tempo. No caso do OSim a questão da localidade é muito forte e pode ser muito bem observada. Todos os programas executados pelo simulador estão em um loop infinito de cinco páginas. Este modelo é explicado no Capítulo 4 – Arquitetura e Implementação.

O working set de um processo é o conjunto de páginas referenciadas por ele durante determinado intervalo de tempo. Uma outra definição seria que o working set é o conjunto de páginas constantemente referenciadas pelo processo, devendo permanecer na memória principal para que ele execute de forma eficiente. Caso contrário, o processo poderá sofrer com a elevada taxa de paginação (thrashing), comprometendo seu desempenho [DENNING, 1968] [DENNING, 1970] [DENNING, 1980].

Quando um processo é criado, todas as suas páginas estão na memória secundária. À medida que acontecem referências às páginas virtuais, elas são transferidas para o working set do processo na memória principal (page in). Sempre que um processo faz referência a uma página, o sistema verifica se a página já se encontra no working set do processo. Caso a página não se encontre no working set, ocorrerá o page fault. O working set do processo deve ter um limite máximo de páginas permitidas. Quanto maior o working set, menor a chance de ocorrer uma referência a uma página que não esteja na memória principal (page fault).

O simulador implementa algo semelhante ao conceito de working set, muito próximo ao utilizado nos sistemas OpenVMS e Windows 2000. Este conceito é melhor examinado no Capítulo 4 – Arquitetura e Implementação.

3.4.7 Políticas de Substituição de Páginas

O maior problema na gerência de memória virtual por paginação não é decidir que página carregar para a memória, mas quais páginas remover. Quando não existem páginas livres disponíveis na memória e novos frames devem ser alocados, a política de substituição (replacement policy) de páginas determina, dentre as diversas páginas residentes, quais devem ser realocadas.

Qualquer estratégia de substituição de páginas deve considerar se uma página foi ou não modificada, antes de liberá-la, caso contrário, possíveis dados armazenados na página serão perdidos. Sempre que o sistema liberar uma página desse tipo, ele antes deverá gravá-la na memória secundária (page out), preservando seu conteúdo. O sistema mantém um arquivo de paginação (page file) onde as páginas modificadas são armazenadas. Sempre que uma destas páginas for novamente referenciada, ela será trazida novamente para a memória principal.

O sistema consegue implementar esse mecanismo através do bit de modificação (dirty ou modify bit), que existe na entrada de cada tabela de páginas. Sempre que uma página é alterada, o valor do bit de modificação é alterado de 0 para 1, indicando que a página foi modificada. No caso de páginas que não são modificadas, como páginas de código, existem as páginas originais no arquivo executável armazenado na memória secundária, que podem ser utilizadas sempre que necessárias. Tais páginas, quando liberadas, não causam o overhead de gravação em disco.

A política de substituição pode ser classificada conforme seu escopo, ou seja, local ou global. Na política local, apenas as páginas do processo que gerou o page fault são candidatas a realocação. Já na política global, todas as páginas residentes são avaliadas,

independente do processo que gerou o page fault. O simulador implementa uma política de substituição local.

Independente se a política seja local ou global, os algoritmos de substituição de páginas devem ter o objetivo de selecionar aquelas que tenham poucas chances de serem utilizadas novamente num futuro próximo. Quanto mais elaborado e sofisticado é o algoritmo, maior também é o overhead para o sistema.

Existem diversos algoritmos na literatura voltados para a implementação da política de substituição de páginas, como FIFO (First-In-First-Out), buffer de páginas, LRU (Least-Recently-Used), LFU (Least-Frequently-Used) e NRU (Not-Recently-Used) [TANENBAUM, 1992]. O simulador implementa o algoritmo FIFO modificado com buffer de páginas, utilizado nos sistemas OpenVMS e Windows 2000. Os algoritmos LRU, LFU, dentre outros, não são implementados na prática em sistemas operacionais modernos. O único algoritmo que é implementado na grande maioria dos sistemas Unix, é o FIFO circular (clock). Este algoritmo é recomendado para trabalhos futuros. A seguir, analisaremos as políticas de substituição de páginas implementadas pelo simulador:

- **First-In-First-Out (FIFO)**

Nesse esquema, a página que primeiro foi utilizada (first-in) será a primeira a ser escolhida (first-out), ou seja, o algoritmo seleciona a página mais antiga na memória. Sua implementação pode ser feita associando a cada página o momento que foi trazida para a memória ou utilizando uma fila, onde as páginas mais antigas estão no início e as mais recentes no final da fila.

Parece razoável pensar que uma página que esteja mais tempo na memória seja justamente aquela que deva ser selecionada. Caso uma página seja constantemente referenciada, o fator tempo torna-se irrelevante, e o sistema tem que retornar a mesma página várias vezes. O algoritmo FIFO raramente é utilizado sem algum outro mecanismo que minimize este problema, como o algoritmo de Buffer de Páginas.

- **Buffer de Páginas**

O buffer de páginas utiliza como base o algoritmo FIFO, combinado com uma lista de páginas livres (Free Page List – FPL), que funciona em um esquema de fila. Sempre que um processo solicita uma nova página, a página que está a mais tempo no seu working set é colocada no final da FPL. Para atender a solicitação do processo, a primeira página da FPL é selecionada. Além da FPL, o simulador implementa uma segunda lista, conhecida como lista de páginas modificadas (Modified Page List – MPL), que armazena as páginas modificadas que são retiradas do working set do processo.

É importante notar que a página substituída continua fisicamente na memória, logo, se a página for novamente referenciada, basta trazê-la da FPL ou MPL, que funciona como um buffer de páginas. O buffer de páginas permite criar um algoritmo de substituição de páginas eficiente, sem o overhead de outras implementações.

3.4.8 Swapping

A técnica de swapping permite aumentar o número de processos compartilhando a memória principal e, conseqüentemente, o sistema. Em sistemas que implementam essa técnica, quando existem novos processos que desejam ser processados e não existe memória real suficiente, o sistema seleciona um ou mais processos que deverão sair da memória para ceder espaço aos novos processos.

Há vários critérios que podem ser aplicados na escolha do(s) processo(s) que deve(m) sair da memória. Os mais utilizados são a prioridade e o estado do processo. O critério de estado seleciona os processos que estão no estado de espera, ou seja, aguardando por algum evento. O critério de prioridade escolhe, entre os processos, os de menor prioridade de execução.

Depois de escolhido o(s) processo(s), o sistema intervém e ativa uma rotina do sistema responsável por retirar (swap out) e trazer (swap in) os processos da memória principal para a memória secundária, onde são gravados em um arquivo de swapping (swap file).

No simulador, os arquivos de swapping e paginação são implementados como um único arquivo em disco para facilitar a visualização dos eventos relacionados.

3.4.9 Thrashing

Thrashing pode ser definido como sendo a excessiva transferência de páginas entre a memória principal e a memória secundária. No modelo proposto, thrashing pode ocorrer em dois níveis: em nível do próprio processo e em nível do sistema.

Em nível do processo, a excessiva paginação ocorre devido ao elevado número de page faults, gerado pelo programa em execução. Esse problema faz com que o processo passe mais tempo esperando por páginas do que realmente sendo executado e ocorre devido ao mau dimensionamento no tamanho do working set do processo, pequeno demais para acomodar as páginas constantemente referenciadas por ele.

Em nível do sistema, o thrashing ocorre quando existem mais processos competindo por memória real que espaço disponível. Neste caso, o sistema tenta administrar a memória de forma que todos os processos sejam atendidos, descarregando processos para a memória secundária e carregando processos para a memória principal. Se esse mecanismo for levado ao extremo, o sistema passará mais tempo fazendo swapping do que executando processos.

De qualquer forma, se persistem mais processos para serem executados que memória real disponível, a solução que realmente restaura os níveis de desempenho adequados é a expansão da memória principal. É importante ressaltar que este problema não ocorre apenas em sistemas que implementam memória virtual, mas também em sistemas com outros mecanismos de gerência de memória.

CAPÍTULO 4 – ARQUITETURA E IMPLEMENTAÇÃO

4.1 Introdução

Este capítulo aborda a arquitetura do simulador e sua implementação, com base nos conceitos e mecanismos discutidos no Capítulo 3 - Modelo Proposto. Além disto, são apresentando o ambiente de trabalho e uma visão geral dos objetos que fazem parte de sua estrutura interna. O capítulo enfatiza especialmente a implementação de processos, dos módulos de gerência do processador e gerência de memória.

Uma preocupação no desenvolvimento deste capítulo foi de não entrar em detalhes da implementação do código fonte do software, mas sim de oferecer uma abordagem top-down do projeto e mostrar a utilização da ferramenta educacional. É importante ressaltar também que a implementação do simulador não segue, necessariamente, as mesmas estruturas de dados e procedimentos internos de um sistema operacional real.

4.2 Orientação por Objetos

O SOsim foi inteiramente implementado em uma linguagem de programação orientada a objetos (OO), o Borland Delphi. A opção pela orientação por objetos foi, principalmente, permitir a redução da complexidade no desenvolvimento do software e aumento da produtividade. No entanto, é interessante perceber que poucos sistemas operacionais são implementados na prática com base nos conceitos de OO, apesar de todas vantagens existentes. Alguns sistemas, como o Windows 2000, adotam esta filosofia em certos níveis em função da maior manutabilidade do sistema e facilidade de ampliação de suas funcionalidades [SOLOMON, 1998].

Um objeto é uma abstração de software que pode representar algo real ou virtual, sendo formado por um conjunto de propriedades (variáveis) e procedimentos (métodos). As variáveis possuem um tipo, que define os possíveis valores que a variável pode representar, como um número inteiro, número real ou string. Os métodos são rotinas que, quando executadas, realizam alguma tarefa, como alterar o conteúdo de uma

variável do objeto. Objetos comunicam-se apenas através de mensagens e o conjunto de mensagens a que um objeto pode responder é definido como protocolo de comunicação [BOOCH, 1994].

Um exemplo de objeto no SOsim é a implementação do conceito de processo. O objeto Processo possui propriedades, como identificação, prioridade e quotas. Além disto, um processo possui procedimentos associados, como criar, eliminar, alterar a prioridade e visualizar suas características.

As variáveis de um objeto só podem ser alteradas por métodos definidos na própria classe. A única maneira de um objeto alterar as variáveis de um outro objeto é a através da ativação de um de seus métodos por uma mensagem. Este conceito, onde variáveis e métodos são visíveis apenas através de mensagens, é conhecido como encapsulamento. O encapsulamento funciona como uma proteção para as variáveis e métodos, além de tornar explícito qualquer tipo de comunicação com o objeto. Os métodos que permitem a comunicação do objeto com o mundo exterior são conhecidos como interfaces públicas (public).

Uma classe consiste de variáveis e métodos que representam características de um conjunto de objetos semelhantes. O conceito de classe é um dos pilares da programação orientada a objetos, por permitir a reutilização efetiva de código.

O conceito de herança permite definir uma nova classe com base em uma já existente. A classe criada (subclasse ou classe derivada) automaticamente herda todas as variáveis e métodos da classe já existente (superclasse). O mecanismo de herança permite ainda que a subclasse inclua ou sobreponha novas variáveis e métodos da superclasse. O mecanismo de herança é recursivo, permitindo criar-se uma hierarquia de classes. Nos níveis mais altos da hierarquia estão características comuns a todos os objetos desta classe, enquanto nos níveis inferiores estão especializações das classes superiores. As subclasses herdam as características comuns, além de definirem suas propriedades específicas.

Uma das grandes vantagens da programação OO é a utilização de bibliotecas de classes. Estas bibliotecas lembram as bibliotecas de código (procedimentos e funções), utilizadas na programação modular. As bibliotecas de classes permitem uma capacidade muito maior de compartilhamento e reutilização de código, pois é possível criar-se subclasses para atender novas necessidades, em função das classes já existentes. Muitas bibliotecas são oferecidas juntamente com as ferramentas de desenvolvimento para reduzir o tempo e a complexidade de projetos de software. O SOsim utiliza a biblioteca de classes do Delphi, a Visual Component Library – VCL [BORLAND, 1999].

O mecanismo de polimorfismo permite tratar objetos semelhantes de uma maneira uniforme. Neste caso, é possível que se envie uma mesma mensagem para um conjunto de objetos e que cada objeto responda de maneira diferente em função da mensagem recebida. Para ser implementado, o polimorfismo exige a utilização do conceito de herança e aplica-se apenas aos métodos da classe. O protocolo de comunicação é estabelecido na classe mais alta da hierarquia que será herdada por todas as subclasses definidas posteriormente. Este mecanismo cria um protocolo padrão de comunicação com um conjunto de objetos, permitindo uma grande flexibilidade na agregação de objetos semelhantes, mas não idênticos.

Concluindo, baseado nos conceitos de objetos, classes, encapsulamento, herança e polimorfismo, o paradigma da OO representa uma forma evolucionária de pensar e desenvolver software, trazendo inúmeros benefícios à criação de programas, dentre os quais o mais notável é a reutilização de código, que reduz drasticamente os tempos de desenvolvimento e manutenção de programas.

4.3 Multithread

Apesar do simulador não oferecer a facilidade de threads para seus usuários, o seu projeto está baseado inteiramente no conceito de ambiente multithread. Diversos objetos são implementados como threads para permitir a implementação do paralelismo necessário à simulação de um sistema operacional multiprogramável.

Basicamente, multithreading é uma técnica de programação concorrente, que permite projetar e implementar aplicações concorrentes e paralelas de forma eficiente, entretanto, o desenvolvimento de programas multithread não é simples. A presença do paralelismo introduz um novo conjunto de problemas, tais como a comunicação e sincronização de threads [MAIA, 1999] [PHAM, 1996].

O uso de (sub)processos no desenvolvimento de aplicações concorrentes demanda consumo de diversos recursos do sistema. Sempre que um novo processo é criado, o sistema deve alocar recursos (contexto de hardware, contexto de software e espaço de endereçamento) para cada processo, além de consumir tempo de CPU neste trabalho. No caso do término do processo, o sistema dispensa tempo para desalocar recursos previamente alocados. Na Fig. 4.1 existem três processos, cada um com seu próprio contexto de hardware, contexto de software e espaço de endereçamento.



Fig. 4.1 - Ambiente monothread

Como cada processo possui seu próprio espaço de endereçamento, a comunicação entre os (sub)processos torna-se difícil e lenta, pois utiliza mecanismos tradicionais como pipes, sinais, semáforos, memória compartilhada ou troca de mensagem. Além disto, o compartilhamento de recursos comuns aos (sub)processos concorrentes, como memória e arquivos abertos, não é simples.

Com threads, um processo pode ter diferentes partes do seu código sendo executadas concorrentemente ou simultaneamente, com muito menos overhead que utilizando múltiplos (sub)processos. Na Fig. 4.2 existe apenas um processo com três threads de execução, cada um com seu program counter (PC).

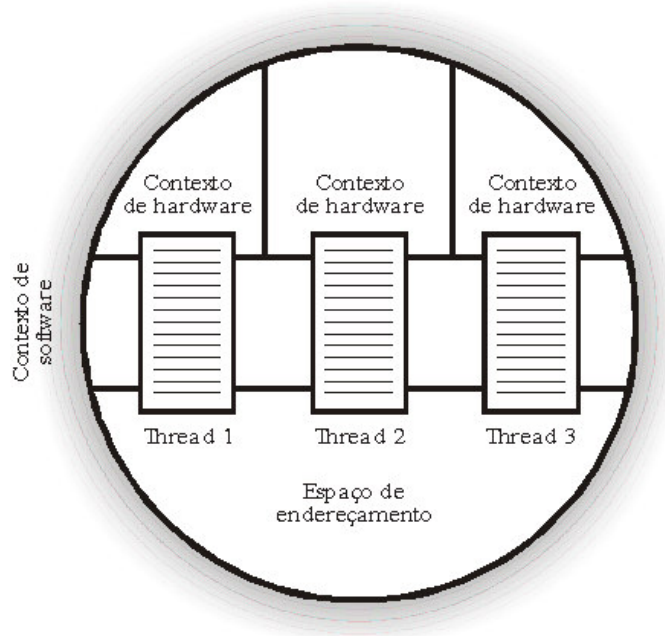


Fig. 4.2 - Ambiente Multithread

Como todos os threads em um processo compartilham o mesmo espaço de endereçamento, a comunicação entre os threads pode ser feita utilizando o compartilhamento de memória (shared memory) de forma rápida e eficiente. De forma semelhante ao compartilhamento de memória, os threads dentro do mesmo processo podem compartilhar facilmente outros recursos, como descritores de arquivos, temporizadores (timers), sinais, atributos de segurança etc.

4.4 Ambiente do Simulador

O ambiente do simulador foi desenvolvido pensando em oferecer aos seus usuários uma interface simples e de fácil interação. A partir do uso de uma ferramenta de desenvolvimento rápido de aplicações (Rapid Development Application – RAD) orientada a objetos, neste caso o Borland Delphi, esta tarefa foi bastante simplificada em função da biblioteca de objetos disponível.

Uma outra grande preocupação na implementação do simulador foi permitir que o usuário possa acompanhar visualmente os diversos eventos envolvendo cada processo no sistema. Se o simulador utilizasse a mesma dinâmica de um sistema operacional real, seria impossível aos olhos humanos acompanhar os eventos e as mudanças de estado do ambiente. Para isto, a utilização de cores e a possibilidade de alterar a velocidade do modelo são requisitos fundamentais.

O simulador quando inicialmente executado oferece três janelas: a console do simulador, gerência do processador e gerência de memória. A console permite abrir mais duas janelas opcionais: uma para a janela de log e outra com estatísticas do simulador.

O ambiente de janelas independentes permite ao usuário selecionar quais janelas são importantes para analisar uma determinada situação, deixando as demais fechadas. Outra vantagem é permitir que o usuário escolha o número de janelas abertas em função da resolução e do tamanho do monitor. Finalmente, um ambiente de janelas independentes permite que se amplie o modelo mais facilmente do que se fosse uma única tela englobando todas as funcionalidades.

- Console

A console permite criar e selecionar processos, abrir e fechar janelas, consultar o help online e os créditos, e encerrar a execução do simulador. Além disto, a console permite parar a simulação a qualquer momento e retomá-la quando desejado. Esta facilidade permite ao professor interromper o modelo em determinados momentos para esclarecer algum evento ou dúvida e, posteriormente, continuar o processamento (Fig. 4.3).

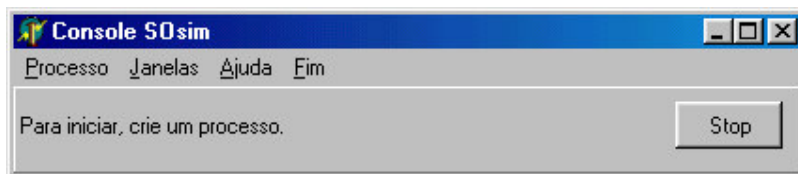


Fig. 4.3 - Console

- Gerência do processador

A janela “Gerência do Processador” permite visualizar a mudança de estado dos processos e definir opções relacionadas ao escalonador. Além disso, é possível alterar a dinâmica do simulador a partir das barras de controle Tempo de espera, Quantum e Frequência clock (Fig. 4.4). Este módulo é melhor detalhado no item 4.7 – Gerência do Processador.



Fig. 4.4 - Gerência do Processador

- Gerência de memória

A janela “Gerência de Memória” permite visualizar as páginas de memória livres, as páginas de memória alocadas (modificadas ou não) e definir opções da gerência de memória virtual (Fig. 4.5). Além disso, é possível acompanhar o tamanho da lista de páginas livre (FPL Size) e lista de páginas modificadas (MPL Size). Este módulo é melhor detalhado no item 4.8 – Gerência de Memória.

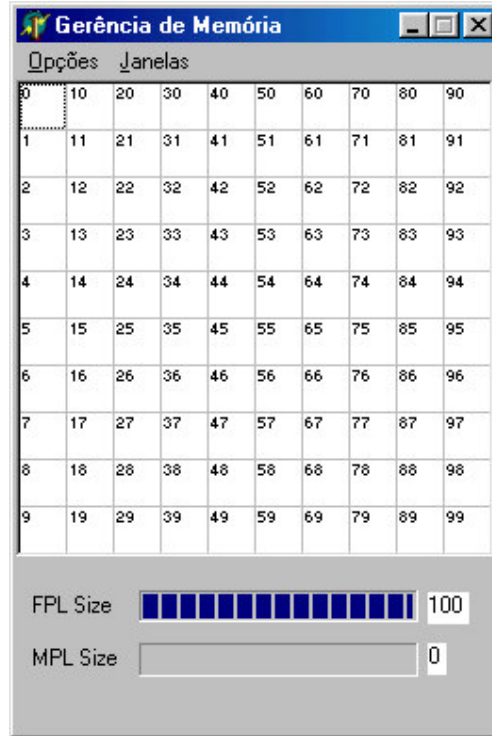


Fig. 4.5 - Gerência de Memória

- Log de mensagens

A janela “Log” permite acompanhar facilmente os eventos ocorridos no simulador e, principalmente, os relacionados a um processo, pois utiliza a cor associada ao processo para exibir a mensagem (Fig. 4.6).

Além disto, o simulador cria em disco um arquivo contendo todas as mensagens exibidas na janela de log. A cada execução do simulador, um novo arquivo é criado no formato mês-dia-hora.txt, que pode ser lido por qualquer editor de textos. É recomendável que o usuário elimine periodicamente os arquivos de log.

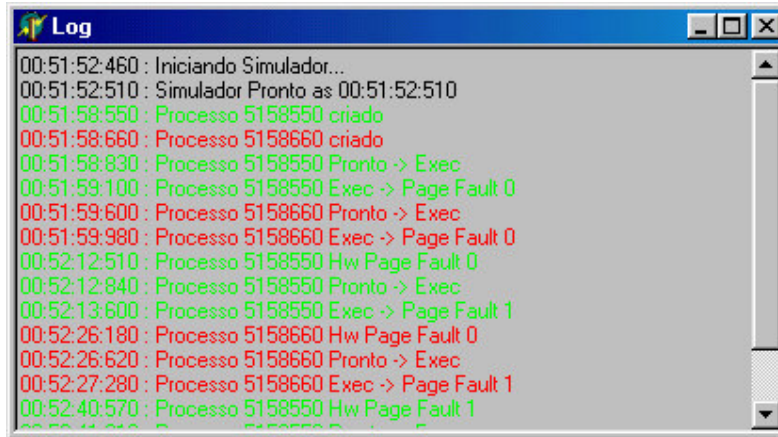


Fig. 4.6 – Log do simulador

- Estatísticas

A janela “Estatísticas” permite se ter uma visão geral do funcionamento do simulador e acompanhar os principais indicadores de desempenho dos módulos de gerência do processador e gerência de memória (Fig. 4.7). As informações contidas nesta janela serão melhor analisadas no item 4.9 - Estatísticas.

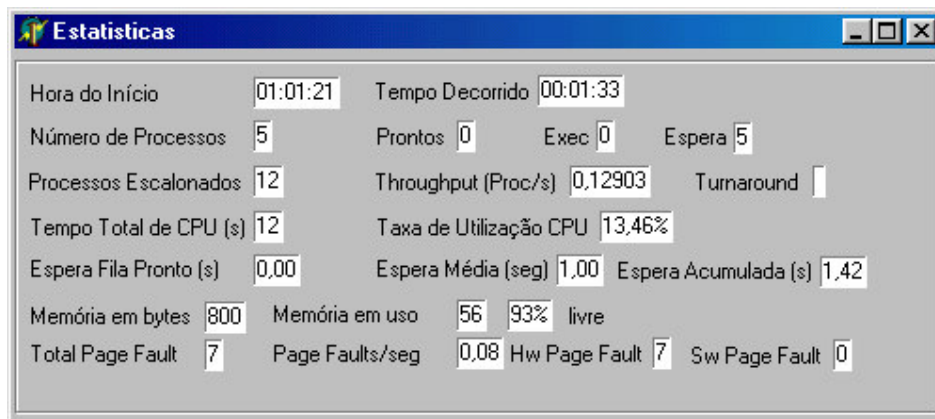


Fig. 4.7 – Estatísticas do simulador

4.5 Objetos do Simulador

O SOsim é formado por diversos objetos de diferentes tipos. A descrição de cada um destes objetos seria longa e tediosa, por isso optou-se por uma abordagem mais didática, sem entrar em detalhes do código fonte.

Alguns objetos funcionam como threads dentro do simulador, permitindo a execução concorrente deste objetos. Esta característica é fundamental para um simulador que trata de eventos assíncronos, como em um sistema operacional. A programação concorrente, porém, introduz uma série de problemas de comunicação e sincronização entre threads, principalmente envolvendo estruturas de dados compartilhadas [BEN-ARI, 1990].

A seguir, são descritos os principais objetos do simulador e suas funções. O nome entre parêntesis significa o nome interno do módulo.

- Clock (u_Clock)

O objeto Clock é um thread que tem a função de apenas gerar “pulsos“ para que o simulador tenha uma temporização. Estes pulsos são sinalizados ao objeto CPU, através de uma interrupção em intervalos de 100 milissegundos. O intervalo do clock pode ser alterado facilmente para acelerar o modelo através da barra de controle “Frequência clock” na janela “Gerência do Processador”.

- Núcleo (u_Kernel)

O objeto Núcleo mantém a lista de PCBs dos processos e o vetor de interrupção. O simulador implementa um vetor de 16 níveis de prioridade, variando de 0 (menor) a 15 (maior). Apesar de poder suportar este esquema, todas as interrupções são tratadas com o mesmo nível de prioridade (zero), ou seja, as interrupções são tratadas seqüencialmente por ordem de chegada. Em um sistema operacional real esta certamente não seria uma solução adequada, mas em um simulador com fins

educacionais esta solução funciona perfeitamente, devido ao pequeno número de interrupções simultâneas geradas no modelo.

- CPU (u_CPU)

O objeto CPU (Central Processor Unit) é o thread que executa as instruções dos programas. Além disto, a CPU reconhece e trata as interrupções armazenadas pelo objeto núcleo. Semelhante a um sistema real, o processador verifica a cada instrução se há interrupções e, neste caso, desvia para a rotina de tratamento apropriada. A CPU só executa instruções de um programa quando não existem interrupções pendentes.

- Processo (u_Processo)

O objeto Processo define os atributos e métodos de um processo no simulador. Como exemplos de atributos temos a identificação do processo (PID), data de sua criação, tempo de CPU e a prioridade de execução. Como exemplos de métodos temos a criação de um processo, sua suspensão e eliminação. Este objeto é detalhado no item 4.6 – Objeto Processo.

- Escalonador (u_Escalonador)

O objeto Escalonador é o thread que seleciona um processo para execução. Este objeto mantém uma estrutura de dados com todos os processos prontos para executar e, conforme o algoritmo de escalonamento definido pelo usuário, seleciona um processo para ganhar o processador.

O Escalonador oferece três opções de algoritmos para a gerência da CPU, sendo o mais importante o algoritmo de escalonamento com múltiplas filas, preemptivo e com prioridade dinâmica. Este objeto é detalhado no item 4.7 – Gerência do Processador.

- Memória Virtual (u_MemoVirtual)

O objeto Memória Virtual é responsável por alocar e desalocar a memória principal para os processos, utilizando o mecanismo de gerência de memória virtual com paginação e incluindo pre-paging, buffer de páginas e swapping. Este objeto mantém estruturas de dados responsáveis por controlar todas as áreas livres e alocadas da memória principal e secundária pelos processos em execução. Este objeto é detalhado no item 4.8 – Gerência de Memória.

- Espera (u_Espera)

O objeto Espera é o thread responsável por tratar todos os processos no estado de espera. Este objeto recebe processos vindos da CPU, trata o tipo de evento ocorrido e retorna o processo para o Escalonador conforme o tipo de evento.

Um processo pode sair do estado de execução para o estado de espera por três eventos: operação de leitura/gravação, page fault ou suspensão. Os dois primeiros eventos dependem do programa em execução, enquanto o terceiro depende de um agente externo (o usuário) que explicitamente suspende o processo.

Um processo pode sair do estado de espera para o estado de pronto por três eventos: término da operação de leitura/gravação, término da leitura da página que gerou o page fault ou resumo do programa. Os dois primeiros eventos são simulados pelo objeto utilizando um temporizador, que pode ser regulado a partir da barra de controle “Tempo de espera” na tela “Gerência do Processador”. O terceiro evento depende de um agente externo (o usuário) para retirar o processo do estado de suspensão (resume).

- Formulários (f_*)

O simulador utiliza vários formulários para entrada/saída de parâmetros e informações, como as janelas de log e estatísticas. A Tab. 4.1 apresenta uma breve descrição dos principais módulos implementados.

Form	Descrição
f_Console	Permite controlar o simulador.
f_Processo	Permite criar processos.
f_SelectProcesso	Permite selecionar um processo.
f_PCB	Permite visualizar o PCB dos processos.
f_Escalonador	Permite visualizar as mudanças de estado dos processos.
f_Opt_Escalonador	Permite alterar características da gerência do processador.
f_MemoriaVirtual	Permite visualizar a memória principal.
f_Opt_MV	Permite alterar características da gerência de memória.
f_Pagefile	Permite visualizar o arquivo de paginação.
f_Log	Permite visualizar as mensagens geradas pelo simulador.
f_Estatistica	Permite visualizar estatísticas do simulador.
f_Creditos	Permite visualizar os créditos do trabalho.

Tab. 4.1 – Formulários

4.6 Objeto Processo

O simulador materializa o bloco de controle do processo (Process Control Block — PCB) através do objeto Processo, que mantém todas as suas informações, como identificação, prioridade, estado corrente e informações sobre o programa em execução. Além destes atributos, o objeto Processo possui métodos que permitem sua criação, alteração, suspensão e eliminação.

O objeto Processo pode ser dividido logicamente nos três elementos básicos que formam o conceito de um processo tradicional: contexto de hardware, contexto de software e espaço de endereçamento, que juntos mantêm todas as informações necessárias ao controle do processo.

O contexto de hardware implementado pelo simulador é formado, basicamente, do conteúdo do registrador program counter (PC) e bits de estado. Quando um processo

está em execução, o seu contexto de hardware está armazenado nos registradores do processador. No momento em que o processo perde a utilização da CPU, o sistema salva suas informações no seu contexto de hardware.

O contexto de software especifica características do processo que vão influir na execução de um programa, sendo que estas características são determinadas no momento da criação do processo. O contexto de software no SOsim define basicamente dois grupos de informações sobre um processo: sua identificação e suas quotas.

Cada processo criado pelo simulador recebe uma identificação única (PID — process identification), representada por um número e uma cor. O número é criado a partir da hora (hora, minutos e segundos) corrente. A cor pode ser selecionada pelo usuário ou o simulador oferece uma das várias opções de cores disponíveis. O simulador evita a repetição de cores utilizando um vetor de cores pré-definido.

As quotas são os limites de cada recurso do sistema que um processo pode alocar. Caso uma quota seja insuficiente, o processo poderá ser executado lentamente ou mesmo não ser executado. Um exemplo de quota implementada no simulador é o número máximo de frames que o processo pode alocar.

O espaço de endereçamento é a área de memória do processo onde o programa será virtualmente executado. O simulador implementa uma tabela de páginas para cada processo. Sendo assim, cada processo possui seu próprio espaço de endereçamento, que é protegido do acesso dos demais processos.

4.6.1 Criação do Processo

A criação de um processo é bastante simples e pode ser feita a partir de valores pré-definidos oferecidos pelo simulador ou definidos pelo próprio usuário (Fig. 4.8). Por uma questão de praticidade, o simulador permite que na hora da criação de um processo se especifique o número de processos a serem criados. Por exemplo, se o usuário

especificar três processos a serem criados, todas as características contidas no formulário, com exceção da cor, são repetidas nos processos.

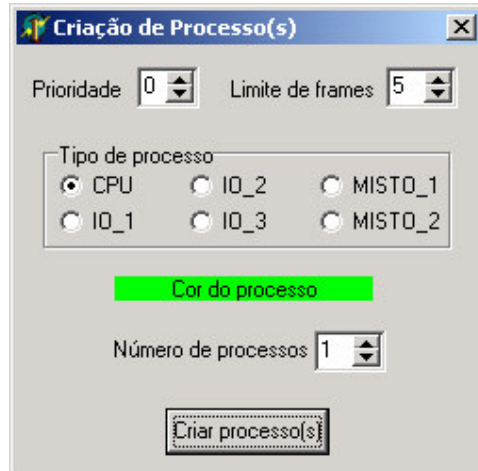


Fig. 4.8 - Criação de processo(s)

A cor de cada processo deve ser atribuída de forma a evitar que dois processos tenham a mesma cor, o que prejudicaria o acompanhamento visual dos processos. O simulador permite selecionar uma cor específica ou que o próprio software selecione uma cor em função de um vetor de cores pré-definido.

Na Fig. 4.8, a “Prioridade” define a prioridade base do processo e varia de 0 a 15 (default 0). O “Limite de frames” define o número máximo de páginas que um processo pode alocar na memória principal e varia de 1 a 5 (default 5).

O programa implementa um esquema de perfis para a escolha do programa a ser executado pelo simulador. Ao contrário do usuário ter que escrever um programa em assembly ou em uma linguagem de alto nível, basta ele selecionar um perfil de código pré-definido. Existem seis “Tipo de processo” pré-definidos que podem ser selecionados no momento da criação do processo (Tab. 4.2). Foi padronizado que todos os tipos de código geram apenas uma operação de gravação na memória, sempre na primeira página do programa.

Programa	Descrição
CPU	Não realiza operações de entrada/saída.
IO_1	Realiza apenas operações de entrada/saída do tipo 1.
IO_2	Realiza apenas operações de entrada/saída do tipo 2.
IO_3	Realiza apenas operações de entrada/saída do tipo 3.
MISTO_1	Realiza instruções de acesso à memória e operações de entrada/saída do tipo 1.
MISTO_2	Realiza instruções de acesso à memória e operações de entrada/saída do tipo 2.

Tab. 4.2 - Códigos pré-definidos

Foram implementados cinco tipos de instruções que podem ser executadas pelo processador (Tab. 4.3). A primeira instrução funciona apenas como um load que não altera o conteúdo da memória. A segunda funciona como um store, alterando o conteúdo da memória principal. Esta instrução é importante para gerar páginas modificadas no modelo de memória virtual. Do terceiro ao quinto tipos, as instruções funcionam como uma operação de entrada/saída, obrigando o processo a ser colocado no estado de espera. A diferença dos tipos de instruções de E/S é apenas o boost na prioridade base do processo.

Instrução	Descrição
0	Leitura da memória principal.
-1	Gravação na memória principal.
1	Operação de entrada/saída que permite aumento de prioridade de um ponto.
2	Operação de entrada/saída que permite aumento de prioridade de dois pontos.
3	Operação de entrada/saída que permite aumento de prioridade de três pontos.

Tab. 4.3 - Tipos de instruções

4.6.2 Visualização dos Processos

O simulador permite visualizar todos os processos criados em um formato semelhante ao apresentado por sistemas operacionais reais. Cada linha exibe a cor e a identificação do processo, a prioridade base e dinâmica respectivamente, o tempo de UCP e o número de frames alocados na memória (Fig. 4.9). As informações contidas na janela são atualizadas em tempo-real, permitindo acompanhar as alterações de alguns campos.



Fig. 4.9 - Visualização dos processos

Operação	Descrição
Suspende	Faz com que o processo selecionado fique no estado de espera.
Resume	Faz com que um processo suspenso retorne para a fila de processos no estado de pronto.
Finalizar	Faz com que o processo seja terminado e seu PCB eliminado.
PCB	Permite visualizar o bloco de controle do processo.

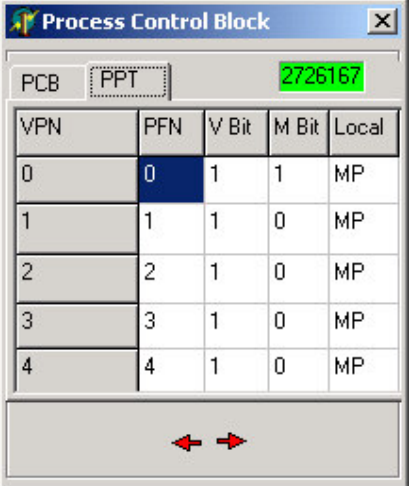
Tab. 4.4 - Operações sobre processos

É possível realizar algumas operações sobre um processo selecionado na janela de visualização de processos (Tab. 4.4). Além destas operações, o usuário poderá alterar a

prioridade base de um processo, realizando um duplo clique no campo “Prio” do processo selecionado.

4.6.3 Visualização dos PCBs

O bloco de controle (Process Control Block - PCB) é a alma de um processo. Nele temos todas as informações importantes para executar um programa em um ambiente multiprogramável. O simulador permite visualizar o PCB de qualquer processo selecionado e observar seu contexto de hardware e software (Fig. 4.10).



VPN	PFN	V Bit	M Bit	Local
0	0	1	1	MP
1	1	1	0	MP
2	2	1	0	MP
3	3	1	0	MP
4	4	1	0	MP

Fig. 4.10 - Process Control Block

O PCB exibe todas as propriedades do processo e são atualizadas periodicamente, permitindo observar as mudanças de suas características em tempo real. A opção PPT permite visualizar a tabela de páginas do processo (Process Page Table – PPT), que representa o seu espaço de endereçamento. A partir das setas da Fig. 4.10, é possível percorrer a lista de processos criados e observar seus atributos.

4.7 Gerência do Processador

O ambiente de gerência do processador é responsável por diversas funções no simulador, sendo implementada por diversos módulos. Na Fig. 4.11, existem três processos no simulador: o processo de cor vermelha está no estado de pronto para execução e aguarda pela liberação da CPU (objeto Escalonador), o processo em verde está sendo executado (objeto CPU) e o de cor amarela aguarda por algum evento no estado de espera (objeto Espera). A gerência do processador permite acompanhar facilmente as mudanças de estado de um processo, através de sua cor e posição no modelo de três estados.

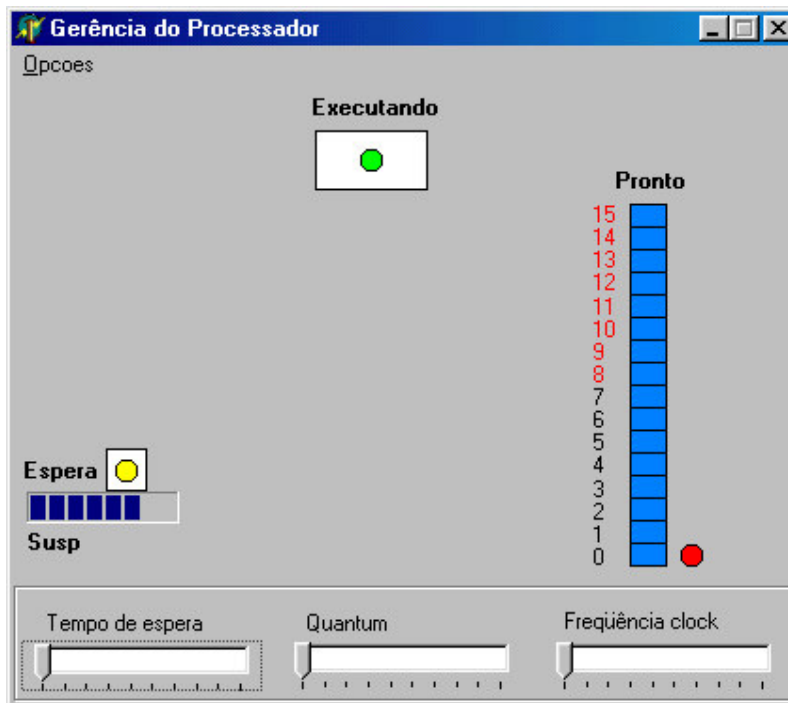


Fig. 4.11 - Estados dos processos no simulador

Na parte inferior da janela “Gerência do Processador” é possível alterar a dinâmica de funcionamento do simulador através de três barras de controle. A barra “Tempo de espera” permite controlar o tempo que um processo fica no estado de espera. A barra “Quantum” permite controlar o tempo máximo que um processo pode permanecer na

CPU sem sofrer uma interrupção de time-slice. Finalmente a barra “Frequência clock” permite alterar o intervalo de tempo que a CPU sofre uma interrupção de clock.

O objeto Escalonador é responsável por manter a lista de processos prontos para execução e selecionar dentre os processos disponíveis um que ganhará o acesso ao processador. Os processos no estado de pronto são selecionados em função do algoritmo de escalonamento escolhido (Fig. 4.12).

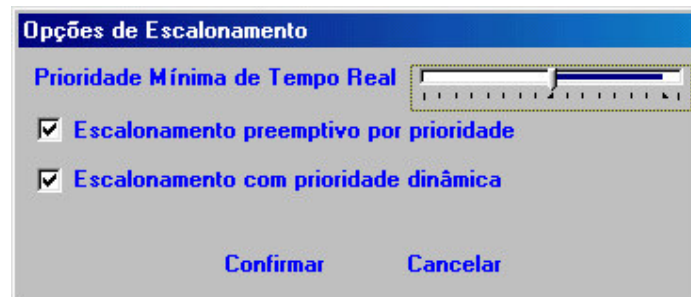


Fig. 4.12 - Opções de escalonamento

O escalonador implementa dezesseis níveis de prioridade, de zero (0) a quinze (15), sendo que a seleção é feita de forma decrescente, ou seja, os processos de maior prioridade são escalonados primeiro. Para cada nível de prioridade existe uma lista de processos de mesma prioridade, funcionando no modelo de fila FIFO. Existem dois motivos para a escolha deste intervalo: o primeiro é oferecer um número suficiente de níveis de prioridade em função do espaço disponível para representá-los na tela e o segundo por ser um intervalo real, semelhante aos utilizados nos sistemas OpenVMS e Windows 2000.

A “Prioridade mínima de tempo real” define a partir de qual nível de prioridade o escalonamento de preempção por tempo está desabilitado. Na Fig. 4.12, a partir da prioridade de nível oito não existe mais o conceito de quantum, ou seja, um processo nesta condição utiliza o processador enquanto não surgir um processo de maior prioridade (preempção por prioridade) ou o processo não deixar espontaneamente a CPU. Este modelo de escalonamento híbrido permite que o simulador ofereça um

ambiente para aplicações de tempo compartilhado e de tempo real. Na mesma figura, processos com prioridade entre zero e sete serão tratados como processos de tempo compartilhado, enquanto processos entre oito e quinze serão considerados como de tempo real.

A opção de “Escalonamento preemptivo por prioridade” habilita o simulador a interromper um processo em execução e transferi-lo para o estado de pronto no momento que um processo de maior prioridade chega na lista de processos em estado de pronto.

A opção de “Escalonamento com prioridade dinâmica” habilita o simulador a dar aumentos de prioridade (boost) aos processos em função do tipo de evento que o fez sair do estado de execução para o estado de espera (Tab. 4.5). O tipo de operação de entrada/saída é definido no momento da criação do processo, a partir da escolha de uma das opções de código pré-definido (Fig. 4.8).

Boost	Tipo de evento
+1	Operação de E/S do tipo 1.
+2	Operação de E/S do tipo 2 e page fault.
+3	Operação de E/S do tipo 3.

Tab. 4.5 - Aumento de prioridade

O aumento de prioridade é sempre dado a partir da prioridade base, ou seja, se um processo tem prioridade base um e prioridade dinâmica três, caso o processo tenha um boost de um, sua prioridade dinâmica não será quatro, mas sim dois. Processos com prioridade base na faixa de prioridade de tempo real não sofrem aumento de prioridade.

4.8 Gerência de Memória

A gerência de memória virtual é responsável, basicamente, por alocar e desalocar memória principal e secundária para os processos. Isto significa que o módulo de

gerência de memória (objeto MemoVirtual) é responsável pelo mapeamento, definição de tamanho de página, políticas de busca, alocação e substituição de páginas e swapping.

A janela “Gerência de Memória” representa a memória principal, formada por 100 frames numerados de 0 a 99 (Fig. 4.13). Conforme os processos são criados, os frames são alocados para o processo. Cada frame alocado é marcado com um círculo da mesma cor que o processo, permitindo uma identificação rápida de quais as páginas cada processo está alocando. No exemplo, o processo em verde está alocando os frames 0, 2, 4, 6 e 8 da memória principal, enquanto o processo em vermelho está alocando os frames 1, 3, 5, 7 e 9. As páginas 0 e 1, de fundo escuro, representam páginas modificadas dos processos.

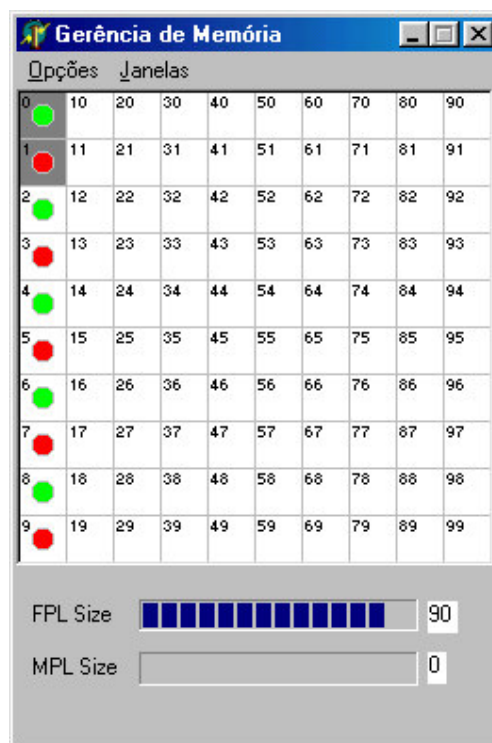


Fig. 4.13 – Memória principal

Na parte inferior da janela “Gerência de Memória” existem dois contadores. O “FPL Size” indica o tamanho da lista de páginas livres (Free Page List – FPL) na memória, ou

seja, quantas páginas estão disponíveis para uso. O “MPL Size” indica o tamanho da lista de páginas modificadas (Modified Page List – MPL), ou seja, quantas páginas modificadas não foram gravadas no arquivo de paginação.

O módulo de gerência de memória oferece alguns parâmetros de configuração da gerência de memória virtual (Fig. 4.14). A opção “Política de busca de páginas” permite selecionar se a política de busca será antecipada (default) ou por demanda. O esquema de busca antecipada faz com que a gerência de memória carregue antecipadamente o programa a ser executado da memória secundária para a memória principal. O esquema de pre-paging também foi incluído no simulador por motivos didáticos. Geralmente, quando ensinamos as mudanças de estados do processo, muito pouco foi falado sobre memória virtual e menos ainda sobre page fault. Na gerência de memória virtual sem paginação antecipada, as páginas de um programa são carregadas para a memória principal por demanda. Isto gera um page fault inicial para cada página referenciada, acarretando uma mudança do estado (execução → espera). Esta situação introduz mais uma mudança de estado (page fault) no modelo, o que poderia complicar a abordagem inicial das mudanças de estados do processo.

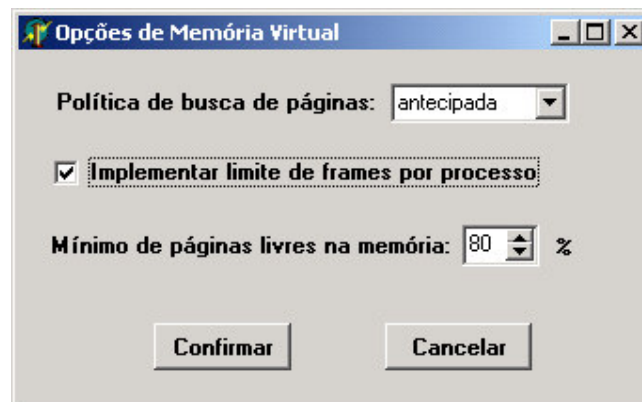


Fig. 4.14 - Opções de memória

A opção “Implementar limite de frames ...” habilitada faz com que o módulo de gerência de memória utilize o conceito de conjunto de páginas mais referenciadas. Este conceito é melhor detalhado no item 4.8.3 – Lista de Frames. A opção “Mínimo de

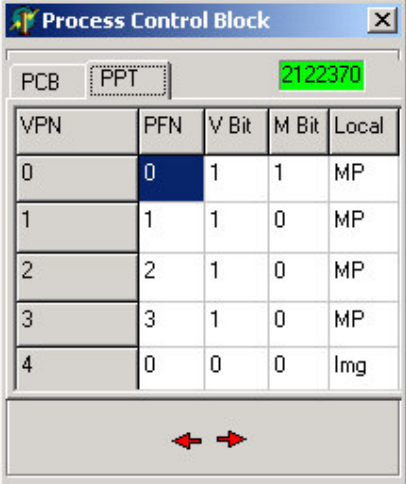
páginas livres ...” representa o número mínimo de páginas disponíveis na memória principal aceito pelo simulador, sendo o default 80% por cento. Quando este valor é atingido, o módulo de gerência de memória ativa o esquema de recuperação de páginas para garantir o tamanho mínimo da lista de páginas livres especificado.

4.8.1 Mapeamento

Um programa no ambiente do simulador não faz referência a endereços físicos de memória (endereços reais), mas apenas a endereços virtuais. No momento da execução de uma instrução, o endereço virtual é traduzido para um endereço físico, pois o processador acessa apenas posições da memória principal. O mecanismo de tradução do endereço virtual para endereço físico é denominado mapeamento.

Cada processo possui seu espaço de endereçamento próprio, implementado através de uma tabela de páginas (Process Page Table – PPT). Cada entrada na tabela de páginas (Page Table Entry – PTE) do processo permite mapear uma página virtual em um frame na memória principal (Fig. 4.15). Cada PTE, além do endereço do frame na memória principal, possui dois bits de controle: um para indicar se a página está na memória principal (V bit ou bit de validade) e outra para indicar se a página foi modificada (M bit ou bit de modificação).

Na Fig. 4.15, o processo selecionado possui quatro páginas virtuais (Virtual Page Number - VPN) na memória principal: VPN0 a VPN3. Isto pode ser facilmente acompanhado a cada page fault, que faz com que o simulador aloque um frame na memória principal (Page Frame Number - PFN) e ligue o bit de validade (V Bit=1). Além disso, a VPN0 possui seu bit de modificação ligado (M bit=1), indicando que a página foi modificada. Finalmente, a figura mostra que a VPN4 não foi ainda referenciada (PFN=-1).



VPN	PFN	V Bit	M Bit	Local
0	0	1	1	MP
1	1	1	0	MP
2	2	1	0	MP
3	3	1	0	MP
4	0	0	0	Img

Fig. 4.15 - Tabela de páginas

4.8.2 Tamanho de página

A definição do tamanho de página em sistemas que implementam memória virtual por paginação é um importante fator no seu projeto. O tamanho da página está associado ao hardware e varia de acordo com a arquitetura. No caso do SOsim, a definição do tamanho de página não está relacionada à arquitetura de hardware ou ao possível problema do tamanho da tabela de páginas dos processos. A questão mais importante para a definição do tamanho de página a ser utilizado no simulador foi como conseguir mostrar o funcionamento do mecanismo de memória virtual de uma forma que o aluno consiga visualizar e entender o mecanismo.

A princípio, o simulador pode suportar qualquer tamanho de página, bastando alterar a definição da constante `PAGE_SIZE`, definido com oito bytes. Como o tamanho máximo de um programa é de 40 bytes, definido na constante `MAX_PGM_SIZE`, o tamanho da tabela de páginas de cada processo é de cinco entradas (PTEs), variando de zero (VPN0) a quatro (VPN4). Apesar de ser um valor pequeno, este número permite ao aluno acompanhar facilmente a evolução da alocação e substituição de páginas de um ou mais processos na memória principal.

4.8.3 Lista de Frames

A lista de frames é o conjunto de páginas que um processo possui na memória principal em determinado instante de tempo. A lista de frames permite criar algo semelhante ao conceito de working set, implementado pelos sistemas OpenVMS e Windows 2000.

Sempre que uma página é referenciada, o bit de validade da entrada da tabela de páginas (PTE) correspondente é verificado. Caso ele esteja ligado indicará que esta é uma página válida, pois está na lista de frames. Desta forma o sistema poderá localizar a página na memória real, através das demais informações contidas no PTE. Por outro lado, se o bit de validade não estiver ligado, significará que esta é uma página não válida. Neste caso, podemos dizer que ocorreu um page fault, que é a interrupção (exceção) gerada quando uma página referenciada não está na lista de frames. Quando isto acontece, o sistema se encarregará de trazer a página para a lista, atualizar o PTE e executar novamente a referência à página, que agora será válida.

O SOsim utiliza a política de alocação de páginas fixa, ou seja, o tamanho da lista de frames não é alterado durante a simulação. O tamanho da lista de frames de cada processo é definido individualmente na sua criação e o default é o processo ser criado com cinco páginas. Este valor default permite que o processo tenha todas as suas páginas na memória principal. O esquema de limite de frames pode ser desabilitado no simulador através das opções da gerência de memória virtual.

4.8.4 Política de Substituição de Páginas

Cada processo possui um limite máximo para o número de frames que podem ser alocados na memória principal. Quando este limite é alcançado e o processo necessita trazer uma nova página para a lista de frames, um page fault ocorre e uma página deve ser cedida em troca por uma nova página (Fig. 4.16a).

O simulador implementa a política de descartar a página mais antiga presente na lista de frames, utilizando uma estrutura de dados do tipo fila (FIFO). Aparentemente esta

política pode não parecer apropriada, se partirmos do princípio que a página descartada pode ser uma página muito referenciada. Para contornar este problema, o simulador implementa o mecanismo de buffer de páginas que compensa a opção por essa política.

No momento em que um determinado processo descarta uma página, esta página pode tomar rumos diferentes em função das suas características. Para entender como isso ocorre, é preciso antes distinguir as páginas modificáveis das não modificáveis.

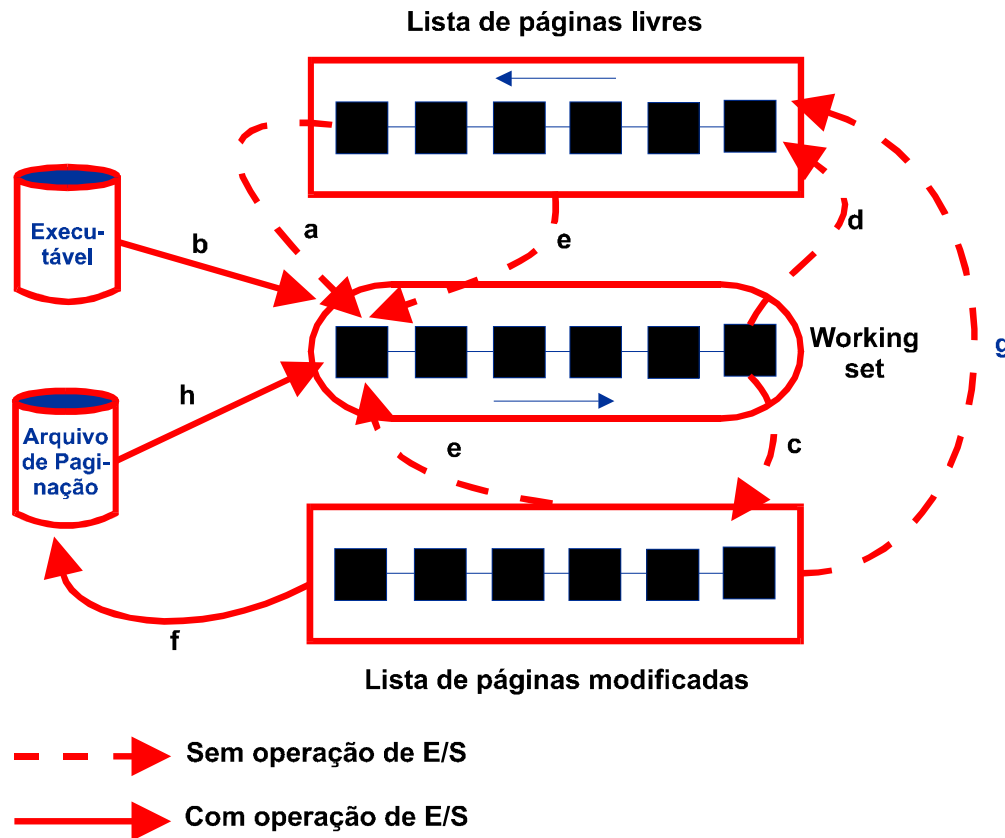


Fig 4.16 - Política de substituição de páginas

As páginas modificáveis são aquelas que armazenam variáveis, que naturalmente trocam de valor durante a execução do programa. Por outro lado, as páginas que contém código executável do programa não podem ser alteradas e portanto são chamadas de não modificáveis. Uma página modificada não pode ser descartada quando sai da lista de frames do processo, pois ela contém dados necessários à continuidade da execução do

programa. É preciso, então, que essa página seja gravada em disco para que, na próxima referência, a mesma volte com os valores originalmente gravados. As páginas não modificadas, por sua vez, não precisam desse tratamento, pois podem ser retiradas do arquivo executável quando necessário (Fig. 4.16b).

Os tempos de leitura e gravação em disco são reconhecidamente elevados quando comparados com o tempo de acesso à memória. Logo, causaria um grande retardamento na execução do programa caso os procedimentos, antes descritos, fossem seguidos fielmente. Imagine se a cada entrada ou saída de página da lista de frames fossem feitas leitura e gravação em disco respectivamente! Para contornar esse problema o sistema mantém duas listas que funcionam em conjunto no sentido de diminuir as operações em disco. Essas listas são a lista de páginas livres (Free Page List – FPL) e a lista de páginas modificadas (Modified Page List – MPL), as quais funcionam como será descrito.

Cada PTE, além do bit de validade, possui um bit de modificação (M bit) que indica se a página sofreu alguma alteração no seu conteúdo. Assim, quando uma página deve ser substituída, o bit de modificação é primeiramente verificado. Caso esse esteja ligado (indicando que houve modificação), a página é passada do working set do processo para a lista de páginas modificadas (Fig. 4.16c). Por outro lado, se o bit estiver desligado, a página é passada para a lista de páginas livres (Fig. 4.16d).

A lista de páginas livres é uma estrutura de dados da gerência de memória virtual responsável por manter informações sobre todas as páginas físicas da memória principal que estão disponíveis para serem utilizadas. Quando um processo libera uma página para a lista de páginas livres, esta página não é imediatamente utilizada, já que a página deve percorrer a lista do fim até o seu início (FIFO). Desta forma, as páginas que saem da lista de frames permanecem na memória por algum tempo, permitindo que essas mesmas páginas possam voltar para o processo sem operações de leitura em disco (Fig. 4.16e).

Na ocorrência de um page fault, a gerência de memória pode eventualmente encontrar a página requerida na lista de páginas livres ou na lista de páginas modificadas, gerando um page fault sem operação de E/S, conhecido como soft page fault. Quando não ocorre o esse tipo de page fault, o sistema é obrigado a trazer a página requerida do disco (hard page fault) e solicitar um novo frame à lista de páginas livres, no qual a página será colocada.

A lista de páginas livres, eventualmente, fica com poucas páginas disponíveis para atender a demanda dos processos. Este parâmetro é definido nas opções de gerência de memória virtual, sendo que o default é cinquenta por cento. Nesse caso, as páginas que estão na lista de páginas modificadas são gravadas em disco no arquivo de paginação (Fig. 4.16f) e suas páginas são passadas para a lista de páginas livres (Fig. 4.16g). Note que apenas nesse caso as páginas modificadas são gravadas em conjunto no disco e, portanto, apenas uma operação de gravação em disco é efetuada.

Quando ocorre um page fault, a página requerida pode estar em vários lugares como no arquivo executável, na lista de páginas modificadas, na lista de páginas livres ou no arquivo de paginação (Fig.4.16h). O arquivo de paginação é uma área em disco comum a todos os processos, onde as páginas modificáveis são salvas para posterior reutilização, além de servir como área de swapping.

4.8.5 Arquivo de Paginação

O arquivo de paginação (pagefile) é utilizado pelo módulo de gerência de memória com dois propósitos básicos: servir como repositório para a lista de páginas modificadas e como área de swapping. Alguns sistemas operacionais, como o Windows 2000, implementam este mecanismo de forma semelhante ao simulador, mas existem sistemas que implementam dois arquivos separados para paging e swapping (OpenVMS) e outros simplesmente não realizam swapping, apenas paginação. A decisão por implementar este modelo foi, novamente, com fins educacionais, pois permite simular tanto o mecanismo de paginação quanto o de swapping, concentrando os eventos em uma mesma janela (arquivo) e simplificando a análise (Fig. 4.17).

Quando um processo é criado, a gerência de memória reserva um número pré-definido (MAX_PAGEFILEQUOTA) de blocos no arquivo de paginação. Como pode ser observado na Fig. 4.17, os blocos numerados de 11 à 19 estão marcados como reservados (“R”). No mesmo arquivo existem páginas modificadas (fundo escuro) e páginas não modificadas (fundo claro) de diversos processos.

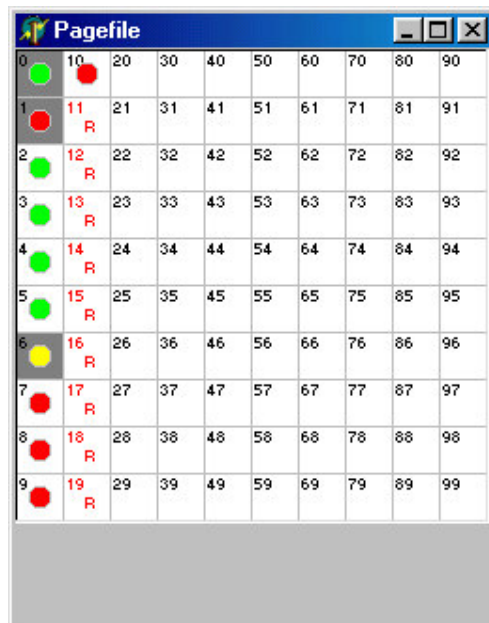


Fig. 4.17 – Arquivo de paginação e swapping

As páginas modificadas no arquivo de paginação são gravadas quando o número mínimo de páginas livres está abaixo do limite mínimo definido nas opções de gerência de memória virtual (Fig. 4.14). Neste caso, após a gravação da lista de páginas modificadas no arquivo de paginação, suas páginas são em seguida liberadas para a lista de páginas livres.

Caso a gravação da lista de páginas modificadas não seja suficiente para recuperar o número mínimo de páginas livres, o processo de swapping é iniciado. O algoritmo de seleção do candidato a ser retirado da memória principal é implementado em etapas e segue os passos descritos a seguir:

1. Os processos no estado de suspensão (“Susp”) são os primeiros a serem selecionados para deixarem a memória principal, passando para o estado de “SuspOut”;
2. Se o limite de páginas livres não for alcançado, são selecionados os processos no estado espera por uma operação de E/S (“IO”). Processos nesta situação são gravados no arquivo de swap e passam para o estado “IOOut”;
3. Em seguida são selecionados os processos no estado de page fault (“PFault”), que passam para o estado de “PFOut”;
4. Finalmente os processos no estado de pronto (“Pronto”) com menor prioridade são selecionados para deixarem a memória e passam para o estado de “ProntoOut”.

Em todos os casos, as páginas dos processos na memória principal são liberadas para a lista de páginas livres. As páginas dos processos que sofreram o processo de swapping retornam à memória quando são escalonados para execução.

4.9 Estatísticas

A janela “Estatísticas” oferece ao usuário uma enorme variedade de indicadores dos diversos módulos que compõem o simulador (Fig. 4.7). Estes valores podem ser utilizados para comparar políticas e mostrar numericamente as vantagens entre elas. Além disso, as estatísticas permitem ao aluno ter contato com um lado da área de sistemas operacionais muito pouco discutido nas disciplinas sobre o tema, a parte de análise de desempenho. Os indicadores apresentados são em sua maioria utilizados na prática para que gerentes de sistemas façam estudos de performance em seus ambientes.

Na Tab. 4.6 são apresentados os principais indicadores estatísticos e uma descrição sucinta de cada um.

Indicador	Descrição
Hora de Início	Hora, minuto e segundos do início de operação do simulador.
Tempo Decorrido	Diferença entre a hora corrente e a hora de início.

Número de Processos	Número de processos ativos no simulador, nos estados de execução, pronto e espera. Como o simulador implementa apenas uma CPU, o número de processos em execução é sempre um.
Processos Escalonados	Número de processos que saíram do estado de pronto para o estado de execução, ou seja, foram escalonados para executar.
Throughput	Taxa de processos executados por segundo.
Turnaround	Tempo médio para a execução de um processo, desde sua criação no simulador até o seu término.
Tempo Total de CPU	Tempo total (em segundos) que a CPU esteve ocupada. Valores próximos de 90% indicam que o processador está no limite de sua utilização.
Taxa de utilização da CPU	Relação entre o tempo total de CPU e o tempo que o simulador está sendo utilizado.
Espera Fila de Pronto	Somatório dos tempos de espera (em segundos) dos processos no estado de pronto para execução.
Espera Média	Tempo médio de espera (em segundos) dos processos no estado de pronto para execução.
Espera Acumulada	Tempo médio acumulado de espera (em segundos) dos processos no estado de pronto para execução.
Memória em bytes	Memória principal total.
Memória em uso	Memória principal sendo utilizada pelos processos.
Livre	Porcentual de memória principal disponível.
Total Page Fault	Número acumulado de page faults.
Page Faults/seg	Taxa de page faults por segundo.
Hw Page Fault	Total acumulado de hardware page faults.
Sw Page Fault	Total acumulado de software page faults.

Tab. 4.6 – Indicadores estatísticos

CAPÍTULO 5 – CONCLUSÕES E TRABALHOS FUTUROS

5.1 Introdução

Esta dissertação de tese é apenas parte de um longo trabalho acadêmico, que envolveu a publicação de um livro com duas edições e vários anos lecionando disciplinas direta ou indiretamente ligadas ao estudo de sistemas operacionais.

O SOsim é uma ferramenta de apoio educacional que permite facilitar enormemente o processo de ensino e aprendizado dos conceitos e técnicas envolvidos em um sistema operacional multiprogramável.

O simulador será colocado gratuitamente para download na Internet, incluindo seu código fonte. Com o lançamento da terceira edição do livro “Arquitetura de Sistemas Operacionais”, espera-se que a comunidade acadêmica ganhe conhecimento da existência do software educacional e passe a utilizá-lo como suporte ao ensino na disciplina de sistemas operacionais.

5.2 Conclusões

Com o simulador são esperados um grande avanço quantitativo e qualitativo no processo de ensino. Esta melhora é percebida principalmente em cursos de curta duração, como em cursos técnicos profissionalizantes e de extensão, ou cursos que tenham alunos que não possuam, ou que possuam parcialmente, os pré-requisitos.

Os testes utilizando o simulador foram realizados nos cursos de graduação em Informática da PUC-Rio e pós-graduação em Programação e Análise de Sistemas da PUC-Rio/CCE. A partir dos testes implementados em sala de aula, constatou-se uma série de contribuições para o ensino e aprendizado de sistemas operacionais, tais como:

- Ampliação da eficiência no processo de ensino/aprendizado por parte de professores e alunos, permitindo que conceitos complexos fossem assimilados mais facilmente pelos alunos;
- Melhora na comunicação professor-aluno, reduzindo o tempo necessário para as apresentações conceituais e, conseqüentemente, permitindo ampliar o programa da disciplina e/ou criar projetos práticos. Além disso, o software motiva o debate e a discussão sobre possíveis cenários de simulação;
- Os testes realizados atingiram apenas a utilização do software para apoio de aulas presenciais. Em experimentos futuros, o simulador será testado como ferramenta de auxílio ao ensino à distância, mas é de se supor que ganhos semelhantes aos alcançados em aulas presenciais sejam obtidos;
- Os testes mostraram que o software facilita a transferência e aquisição de conhecimento em função do maior grau de participação dos alunos, que trocam suas experiências;
- Os testes indicam que os resultados variam conforme o tipo do curso (extensão, graduação e pós-graduação) e dos pré-requisitos dos alunos,. Em cursos de menor duração e/ou com alunos sem os pré-requisitos necessários verifica-se um ganho maior com a utilização do simulador.

5.3 Trabalhos Futuros

O simulador aqui apresentado é comprovadamente um avanço para a melhoria do ensino e aprendizado de sistemas operacionais, mas é possível ir muito além. Em função do escopo inicialmente definido, foram implementados o núcleo (kernel) do simulador, os módulos de gerência do processador e gerência de memória.

Como motivação para trabalhos futuros, é possível considerar o desenvolvimento de novos módulos e funcionalidades descritas a seguir:

- Desenvolvimento do módulo de gerência de sistema de arquivos, que permita a criação de arquivos e diretórios, além da manipulação dos mesmos através de uma interface gráfica;
- A gerência de dispositivos é uma das áreas mais complexas de um sistema operacional. Mostrar o funcionamento de discos e fitas, além de interfaces de rede, seria desejável;
- A gerência de múltiplos processadores permitiria visualizar um sistema multiprocessado, sendo que o número de processadores poderia ser um parâmetro definido pelo próprio professor ou aluno;
- Apesar de existirem bons simuladores na área de comunicação e sincronização entre processos, nenhum possui uma interface gráfica que facilite o processo de ensino, além de não apresentarem as interfaces com os demais módulos de um sistema operacional;
- O estudo de sistemas operacionais, muitas vezes, deve considerar questões da arquitetura de computadores. A criação de uma máquina virtual de hardware que oferecesse a possibilidade de observar as dependências entre os dois mundos seria uma ótima ferramenta para ambas as disciplinas;
- A partir de uma máquina virtual seria possível a criação de programas simples em assembly, que poderiam ser interpretados pelo processador. Com esta facilidade, o simulador permitiria comparar o código de dois programadores e observar, por exemplo, aquele de melhor desempenho;
- A técnica de paginação antecipada foi apenas empregada na criação de um processo. É possível utilizar a mesma técnica na ocorrência de um page fault. Neste caso, sempre que houver um page fault, o sistema poderia trazer, além da página referenciada, um conjunto de páginas para evitar operações de leitura sucessivas;

- O simulador implementa apenas um modelo de gerência de memória: memória virtual com paginação. Apesar deste modelo ser o mais empregado atualmente, é possível implementar mecanismos alternativos, como por exemplo segmentação com paginação, de forma a ampliar as funcionalidades do simulador;
- O ambiente gráfico, apesar de ser uma ferramenta poderosa de ensino, distancia o aluno do mundo real. Existem ambientes que exigem uma interação entre o sistema e o usuário através de linhas de comandos. O desenvolvimento de um shell de comandos como opção, por exemplo, para a eliminação ou a suspensão de um processo, seria bastante útil em cursos mais práticos e menos conceituais;
- É possível tornar o SOsim uma ferramenta educacional mais colaborativa, permitindo o desenvolvimento de trabalhos em grupos, na forma de projetos, e a construção do conhecimento cooperativo. Seriam introduzidas interfaces pré-definidas com os principais módulos do simulador, de forma que novos módulos pudessem ser alterados e/ou implementados conforme a realidade a ser simulada;
- Com a evolução do ensino à distância, é possível imaginar um curso de sistemas operacionais totalmente não-presencial, utilizando apenas a Internet como meio de comunicação entre alunos e professor que poderiam estar geograficamente distantes. Neste caso, o simulador teria um papel muito mais importante no processo de ensino do que apenas uma ferramenta de apoio [DRISCOLL, 1998] [HALL, 1998].

Apesar da maioria dos tópicos sugeridos apresentarem um certo grau de complexidade para o desenvolvimento por alunos principiantes, é possível que em cursos de graduação e pós-graduação alguns dos temas propostos possam ser implementados.

REFERÊNCIAS BIBLIOGRÁFICAS

- ANDERSON, T., LEVY, H., BERSHAD, B., LAZOWSKA, E. **The Interaction of Architecture and Operating System Design**. ACM SIGPLAN Notices, vol. 26, n° 4, pp. 108-120, April 1991.
- ANDERSON, T.E., CHRISTOPHER, W.A., PROCTER, S.J. **The Nachos Instructional Operating System**. Disponível na Internet em <http://www.cs.washington.edu/homes/tom/nachos/>, 1999.
- BACH, M.J. **The Design of Unix Operating System**. Prentice-Hall, 1986.
- BEN-ARI, M. **Principles of Concurrent and Distributed Programming**. Prentice-Hall, 1990.
- BOOCH, G. **Object-Oriented Analysis and Design with Applications**. 2ª Ed., Addison-Wesley, 1994.
- BORLAND. **Object Pascal Language Reference**. Borland, 1999.
- BORLAND. **Developer's Guide**. Borland, 1999.
- BORLAND. **Programming with Delphi**. Borland, 1999.
- BORLAND. **Visual Component Library (VCL)**. Borland, 1999.
- BORLAND. **Win32 API Documentation**. Borland, 1999.
- BRENDA, M. **Instructional Design and Learning Theory**. Disponível na Internet em <http://www.usask.ca/education/coursework/802papers/mergel/brenda.htm>, 1998.

BYNUM, B., CAMP, T. **After You, Alfonse: A Mutual Exclusion Toolkit**. Proceedings of the 27th SIGCSE Technical Symposium on Computer Science Education, Feb. 1996.

BYNUM, B., CAMP, T. **After You, Alfonse: A Mutual Exclusion Toolkit - An Introduction to BASI**. Disponível na Internet em http://www.mines.edu/fs_home/tcamp/baci/, 1999.

CALVERT, C. **Delphi Unleashed**. Sams Publishing, 1996.

DENNING, P. **The Working Set Model for Program Behavior**. Communications of the ACM, vol. 11, n° 5, Mai 1968.

DENNING, P. **Virtual Memory**. Computer Surveys, vol. 2, n° 3, Dec. 1970.

DENNING, P. **Third Generation Computer Systems**. Computer Surveys, vol. 3, n° 4, Dec. 1971.

DENNING, P. **Working Sets Past and Present**. IEEE Transactions on Software Engineering, vol. SE-6, n° 1, Jan. 1980.

DRISCOLL, M. **Web-based Training: Tactics and Techniques for Designing Adult Learning**. Jossey-Bass/Pfeiffer, 1998.

EDWARDS, S. D., KING, D. R., WINBLAD, A. L. **Object-Oriented Software**. Addison-Wesley, 1990.

FREEBSD. **FreeBSD Handbook**. Disponível na Internet em <http://www.freebsd.org/handbook/index.html>, 2000.

HALL, B. **Web-based Training Cookbook**. John Wiley & Sons, 1997.

- HANLEY, S. **On Constructivism**. <http://www.inform.umd.edu/UMS+State/UMD-Projects/MCTP/Essays/Constructivism.txt>, 1994.
- HERROD, S.A., **Using Complete Machine Simulation to Understand Computer System Behavior**. Tese de D.Ph., Stanford University, Feb. 1998.
- JUNQUEIRA, L. MESQUITA, F., ARAÚJO, M. **CPUsim – Simulador da Arquitetura de um Processador Acadêmico**. Projeto de Final de Curso, Depto de Ciência da Computação da UFRJ, 1997.
- KIFER, M., SMOLKA, S. **OSP: An Environment for Operating Systems**. Addison-Wesley, 1991.
- KIFER, M., SMOLKA, S. **OSP: An Environment for Operating Systems (Instructor Version)**. Addison-Wesley, 1991.
- KLEIMAN, S., SHAH, D., SMALDERS, B. **Programming with Threads**. SunSoft Press, Prentice-Hall, 1996.
- LEVY, H.M., ECKHOUSE, R.H. **Computer Programming and Architecture: The VAX-11**. Digital Press, 1980.
- MACHADO, F.M., MAIA, L.P. **Introdução à Arquitetura de Sistemas Operacionais**. LTC, 1992.
- MACHADO, F.M., MAIA, L.P. **Arquitetura de Sistemas Operacionais**. 2º ed., LTC, 1997.
- MAIA, L.P. **Página do Livro Arquitetura de Sistemas Operacionais**. Disponível na Internet em <http://home.ismnet.com.br/~lpmaia/aso.htm>, 1998.

- MAIA, L.P. **Multithread**. Monografia da disciplina Laboratório de Sistemas Operacionais II, NCE/UFRJ, 1999.
- MELO, P.T. **Reclificação de Trabalhadores e Formação à Distância no Ensino Médio**. Tese de M.Sc. COPPE/UFRJ, Fev. 1999.
- MILLER, D.D. **VAX/VMS Operating System Concepts**. Digital Press, 1992.
- NICHOLS, B., BUTTAR, D., FIRRELL, J. **Pthreads Programming**. O'Reilly Press, 1996.
- PHAM, T.Q., GANG, P.K. **Multithreaded Programming with Windows NT**. Prentice-Hall, 1996.
- PLAT, D.S. **Windows 95 and NT: Win32 API from Scratch: A Programmer's Workbook**. Prentice-Hall, 1996.
- RICHTER, J. **Advanced Windows**. 3° ed., Microsoft Press, 1997.
- ROBBINS, K.A., ROBBINS, S. **Practical Unix Programming: A Guide to Concurrency, Communication, and Multithreading**. Prentice-Hall, 1996.
- SILBERSCHATZ, A., GALVIN, P. **Operating System Concepts**. 5° ed., Addison-Wesley, 1998.
- SOLOMON, D.A. **Inside Windows NT**. 2° ed., Microsoft Press, 1998.
- STALLINGS, W. **Operating Systems: Internal and Design Principles**. 3° ed., Prentice-Hall, 1997.
- TANENBAUM, A.S. **Modern Operating Systems**. Prentice-Hall, 1992.

TANENBAUM, A.S. **MINIX Information Sheet**. Disponível na Internet em <http://www.cs.vu.nl/~ast/minix.html>, 1996.

TANENBAUM, A.S., WOODHULL, A.S. **Operating Systems: Design and Implementation**. Prentice-Hall, 1997.

TROPIX. **Projeto TROPIX**. Disponível na Internet em <http://www.tropix.nce.ufrj.br>, 2000.

VAHALIA, U. **Unix Internals: The New Frontiers**. Prentice-Hall, 1996

VALENTE, J.A. **Diferentes Usos do Computador na Educação**. nº 57, jan/mar, 1993.

VOSS G. **Object-Oriented Programming: An Introduction**. McGraw-Hill, 1991.

WIRFS-BROCK, R., WILKERSON, B., WIENER, L. **Designing Object-Oriented Software**. Prentice-Hall, 1990.