UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

SIMULADOR DE MECANISMOS DE GERÊNCIA DE MEMÓRIA REAL E VIRTUAL

GUSTAVO MORITZ

BLUMENAU
2004

2004/2-20

GUSTAVO MORITZ

SIMULADOR DE MECANISMOS DE GERÊNCIA DE MEMÓRIA REAL E VIRTUAL

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. Mauro M. Mattos - Orientador

BLUMENAU
2004

2004/2-20

SIMULADOR DE MECANISMO DE GERÊNCIA DE MEMÓRIA REAL E VIRTUAL

Por

GUSTAVO MORITZ

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Mauro Marcelo Mattos, Dr. – Orientador, FURB
Membro:	Prof. Antonio Carlos Tavares, FURB
Membro:	Prof. Rômulo Silva de Oliveira, UFSC

Blumenau, 09 de dezembro de 2004

Dedico este trabalho aos meus pais pelo incentivo, carinho e amor que nunca me faltaram, e também pela paciência em esperar o primeiro filho adquirir o grau superior.

Os bons livros fazem “sacar” para fora o que a pessoa tem de melhor dentro dela.

Lina Sotis Francesco Moratti

AGRADECIMENTOS

À Deus, pelo seu imenso amor e graça.

Aos meus pais, Osnir e Ivete, por me amarem, por estarem sempre ao meu lado em todos os momentos, pelos conselhos, apoio e incentivo à conclusão deste curso.

À minha namorada, Fabiana, pelo amor e compreensão em todos os momentos, principalmente na minha ausência no decorrer deste trabalho.

Aos meus amigos, pelos empurrões e cobranças.

Ao meu orientador, Mauro Marcelo Mattos, por ter acreditado na conclusão deste trabalho.

RESUMO

O presente trabalho descreve a aplicação de um simulador didático de mecanismo de gerência de memória em sistemas operacionais. O software foi concebido de forma a exemplificar o funcionamento de alguns mecanismos comuns e serve como ferramenta de apoio ao ensino.

Palavras chaves: Simulador; Gerência de Memória.

ABSTRACT

This work describes a teaching tool to simulate memory management mechanisms behanor in operatiny systems. The software has been conceived as a way help operatiny systens students to learn about concepts.

Key-Words: Simulator; Abstract; Manages of Memory.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

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.

Na maioria dos cursos de ciência da computação, quer sejam graduação, mestrado, pós-graduação ou extensão é oferecida uma ou mais disciplinas 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 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 (MAIA, 2001, p. 13).

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 os 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, que envolve tabelas de mapeamento e tradução de endereços, é extremamente difícil de ser apresentado de forma clara, devido à sua dinâmica e complexidade.

Num ambiente de multiprogramação, diversos processos são executados simultaneamente, através da divisão do tempo do processamento. Para que o chaveamento entre eles seja rápido, esses processos devem estar na memória, prontos para executar. É função do mecanismo de gerência de memória do sistema operacional prover os mecanismos

necessários para que os diversos processos compartilhem a memória de forma segura e eficiente (OLIVEIRA, 2000, p. 97).

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.

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 (MAIA, 2001, p. 45).

O presente trabalho tem por objetivo a construção de um simulador gráfico que sirva de ferramenta visual de suporte efetivo para o ensino e aprendizado dos conceitos e técnicas de gerência de memória virtual implementados nos sistemas operacionais atuais, objetivando tornar este processo de ensino mais fácil e eficaz.

1.1 OBJETIVOS DO TRABALHO

O objetivo do trabalho proposto é desenvolver um protótipo de software para demonstrar graficamente o funcionamento dos principais algoritmos de gerência de memória real e virtual.

Os objetivos específicos do trabalho são:

a) construção de um modelo que permita a configuração da estrutura da máquina simulada, do processador, dos processos e dos algoritmos de gerência de memória a ser demonstrados;
b) construção de um modelo de interface que viabilize o acompanhamento visual do modelo do computador simulado.

1.2 ESTRUTURA DO TRABALHO

O presente trabalho está subdividido em capítulos que serão explicitados a seguir.

O primeiro capítulo apresenta a contextualização e justificativa para o desenvolvimento do trabalho desenvolvido.

O segundo capítulo trata do papel da informática na educação, destacando os benefícios e tipos de software para auxilio na educação.

O terceiro capítulo aborda os aspectos de gerência de memória, detalhando alguns conceitos importantes sobre: partições fixas, variável, memória virtual.

O quarto capítulo trata sobre o desenvolvimento do trabalho, apresentando os diagramas de classe, casos de uso e diagramas de seqüência. Este capítulo também descreve a implementação do protótipo e os resultados obtidos.

Finalmente, o capítulo quinto trata das conclusões sobre o trabalho apresentado e oferece sugestões para trabalhos futuros.

2 A INFORMÁTICA NA EDUCAÇÃO

Segundo Komosinski (2000), o ensino é um processo complexo de relações entre as pessoas envolvidas (tipicamente professores e estudantes) onde a tecnologia sempre está presente. Na situação típica, giz, quadro negro, livro, papel, caneta, lápis e borracha são as tecnologias disponíveis para professores e estudantes.

Durante toda a história da educação, a sociedade sempre buscou novas tecnologias que permitissem o aperfeiçoamento do processo educacional. “Como, por definição, as tecnologias mudam ao longo do tempo, mudaram também os artefatos estudados. Rádio, televisão, slides, transparências, vídeo cassete, etc., todos tiveram seus ‘momentos de fama’. Acreditava-se, erroneamente, que eles iriam revolucionar a educação”, Komosinski (2000).

Segundo Tajra (1998, p. 22), a tecnologia na educação não é uma ciência, mas uma disciplina orientada para prática controlável e pelo método científico, a qual recebe contribuições das teorias de psicologias da aprendizagem.

> Com o crescente processo de transformação, que ocorre a cada instante na área da informática, precisamos atualizar nossos conhecimentos, para o desenvolvimento de novas técnicas de aprendizagem. Uma dessas técnicas é o desenvolvimento de softwares educativos. (SILVA, 2000, p. 6).

Conforme Tajra (1998, p. 49), “por meio dos softwares é que podemos ensinar, aprender ou, simplesmente, produzir trabalhos com excelentes apresentações”. Segundo Mattos (1999), todos têm os mesmos instrumentos para chegar ao conhecimento, mas não os utilizam com a mesma intensidade. Normalmente, os processos educacionais baseiam-se, quase que exclusivamente, no desenvolvimento da inteligência lingüística e lógico-matemática, deixando de lado as outras formas de acesso ao conhecimento.

Informação refere-se a uma representação exteriorizada (por meio de sons, imagens, gestos, etc.) de fatos experimentados enquanto conhecimento significa uma internalização destes fatos pelo ser humano. O novo desafio é, portanto, transformar o objetivo da educação de pesquisa e aprendizagem da informação em construção do conhecimento. Na verdade, o estudante deixa de ser ensinado, mas encontra condições para aprender, construindo o seu conhecimento. O uso de novas tecnologias nesta nova abordagem deveria explorar suas particularidades e possibilidades de trocas qualitativas na rotina envolvendo sala de aula, aluno e professor. As trocas seriam na essência, e não somente uma apresentação mais agradável de conteúdos tradicionais (CYSNEIROS apud SLOCZINSKI, 2000).

Todo software ou programa de computador como é conhecido, possui o conhecimento sobre um determinado problema ou processo a ser percorrido e resolvido. É possível encontrar este conhecimento nos algoritmos empregados pelo software e nos procedimentos de decisão que determina qual é o algoritmo melhor para utilizar em determinada situação.

Conforme Silva (2000, p.10), as justificativas para introdução dos computadores na educação são diversificadas e, na medida em que aumenta a intimidade dos alunos e professores com os novos recursos, esses podem e devem ser expandidos.

Bizzotto (2003) informa que, em sua grande maioria, o processo de aprendizagem é "centralizado" no professor. O conteúdo passado através de um software é entendido como um "produto completo e acabado", não dando ao aluno a possibilidades de adaptação do material as suas necessidades.

Em função dessa "centralização" e "inflexibilidade" Bizzoto (2003) observa que "os ambientes de aprendizagem criados (softwares educacionais, páginas da internet, etc) possuem, em sua grande maioria" características como: o reforço do papel passivo do aluno na recepção do conteúdo, a falta de incentivo ao trabalho cooperativo, não evolui em função das necessidades do aluno, do professor ou do grupo e "não permitem que o aluno seja autor de seu próprio trabalho".

O processo de aprendizagem "deve ser sempre aberto para a imprevisibilidade", salienta Bizzotto (2003), afirmando que o ambiente de aprendizagem como ferramenta de auxílio no processo de aprendizagem deve estar aberto a imprevisibilidade, dando suporte ao aluno no "trato com o devir". Assim, o ambiente permite que o aluno tenha uma participação ativa no processo de aprendizagem, evoluindo em termos de criatividade, tornando-se uma pessoa mais pró-ativa.

Campos (1994) afirma que no Brasil, a primeira iniciativa na área de informática na educação aconteceu em 1981 em Brasília, no I Seminário de Informática na Educação que foi liderada pela SEI em esforço conjunto com o Ministério da Educação – MEC e o Conselho Nacional de Desenvolvimento Científico Tecnológico – CNPq. O I Seminário de Informática na Educação foi dirigido a pesquisadores em educação, psicologia, sociologia e informática, além de representantes da Sociedade dos Usuários de Computadores e Equipamentos Subsidiários – SUCESU, salienta Campos (1994).

2.1 FERRAMENTAS DE APOIO AO ENSINO

Segundo Maia (2001) 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, lógica 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.

Conforme Maia (2001) 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.

Foram criação de vários tipos de software, como tutoriais, exercício-e-prática, avaliações, jogos e simulações:

a) programas tutorias: é 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;
b) programas exercício-e-prática: 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;
c) simulações: 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, 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 (MAIA, 2001).

Conforme Maia (2001) 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.

Segundo Maia (2001) 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.

2.2 TRABALHOS CORRELATOS

Existem muitos trabalhos desenvolvidos que podem ser considerados como ferramentas de apoio ao ensino, as quais podem ser consideradas como trabalhos correlatos a esse.

O trabalho apresentado por Maia (2001), permite que o professor apresente os conceitos e mecanismos de um sistema operacional multiprogramável e/ou multitarefa, como *Unix*, *OpenVMS* e *Windows*, de forma simples e animada. O simulador permite visualizar os

conceitos de multiprogramação, processo e suas mudanças de estado, gerência do processador (escalonamento) e a gerência memória virtual. A partir das opções de configuração, é possível selecionar diferentes políticas e alterar o funcionamento do simulador. As principais características deste simulador é a implementação do conceito de processos como criação de processos *CPU-bound* e *IO-bound*, visualização do *Process Control Block* , suspender/resumir e eliminar processos, permite visualizar estruturas internas do sistema como *Process Page Table*, *Page Table Entry*, gerência de processador através de escalonamento circular com prioridades e gerência de memória.

Weber (2001), descreve um simulador que foi desenvolvido para computadores PC e compatíveis, que permite a execução e depuração de programas em linguagem de máquina escritos para os computadores *Neander*, *Ahmes*, *Ramses e Cesar*.

3 GERÊNCIA DE MEMÓRIA

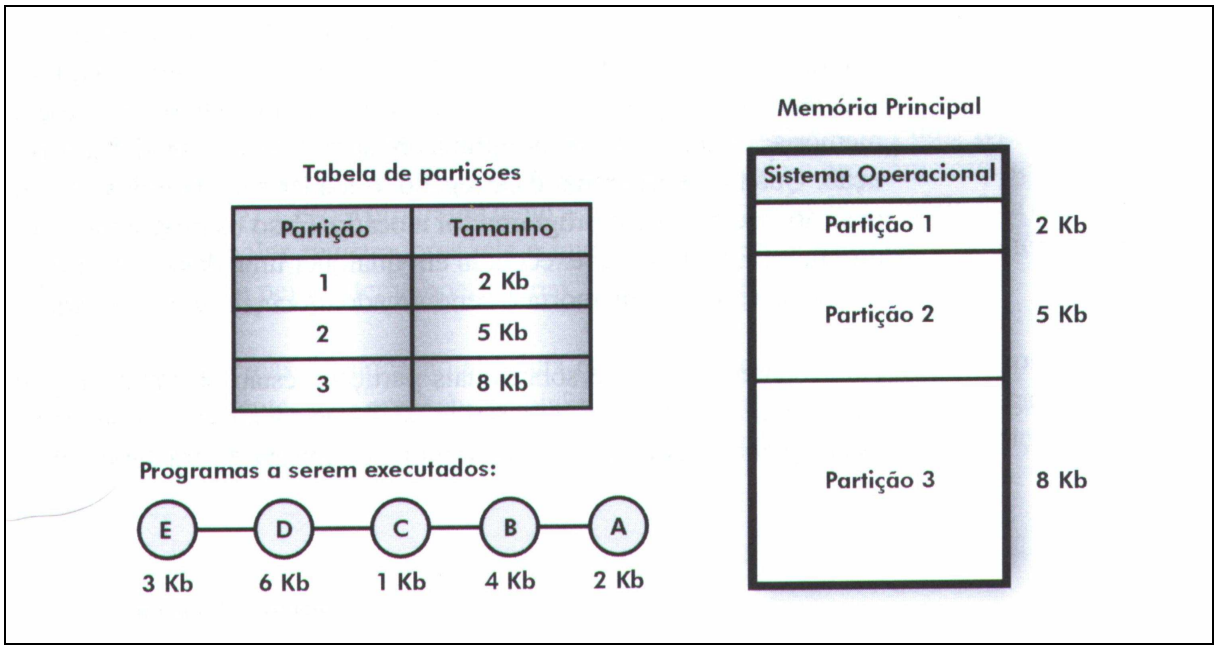
Uma vez que diversos processos podem ocupar a memória ao mesmo tempo, torna-se necessário um gerenciamento adequado de modo a otimizar a utilização da mesma. O componente do sistema operacional responsável por esta tarefa é o gerente de memória, que entre outras atribuições deve controlar quais partes da memória estão sendo usadas, alocar memória de acordo com as necessidades de cada processo, liberar memória alocada a um processo quando este termina sua execução, transferência de processos ou partes destes entre a memória principal e secundária.

Conforme Maia (2001), as diversas abordagens propostas para alocação e restrição do espaço de endereçamento envolvem a utilização de registradores que definem os limites dos endereços de memória que um processo poderá acessar. O significado do valor armazenado nos registradores pode variar um pouco de uma técnica para outra, mas a proteção é sempre feita através de comparações do valor do endereço desejado pelo processo para acesso com o endereço armazenado no(s) registrador(es), e caso o endereço esteja fora do espaço autorizado para o processo, uma interrupção é gerada indicando uma violação da memória e o processo pode ter sua execução abortada. As técnicas de gerenciamento de memória bastante conhecidas: partições fixas, partições variáveis, paginação e memória virtual.

3.1 PARTIÇÕES FIXAS

Segundo Oliveira, Carissimi e Toscani (2000), partições fixas são a forma mais simples de gerenciamento de memória para ambientes de multiprogramação. A memória é dividida em um número fixo de blocos, eventualmente de diversos tamanhos. À medida que os processos são submetidos à execução (processos no estado pronto) o gerenciador de memória localiza uma partição livre cujo tamanho atenda às necessidades do processo.

Esta abordagem, apesar de simples implementação, tem um aspecto importante que é o tamanho das partições.(figura 1) Se as partições forem muito maiores que as necessidades dos processos em geral, ocorre um desperdício de memória chamado fragmentação interna. A fragmentação interna caracteriza-se por espaços não utilizados dentro das partições. Por outro lado, se as partições forem menores que o devido, alguns processos poderão ser impedidos de executar devido a falta de partição de tamanho adequado, apesar de haver memória suficiente só que distribuída em pequenas partições (MACHADO; MAIA, 2002).



Fonte: Machado e Maia (2002)

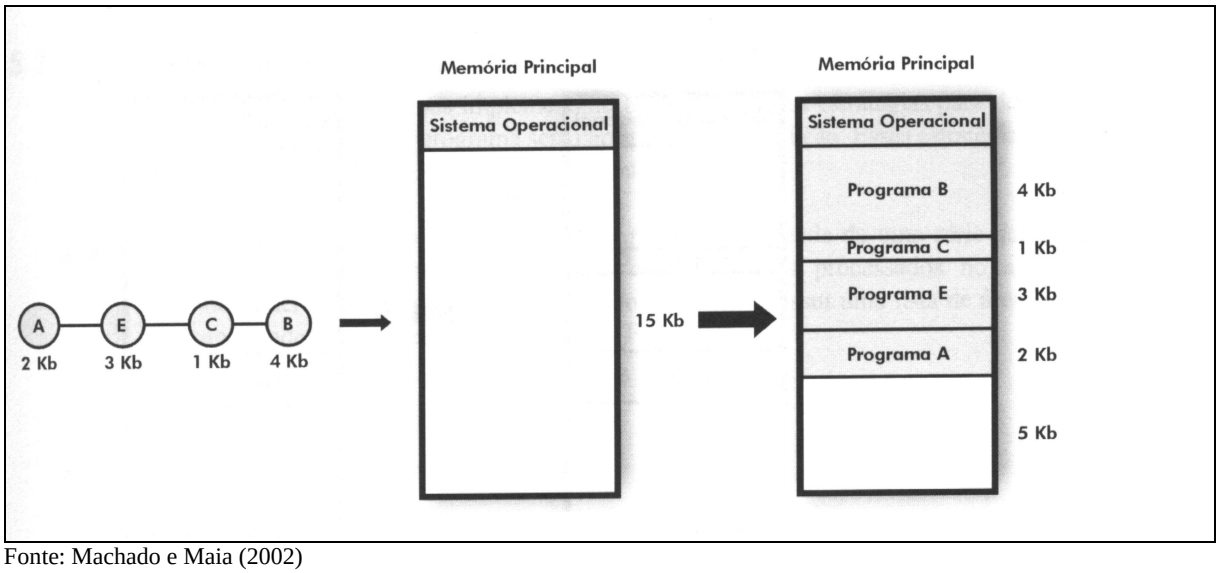
Figura 1 – Tamanho das Partições.

3.2 PARTIÇÕES VARIÁVEIS

Quando partições variáveis são empregadas, o tamanho das partições é ajustado dinamicamente as necessidades exatas dos processos. Essa é uma técnica de gerência mais flexível que partição fixa (OLIVEIRA; CARISSIMI; TOSCANI,2000 p.101).

O uso de partições de memória de tamanho variável resolve o problema da fragmentação interna. Outro problema da utilização das partições fixas relaciona-se com processos que variam de tamanho durante sua execução, que podem ser impedidos de continuar sua execução, caso a partição em que está alocado não comporte um aumento no seu tamanho (MACHADO; MAIA, 2002).

No método de partições variáveis a memória não é dividida previamente em partições, mas estas são criadas à medida que os processos são submetidos à execução, sendo neste momento criadas as partições.(figura 2) No entanto, quando os processos vão terminando suas execuções, ocorre a chamada fragmentação externa, pois os espaços livres são preenchidos por novos processos, que dificilmente serão do tamanho exato da partição. Assim pequenos buracos vão surgindo na memória (MACHADO; MAIA, 2002).



Fonte: Machado e Maia (2002)

Figura 2 – Partição Variável.

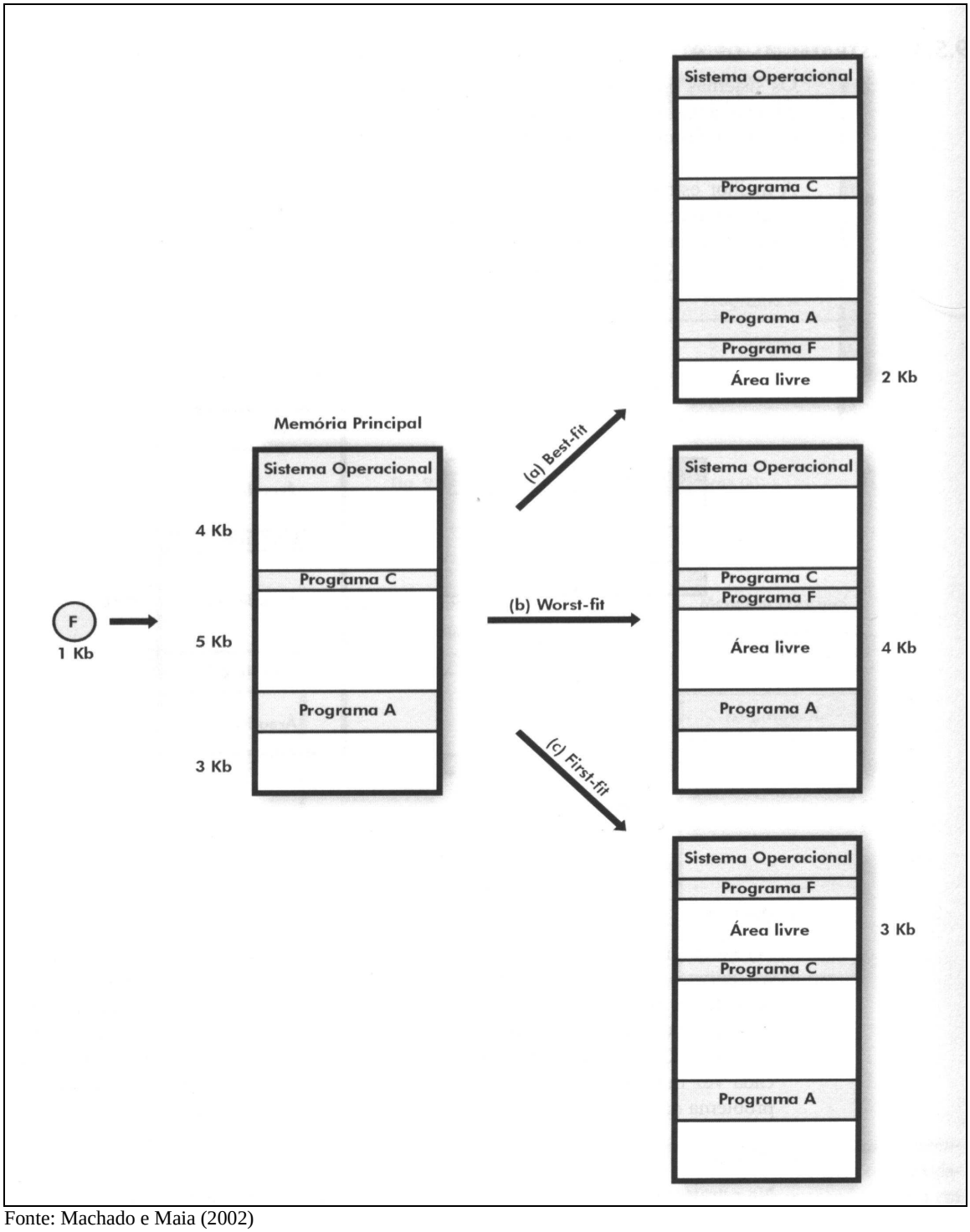
Segundo Machado e Maia (2002) quando um processo é criado, a lista de lacunas é percorrida. Será usada uma lacuna de tamanho maior ou igual ao tamanho do processo. No entanto, o que a lacuna original tiver a mais que o necessário para executar será transformado em uma nova lacuna, apenas menor que a original, assim o processo receberá o tamanho exato de memória. Existem três algoritmos básicos para percorrer a lista de lacuna atrás de uma lacuna de tamanho suficiente:

a) *first-fit* aloca o processo na primeira partição que encontrar com tamanho suficiente para atender ao processo (figura 3a).Ao alocar um processo em uma partição, esta é dividida sendo usada a parte necessária com a ocupação do processo e o restante passa a ser considerada como uma nova partição, sendo colocada na lista de partições livres;

b) *best-fit* procura-se a partição de tamanho mais próximo ao tamanho do processo (figura 3b). Para essa abordagem ser mais efetiva, a lista de partições livres deve ser mantida ordenada. Entretanto, esse algoritmo tem o inconveniente de aumentar a fragmentação externa, pois como os processos têm tamanhos próximos aos tamanhos das partições, em geral sobrará pouco espaço, ou seja, pequenas partições espalhadas pela memória que provavelmente serão insuficientes para conter novos processos. Observe que a soma dessas partições podem muito bem ser suficiente para a alocação de um novo processo que, entretanto, terá sua execução impedida por não haver memória disponível;

c) *worst-fit* procura-se a partição que tenha maior tamanho (figura 3c). Para essa abordagem ser mais efetiva, a lista de partições livres deve ser mantida ordenada.

Esse algoritmo em geral sobrará maiores espaços, ou seja, grandes partições na memória que provavelmente serão utilizadas por processos menores.



Fonte: Machado e Maia (2002)

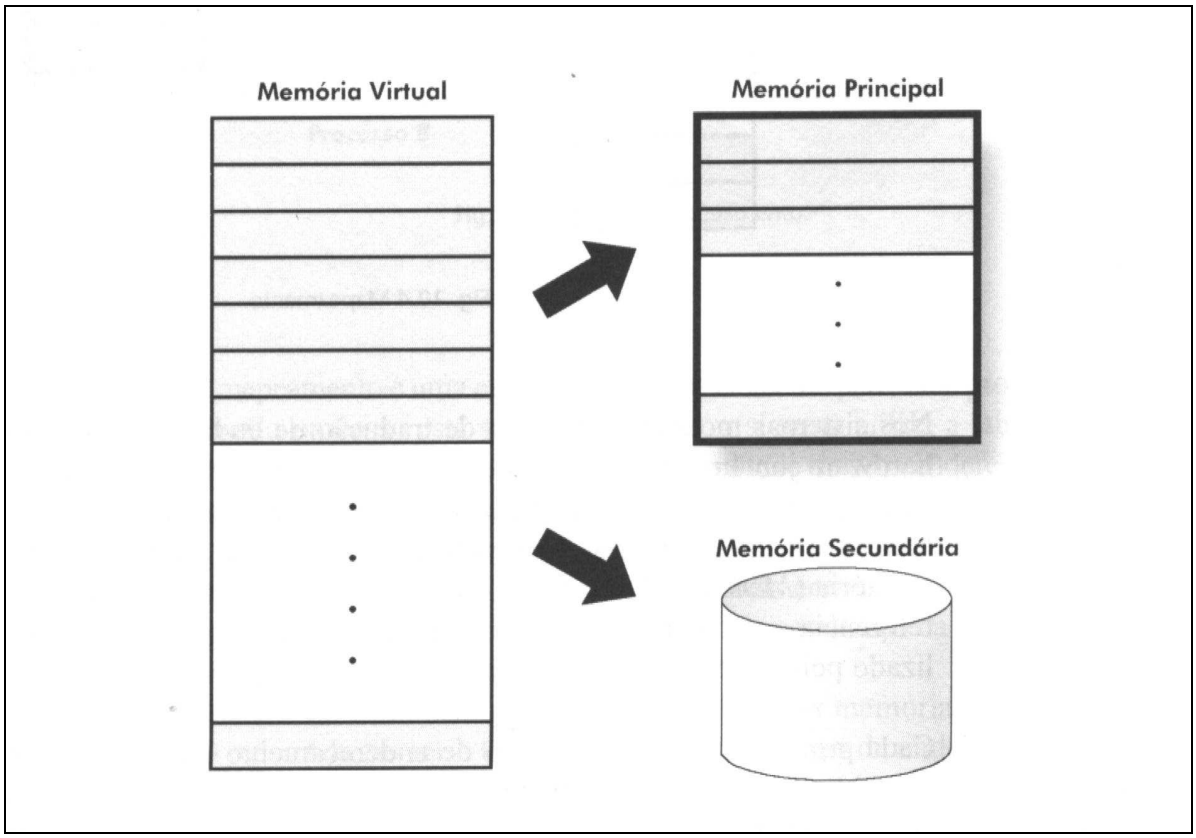
Figura 3 – Estratégia para a escolha da partição.

3.3 MEMÓRIA VIRTUAL

A memória virtual é a separação da memória lógica do usuário da memória física.Essa separação permite que uma memória virtual extremamente grande seja fornecida para os programadores quando apenas uma memória física esteja disponível (SILBERSHATZ; GALVIN; GAGNE, 2000 p. 211).

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 mapeado 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 (MAIA, 2001, p. 45).

Conforme Maia (2001, p.47), 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 (figura 4).



Fonte: Machado e Maia (2002)

Figura 4 - Espaço de endereçamento virtual.

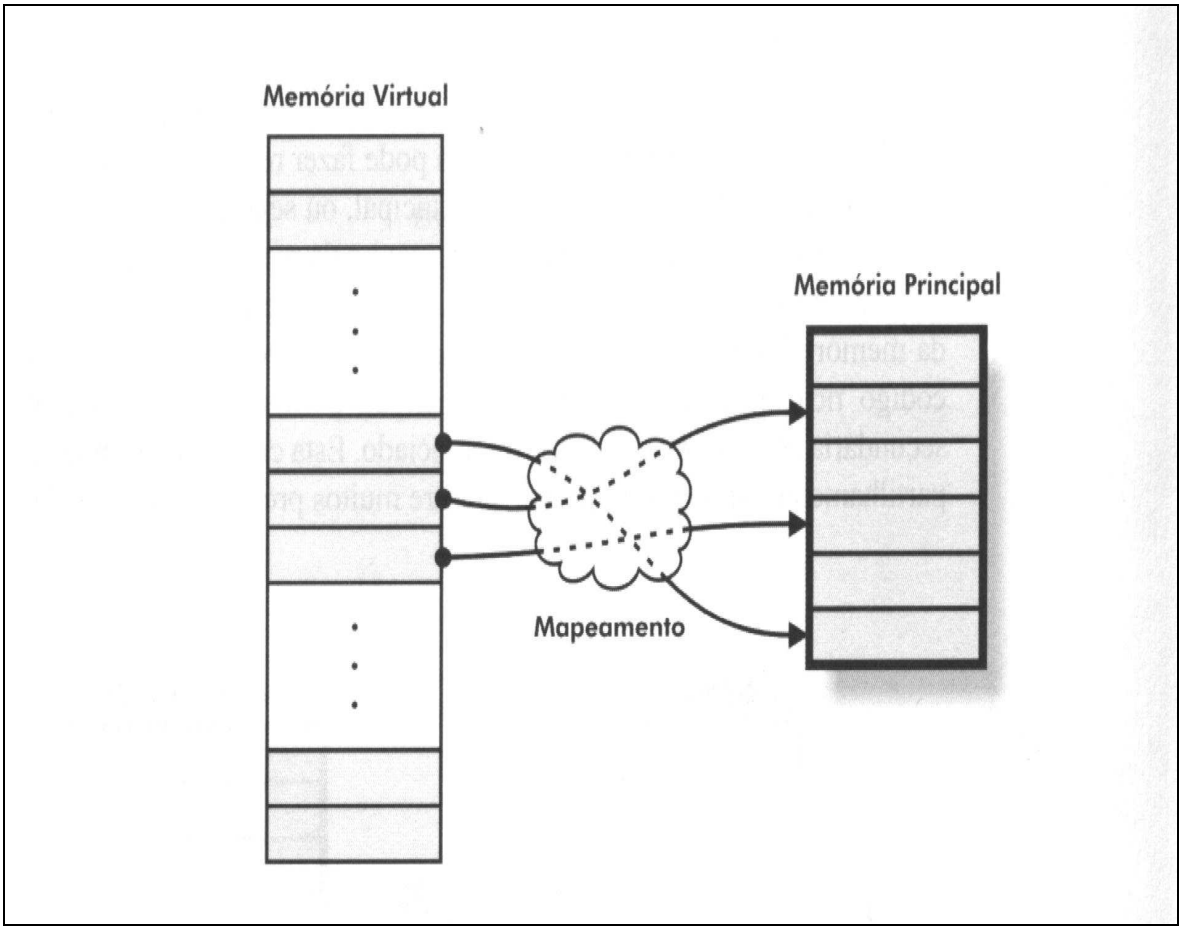
Outra vantagem da memória virtual é permitir um número maior de processos compartilhando a memória, já que apenas algumas partes de cada processo estarão residentes. Isto leva a uma utilização mais eficiente também do processador, permitindo um maior número de processos no estado de pronto (OLIVEIRA; CARISSIMI; TOSCANI, 2000).

A memória virtual também é possível em sistemas com multiprogramação, com pedaços e partes de diferentes programas simultaneamente na memória. Se um programa estiver esperando por outra parte de si próprio ser carregado na memória, ele estará conseqüentemente esperando por E/S, e não estará apto a ser executado, de modo que a CPU poderá ser entregue a outro processo, como acontece em qualquer sistema com multiprogramação (TANENBAUM, 2003, p. 149).

3.3.1 MAPEAMENTO

O mapeamento permite ao sistema operacional traduzir um endereço localizado no espaço de endereçamento virtual do processo para um endereço no espaço real

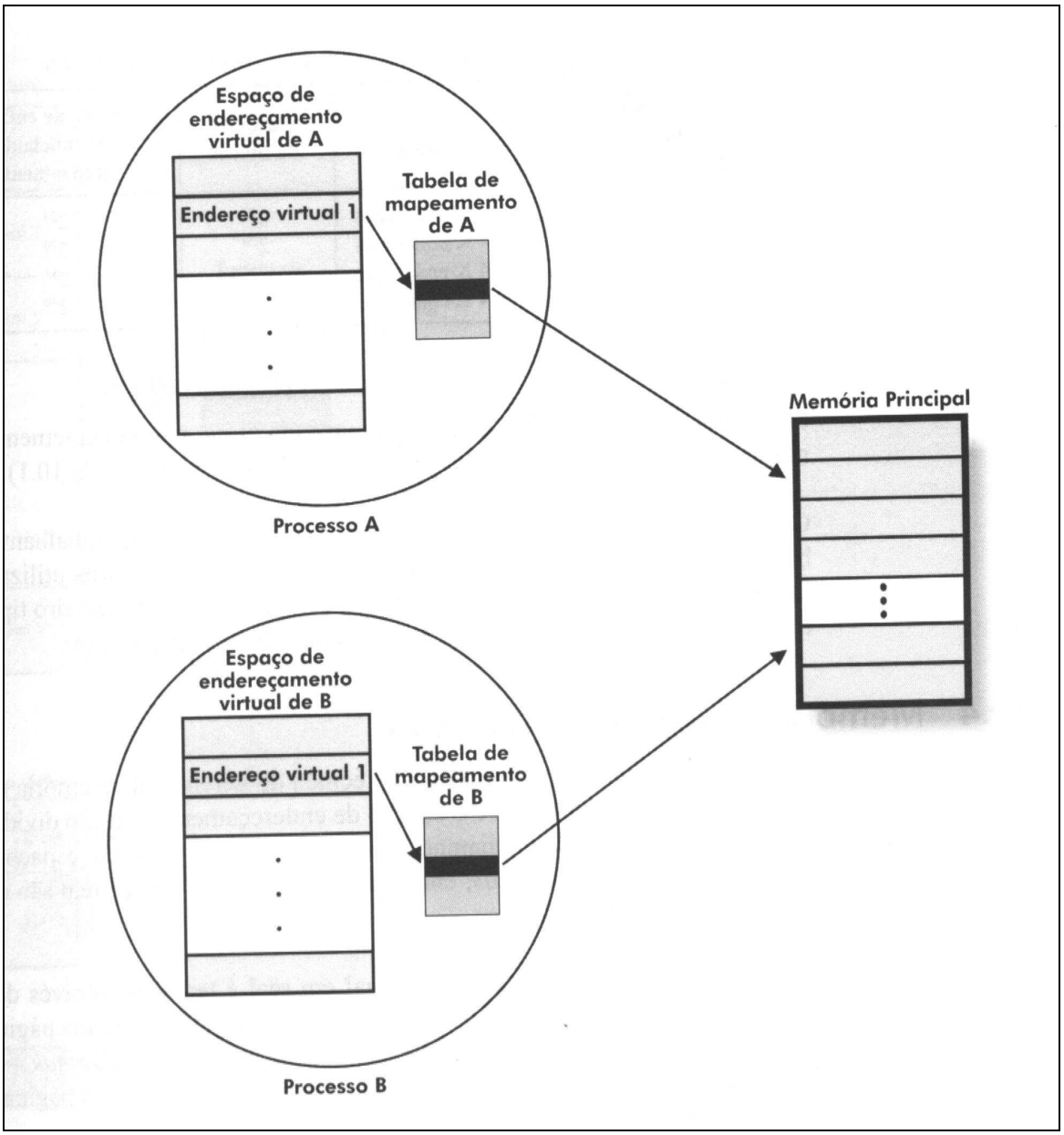
(figura 5). Como conseqüência do mapeamento, um programa não precisa estar necessariamente contíguo na memória principal para ser executado (MACHADO; MAIA, 2002, p. 174).



Fonte: Machado e Maia (2002)

Figura 5 - Mapeamento.

Segundo Maia (2001) 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 (figura 6). 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.



Fonte: Machado e Maia (2002)

Figura 6 - 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 (MAIA, 2001).

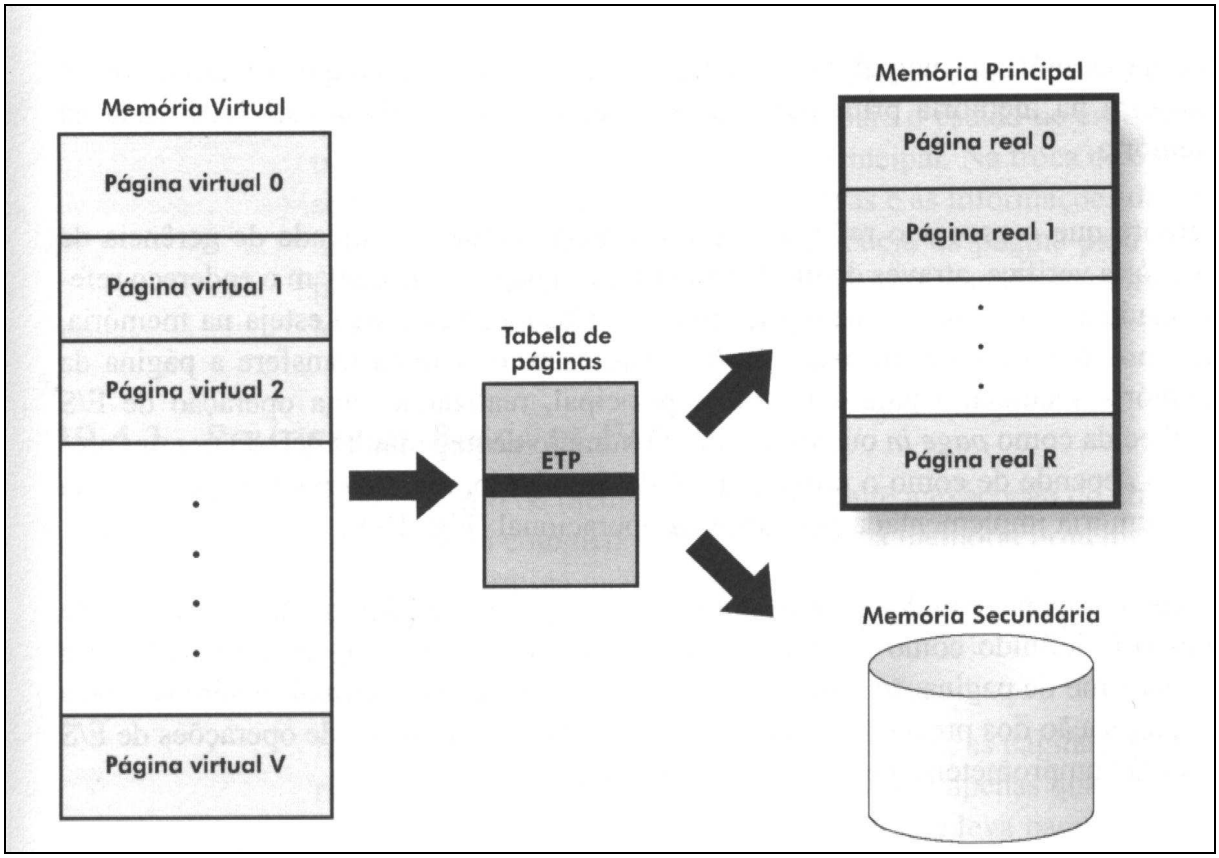
Caso o mapeamento fosse realizado para cada célula na memória principal, o espaço ocupado pelas tabelas na memória real seria tão grande quanto o espaço de endereçamento virtual de cada processo, o que inviabilizaria a implementação do mecanismo de memória virtual. Em função disso, as tabelas mapeiam 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 (MACHADO; MAIA, 2002).

3.3.2 PAGINAÇÃO

Paginação 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 lógicas, enquanto as páginas no espaço real são chamadas de páginas físicas (MACHADO; MAIA, 2002, p.176).

Segundo Maia (2001) 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, com informações de mapeamento que permitem ao sistema localizar a página real correspondente na memória principal (figura 7).



Fonte: Machado e Maia (2002)

Figura 7 - 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, na tabela de páginas da tabela do processo, o endereço físico do *frame* (MACHADO; MAIA, 2002).

Além da informação sobre a localização da página virtual, o PTE (page table entry) 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 (SILBERSHATZ; GALVIN; GAGNE, 2000).

Quando um processo faz referência a um endereço e ocorre uma exceção de *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 escalonado poderá continuar seu processamento.

3.3.3 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 determina quando uma página deve ser trazida para a memória principal. Existem, basicamente, duas alternativas: paginação por demanda e paginação antecipada (MACHADO; MAIA; 2002).

Na paginação por demanda, 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 (OLIVEIRA; CARISSIMI; TOSCANI, 2000).

Na paginação antecipada, 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 (MAIA, 2001).

3.3.4 POLÍTICA DE ALOCAÇÃO DE PÁGINAS

Segundo Maia (2001) 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:

a) alocação fixa: cada processo recebe um número máximo de páginas que pode ser utilizado. Se o número de páginas for insuficiente, o processo gera uma exceção de *page fault* e cede 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, no início da sua execução.
b) 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 reduzí-las, ao mesmo tempo em 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*.

3.3.5 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 (MAIA, 2001).

Segundo Maia (2001) 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 (MAIA, 2001).

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 (MACHADO; MAIA, 2002).

Conforme Machado e Maia (2002) 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*).

3.3.6 POLÍTICAS DE SUBSTITUIÇÃO DE PÁGINAS

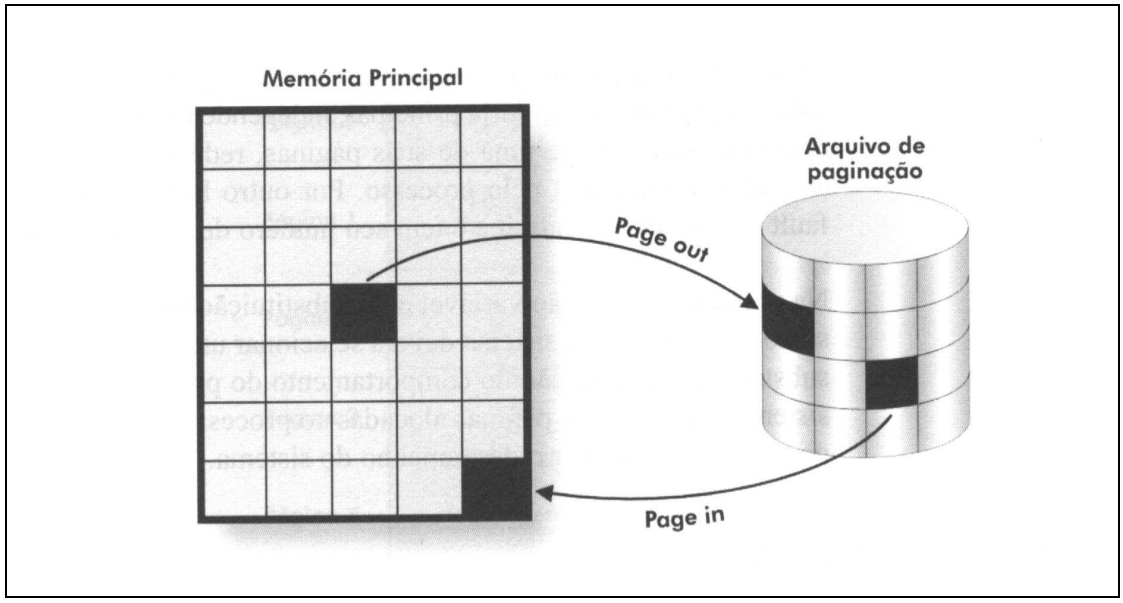
O maior problema na gerência de memória virtual por paginação não é decidir que página carregar para a memória, mas quais páginas remover. Quando não existem páginas livres disponíveis na memória e novos *frames* devem ser alocados, a política de

substituição (*replacement policy*) de páginas determina, dentre as diversas páginas residentes, quais devem ser realocadas (SILBERSHATZ; GALVIN; GAGNE, 2000).

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, preservando seu conteúdo. O sistema mantém um arquivo de paginação onde as páginas modificadas são armazenadas. Sempre que uma destas páginas for novamente referenciada, ela será trazida novamente para a memória principal (MACHADO; MAIA, 2002).

O sistema consegue implementar esse mecanismo através do bit de modificação, 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 (MAIA, 2001).

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* (figura 8) são candidatas a realocação. Já na política global, todas as páginas residentes são avaliadas, independente do processo que gerou o *page fault* (MAIA, 2001).



Fonte: Machado e Maia (2002)

Figura 8 – Substituição de páginas.

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 (MAIA, 2001).

Existem diversos algoritmos na literatura voltados para a implementação da política de substituição de páginas:

a) *first in first out* (FIFO é o mais simples, onde a cada página é associada um *timestamp* (registro de tempo). Aquela página que estiver na memória mais tempo será selecionada. Apesar de simples implementação, este algoritmo não é muito eficiente, pois páginas que estão mais tempo na memória podem ser justamente aquelas que são freqüentemente acessadas. Esse algoritmo pode ser implementado também através de uma fila simples, selecionando-se a primeira página da fila para expulsão da memória (OLIVEIRA; CARISSIMI; TOSCANI, 2000).

b) *least recently used* (LRU) seleciona a página que por mais tempo não foi acessada. Para sua implementação utiliza uma pilha, que conterá as páginas organizadas de forma que as páginas que foram acessadas recentemente encontram-se no topo da pilha e aquelas que foram acessadas a mais tempo atrás ficam no fundo da pilha. Assim, toda vez que o processo acessar uma determinada página, esta é movida para o topo da pilha. Quando houver a necessidade de troca de página, a do fundo da pilha será selecionada para expulsão da memória. Apesar de ser um bom algoritmo, sua implementação não é muito simples e, portanto, na prática utiliza-se aproximações do algoritmo LRU (SILBERSHATZ; GALVIN; GAGNE, 2000).

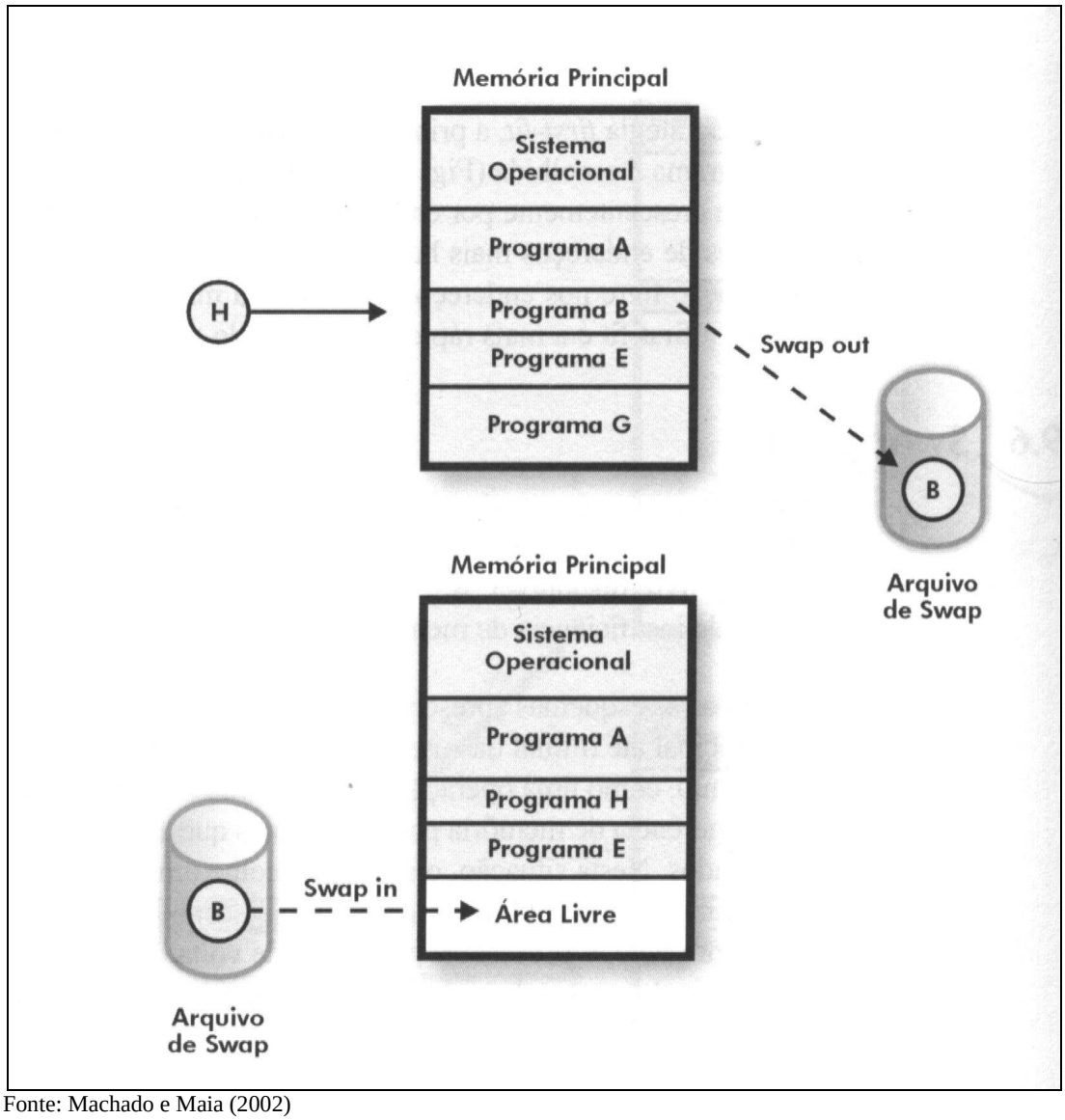
c) *not used recently* (NUR) é uma das aproximações do algoritmo LRU que é utilizado na prática por diversos sistemas operacionais. No algoritmo NUR as páginas não usadas recentemente são selecionadas para expulsão da memória. Para sua implementação são utilizados dois bits que são associados às páginas ativas. Se bit de referência estiver 0, a página não foi referenciada, mais se estiver 1 a página foi referenciada. Já no bit de modificação se estiver 0 a página não foi modificada, mais se estiver 1 a página foi modificada.A estratégia do algoritmo consiste na inicialização de todos os bits com 0. À medida que as páginas vão sendo acessadas ou modificadas, os bits correspondentes vão sendo alterados, assumindo o valor 1. Àquelas páginas com bits de referência e modificação iguais a 0 serão as candidatas a remoção da memória. Observe-se que eventualmente todos os bits podem ficar

com valores iguais a 1. Por isso, periodicamente é executada uma operação que reinicializa todos os bits de modificação (TANENBAUM, 2003).

3.3.7 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 executados e não existe memória real suficiente, o sistema seleciona um ou mais processos que deverão sair da memória para ceder espaço aos novos processos (MAIA, 2001).

Segundo Maia (2001) 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 (figura 9).



Fonte: Machado e Maia (2002)

Figura 9 – Swapping.

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* (MAIA, 2001).

Swapping impõe aos programadores um grande custo em termos de tempo de execução. Copiar todo o processo da memória para disco e mais tarde de volta para memória é uma operação demorada. É necessário deixar o processo em tempo razoável no disco para justificar tal operação (OLIVEIRA; CARISSIMI; TOSCANI, 2000, p. 103).

3.3.8 THRASHING

Thrashing pode ser definido como sendo a excessiva transferência de páginas entre a memória principal e a memória secundária podendo ocorrer em dois níveis: em nível do próprio processo e em nível do sistema (MACHADO; MAIA, 2002).

Sob a ótica de um 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 mesmo, pequeno demais para acomodar as páginas constantemente referenciadas por ele (MAIA, 2001).

Sob a ótica 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 (MAIA, 2001).

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 (MACHADO; MAIA, 2002).

4 DESENVOLVIMENTO DO TRABALHO

Neste capítulo são apresentadas a especificação e a descrição da implementação do modelo proposto neste trabalho.

4.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O simulador é um software educacional voltado ao suporte do ensino de conceitos ligados a mecanismos de gerência de memória em sistemas operacionais.

O simulador deixa disponível ao usuário algumas atividades que antes eram quase impossíveis de visualizar. Com base nas notas de aula do Prof. Mauro Mattos (MATTOS, 2003) foram determinados os objetivos principais do projeto. Baseado nesse fator foram definidos os tipos de gerência que serão manipuladas, levando-se em consideração a complexidade de certas atividades.

Visando facilitar as atividades de alunos e professores, foi criado somente um usuário, este terá como dados de entrada os processos pré-cadastrados, podendo inserir ou retirar os processos da memória principal, e como saída obter a demonstração do método visualmente ou em um *log* gerado na sua execução.

Deste modo é possível, através do simulador, verificar o funcionamento da gerência de memória real e virtual definindo quais processos serão carregados ou retirados da memória principal em um processo manual ou automático, sendo possível escolher o algoritmo de alocação.

Para fazer a especificação deste protótipo foram utilizados recursos de Unified Modeling Language (UML), usando como ferramenta o *Rational Rose*.

4.2 ESPECIFICAÇÃO

Esta parte do trabalho, apresenta a especificação do simulador de Gerência de Memória, através do Diagrama de Classes, Diagramas de Casos de Uso.

No desenvolvimento da aplicação é utilizado o ambiente de desenvolvimento Delphi 5. Para especificação do protótipo utiliza-se a ferramenta *SmartDraw e Rational Rose*, onde são construídos os fluxos da lógicas das gerências de memória e a especificação.

4.2.1 CASOS DE USO

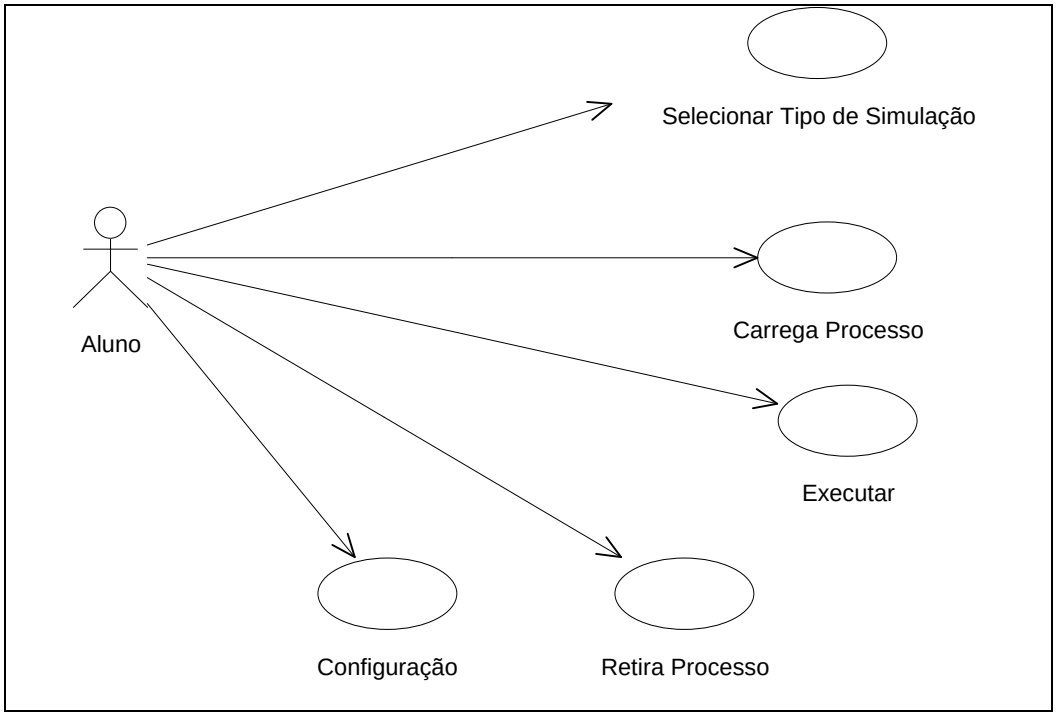


Figura 10 – Casos de Uso do Simulador.

O protótipo possui 5 casos de usos:

a) selecionar tipo de simulação: responsável por escolher qual será o tipo de gerência que será simulado;
b) carrega processo: responsável por carregar o processo para a memória principal, verificando o tipo de estratégia de alocação e a memória principal que será carregado;
c) retira processo: responsável por retirar o processo da memória principal na qual ele foi carregado anteriormente;
d) executar: responsável por executar os processos carregados na memória, verificando se a página que encontra a instrução esteja carregada na memória ou disco, caso esteja no disco transferir para memória principal;
e) configuração: responsável por fazer algumas configurações para funcionamento do simulador.

O sistema implementa 3 técnicas de gerência: memória real, memória real com partição fixa e memória virtual.

4.2.2 DIAGRAMA DE CLASSES

As classes utilizadas no protótipo são:

a) TSimulador: classe do protótipo geral do protótipo, utilizadas para todos os tipos de gerência de memória;

b) TclassReal: classe da gerência de memória real e gerência de memória real com partição fixa, contem as rotinas de carregamento e retirada do processo da memória principal;

c) TclassVirtual: classe da gerência de memória virtual , contem a rotina de execução dos processos;

d) TclassAloc: contém rotina de alocação de memória, nela retorna a posição inicial que será inserido o processo conforme estratégia de alocação.

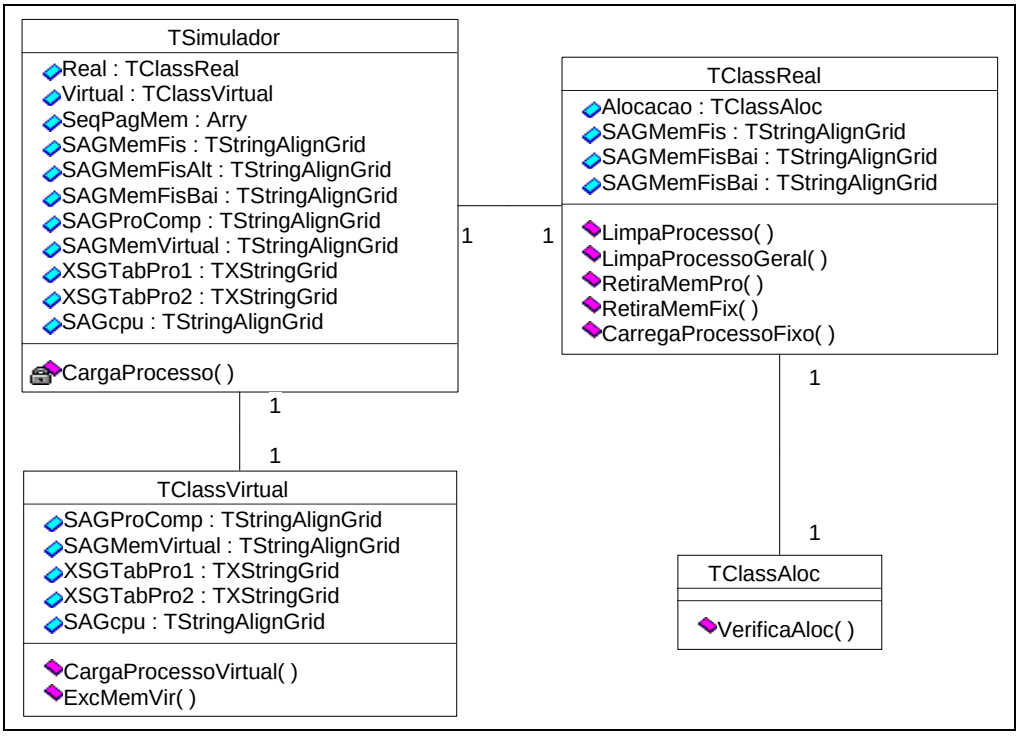


Figura 11 – Diagrama de Classes do Simulador.

4.3 IMPLEMENTAÇÃO

Nesta seção descreve-se a implementação propriamente dita do protótipo. É descrita a estrutura lógica de cada método de gerência de memória, ferramentas utilizadas e a operacionalidade da implementação. Para finalizar, são comentados os resultados obtidos.

4.3.1 SIMULAÇÃO DE GERÊNCIA DE MEMÓRIA REAL

A figura 12 apresenta, a página principal do simulador, onde é implementado o primeiro método de gerenciamento de memória conhecido como memória real. Neste método todo o processo tem que ser carregado para a memória física para a sua execução.

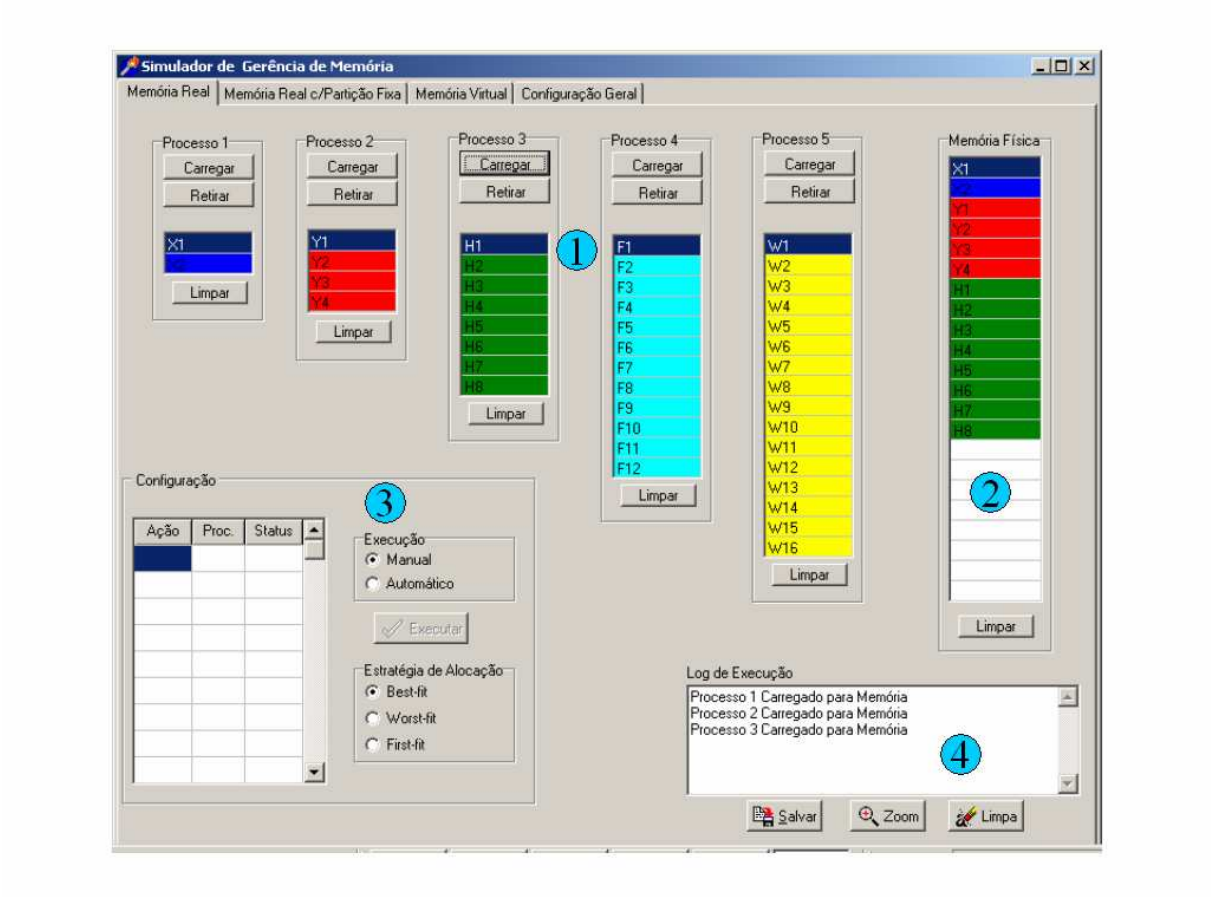


Figura 12 - Tela Gerência de Memória Real.

A tela é dividida em 4 regiões, quais sejam:

a) área 1 representa os processo ou programas que poderão ser carregados para a memória ;

b) área 2 representa a memória física do simulador;

c) área 3 representa as configurações possíveis para a simulação da gerência de memória real;
d) área 4 guarda a seqüência de execução.

Os programas são representados em cores diferentes para facilitar a avaliação do layout da memória após a carga. Os programas são implementados com o componente *TStringGrig* onde cada linha se refere a uma página do processo. Cada página está descrita com uma nomenclatura diferente que representa um conjunto de instruções à ser executada pelo processador (os dados a serem acessados).

Na gerência de memória real há cinco processos com tamanho e cor diferentes, para possibilitar uma melhor visualização diferenciada na demonstração do funcionamento.

A figura 13 mostra o layout de um programa executável em disco o qual possui 12 “páginas” de código (mais dados) a ser carregados para a memória.

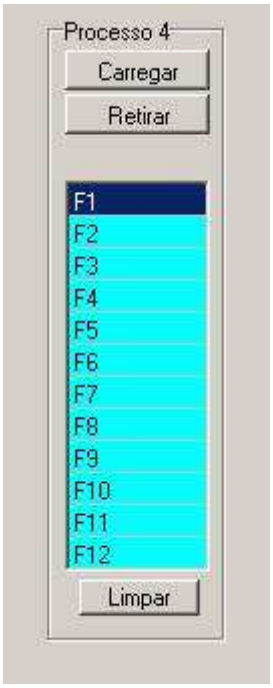


Figura 13 - Processo.

Segundo Maia (2001) como o processador somente executa instruções localizadas na memória física, o sistema operacional deve sempre transferir o processo da memória secundário, para a memória física antes de executar. A figura 14 representa a memória física, a qual é subdividida em 22 páginas, ou seja, a capacidade de memória do simulador é de 22 páginas de 1 bytes. A memória foi implementada com o componente *TStringAlignGrid*.

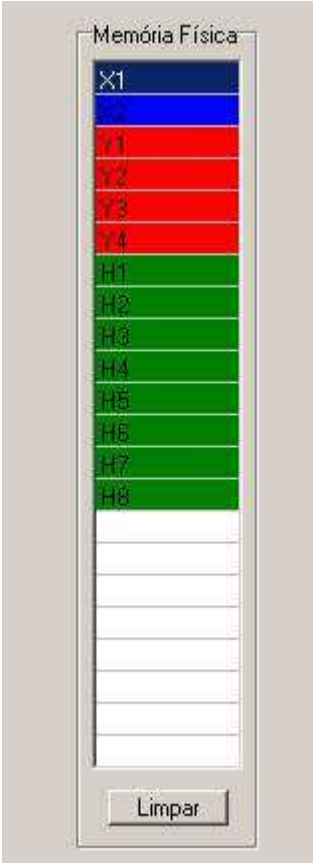


Figura 14 - Memória Física.

A figura 15 apresenta o painel de configuração que permite determinar a seqüência a ser simulada. Uma das configurações possíveis é poder escolher o tipo de execução do simulador. Pode-se optar pela execução manual ou automática.

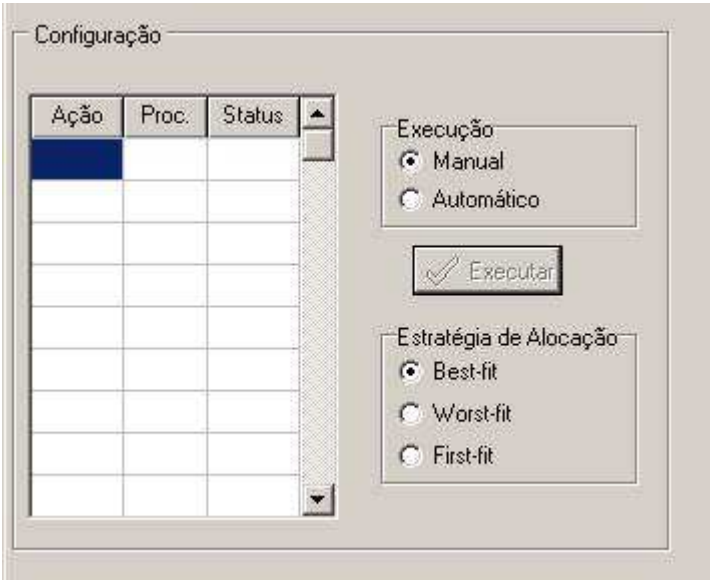


Figura 15 - Configurações da Gerência de Memória Real.

Na execução manual, fica sob responsabilidade do usuário estabelecer a seqüência de carga para a memória física e retira-los, através dos botões “Carregar” e “Retirar” existente em cada um dos processos (figura 13).

No simulador de gerência de memória real podem ser escolhidos dois tipos de execução, manual ou automático. A figura 16 apresenta o diagrama de atividades da lógica da execução manual onde é possível carregar e retirar os programas da memória separadamente.

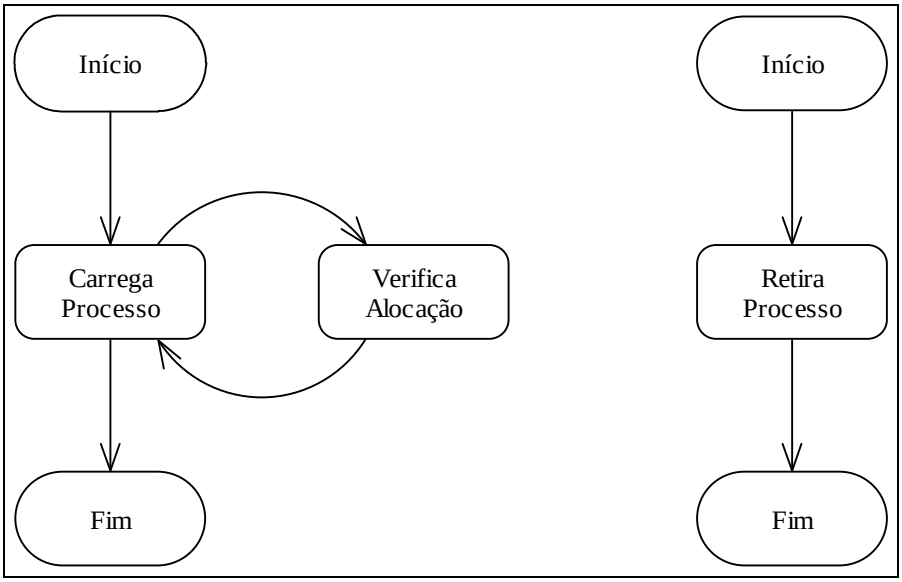


Figura 16- Lógica do processo manual.

O botão “Carregar” tem a função de transferir o processo da memória secundária para a memória física, para que o processador execute este processo. O botão “Retirar” tem função de retirar o processo da memória física após ser executado pelo processador. Ver seção 4.4.1, um exemplo de carga manual.

Na execução automática deve ser cadastrada uma seqüência de ações que o simulador irá executar. Cada coluna tem uma função importante. Na primeira coluna deve ser selecionada, a ação (retirar ou carregar) que o processo irá sofrer. Na segunda coluna será escolhido o processo que sofrerá a ação que foi selecionada na coluna anterior e na terceira coluna será mostrado o status do processo no momento da execução automática (figura 17).

Figura 17 - Cadastro da Execução Automática.

Ao pressionar o botão “Executar”, (figura 17) o usuário visualizará a execução automática das ações cadastradas na tabela de execução automática. Este botão somente é habilitado quando o tipo de execução for automático.

A figura 18 está apresentado o diagrama de atividades da lógica de funcionamento da execução automática da gerência de memória real, onde será cadastrada uma tabela de seqüência de execução definido quais processo serão carregados ou retirados da memória real.

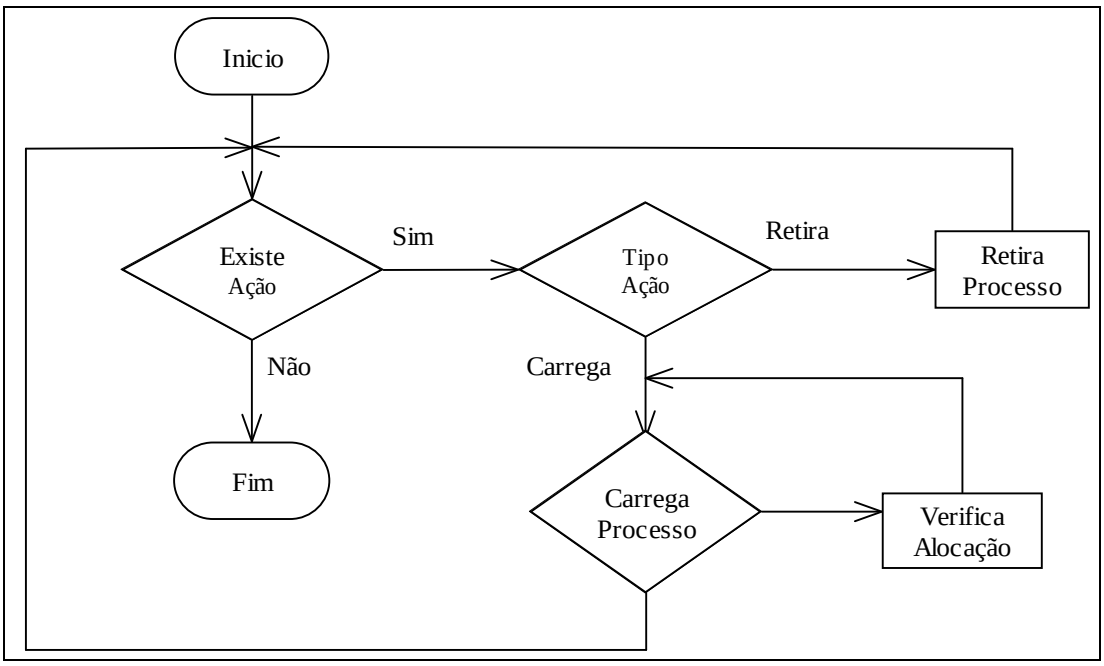


Figura 18 - Lógica do processo automático.

Na implementação da carga dos processos na gerência de memória real há uma função responsável por retornar o endereço inicial da alocação do processo que está sendo carregado para a memória principal. Nesta função utiliza-se uma estrutura para auxiliar no retorno da posição inicial (Quadro 1). Esta estrutura guarda a posição inicial da lacuna (espaço livre de memória) e a posição final, podendo ser calculada a quantidade de páginas livres e o tamanho da lacuna que irá surgir com a carga do processo que está sendo carregado. Com estes resultados pode-se ordenar a estrutura conforme a estratégia de alocação que foi escolhida (apêndice A).

```
type
   TAloc = record
      Inicio : Integer;  \\ Posição de inicio da lacuna
      Fim    : Integer; \\ Posição de fim da lacuna
      Qtd    : Integer; \\ Quantidade de paginas
      Resto  : Integer; \\ Qtd. Pág. livres com processo
   end;
```

Quadro 1 – Declaração da estrutura de alocação.

Para os dois tipos de execução (manual ou automático) pode ser escolhida uma estratégia de Alocação: o método o ***First-fit*** aloca o processo na primeira partição que encontrar com tamanho suficiente para atender ao processo; ***Best-fit*** procura a partição de tamanho mais próximo ao tamanho do processo e o método ***Worst-fit*** procura a partição que tenha maior tamanho.

Para permitir uma análise posterior é gravado um *log* de execução, onde são descritos os processos que foram carregados e retirados da memória física na sua seqüência de execução, além de informar quais os processos não foram carregados por falta de área livre de memória. Este *log* pode ser visualizado no próprio simulador como visto na figura 19, podendo ser salvo como arquivo txt para futura análise.

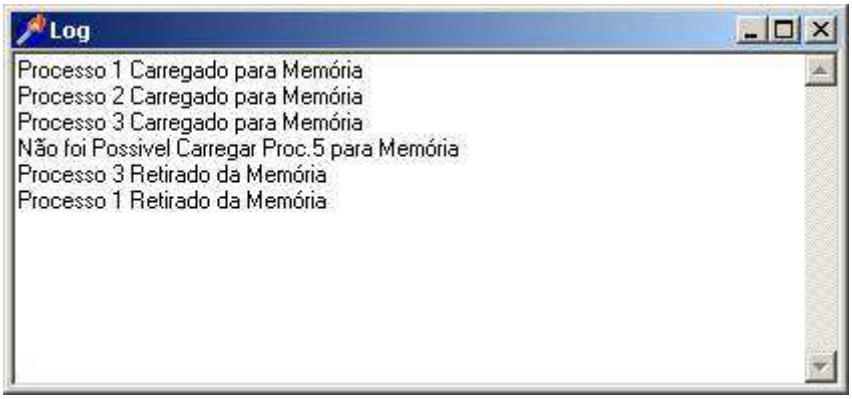


Figura 19 - Log de Execução.

A seção 4.4.2 apresenta um exemplo de execução.

4.3.2 SIMULAÇÃO DE GERÊNCIA DE MEMÓRIA REAL COM PARTIÇÃO FIXA

A figura 20 apresenta o modelo de gerência de memória real com partição fixa. Neste simulador o usuário poderá configurar em qual partição que processo será carregado: memória baixa ou alta. As outras configurações e funcionalidades do simulador de gerência de memória real com partição fixa são idênticas ao processo de gerência de memória real descrito no item 4.3.1

A diferença em relação ao modelo anterior reside no fato que a memória real está dividida em 2 partições: uma com 12 páginas e outra com 10 páginas. A funcionalidade é semelhante em cada partição. Um exemplo de uso é apresentado na seção 4.4.3.

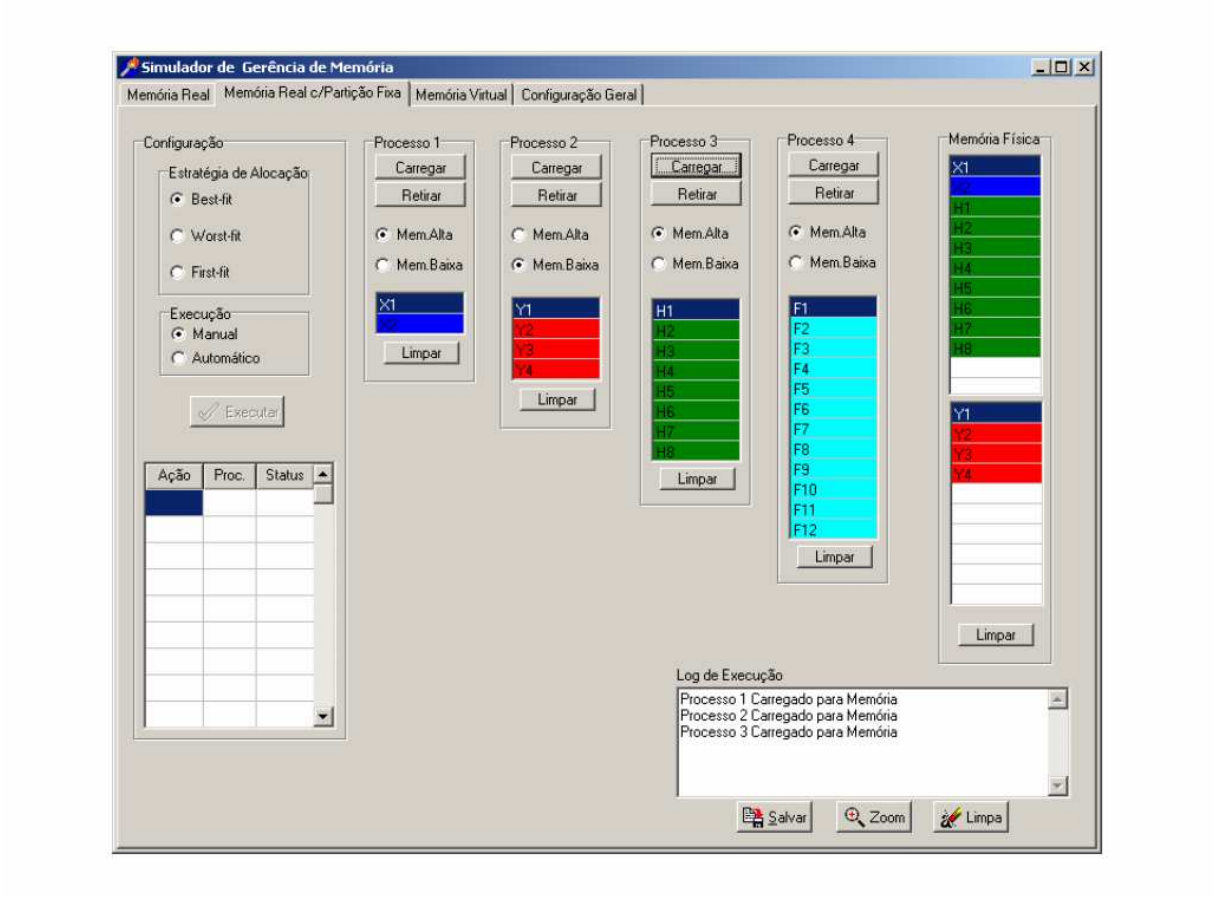


Figura 20 - Gerência de Memória Real com Partição Fixa.

4.3.3 GERÊNCIA DE MEMÓRIA VIRTUAL

O modelo de gerência de memória virtual é um pouco mais complexo (figura 21). Nesta estratégia de gerenciamento não é necessário que todo o processo esteja presente na memória física para que o processador possa executá-lo (seção 3.2). Assim pode-se ter vários processos não sendo executado em determinado momento. Além disso, é possível a execução de processos cujo o tamanho excede ao tamanho da memória real. Ao contrário dos modelos anteriores, o modelo de gerência de memória virtual simula a carga e execução de apenas 2 processos em função das restrições do número de informações a serem apresentadas e das restrições de tamanho de tela.

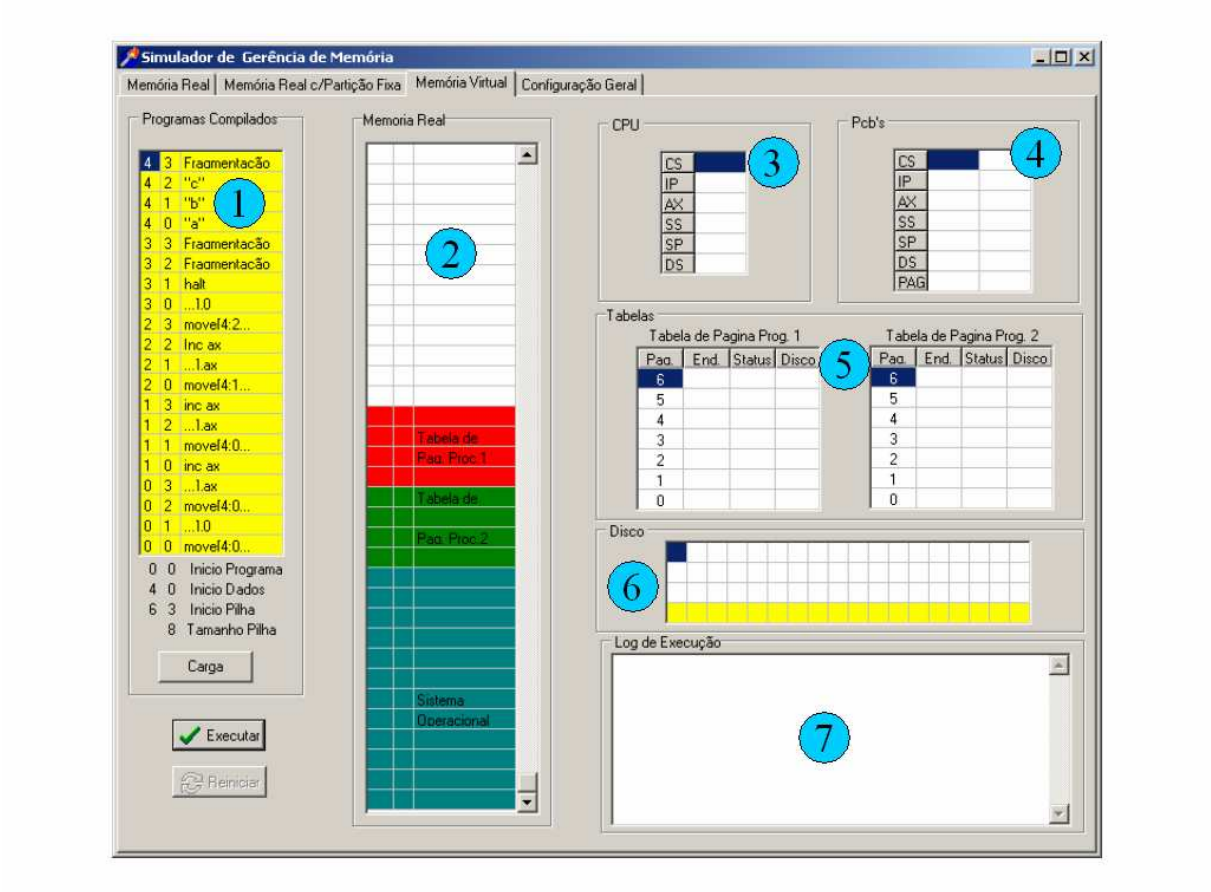


Figura 21 - Gerência de Memória Virtual.

A tela é dividida em 7 regiões, quais sejam:

a) área 1 demonstra o contendo o código do programa a ser carregado. Esta área deve ser considerada para fins de simulação sendo o programa completo, armazenado em disco. Seria equivalente aos arquivos exe do windows;

b) área 2 representa a memória física do simulador;

c) área 3 representa a cpu do sistema;

d) área 4 representa o registro de controle dos processos em execução. É utilizado para armazenar o conteúdo dos registradores da CPU quando o processo perde o processador;

e) área 5 demonstra as tabelas de páginas, representa em forma destacada, o conteúdo de tabela de páginas de cada processo que foi carregado;

f) área 6 representa o disco rígido, caracterizando inicialmente a área ocupada pelo arquivo exe;

g) área 7 representa seqüência de execução.

A figura 22 apresenta o código do programa. Este painel representa o código executável no disco (antes da carga para a memória). Pressionando-se o botão carregar, executa-se o procedimento de carga do programa para a memória real.

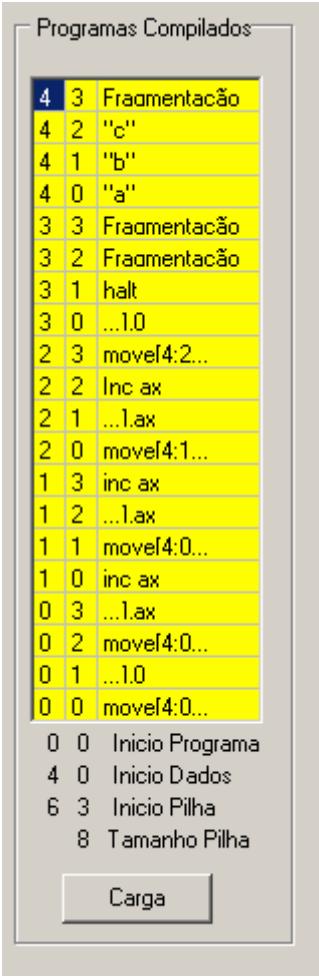


Figura 22 – Programa.

O código fonte do programa (quadro 2) que gerou esta seqüência de instruções é descrito a seguir:

```
a = 0
a = a + 1
b = b + 1
c = a + 1
```

Quadro 2 - Código Fonte.

Basicamente é uma aplicação que possui 3 variáveis a,b,e c. Inicializa-se a variável “a” com 0 e soma-se 1. Posteriormente, incrementa-se o valor de “a” agregando em “b” e da mesma forma incrementa-se o valor de “b” agregando a “c”.

O código “compilado” (quadro 3) a partir do fonte acima produziu o seguinte código “executável”, o qual é expresso em uma pseudo linguagem assembly.

```
move   a , 0
move ax , a
inc    ax
move  a , ax
inc   ax
move  b, ax
inc  ax
move c, ax
```

Quadro 3 – Código Compilado

A memória principal (figura 23) foi implementada com o componente *TStringAlignGrig.*

A memória física possui um tamanho de 18 páginas, cada página contem 4 bytes. Cada divisão representa 1 byte. O código executável representa instruções de 1 e 2 bytes, alguns alocados estrategicamente na fronteira entre 2 páginas de modo a simular a ocorrência de *page fault* como será demonstrado no estudo de caso.

A figura 23 apresenta o layout completo das páginas na memória real, além da representação da tabela de páginas de cada processo.

A figura 23 apresenta o layout da memória no contexto de simulação. Na base da memória está o código do sistema operacional. Acima se encontra as tabelas de páginas dos processos. O restante de memória disponível é paginável.

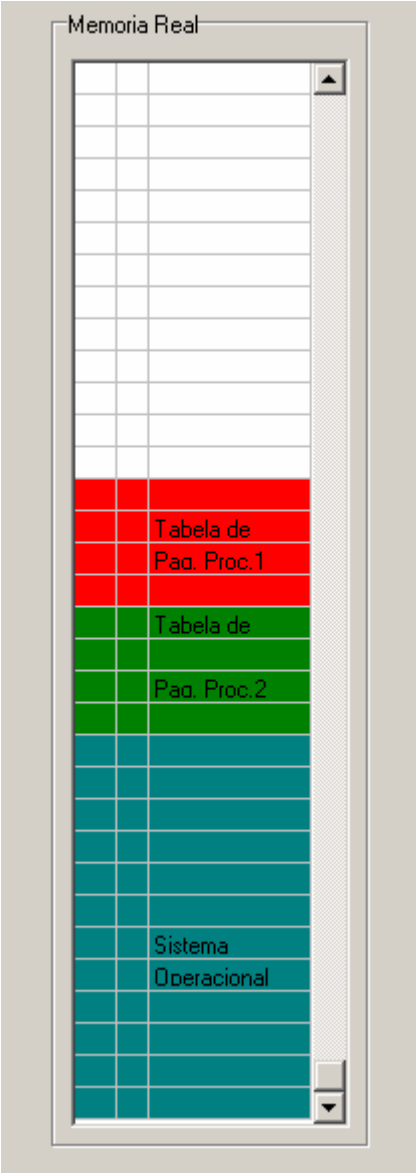


Figura 23 - Memória Principal.

O botão de carga tem a função de carregar os programas para a memória principal, automaticamente montando a tabela de página do processo (figura 24). Essa tabela informa, para cada página lógica, qual será a página física correspondente na memória real.

Segundo Maia (2001) todo o mapeamento é realizado em nível de página, através das tabelas de páginas. Cada página virtual do processo possui uma entrada na tabela de páginas, com informações de mapeamento que permitem ao sistema localizar a página real correspondente na memória principal.

Dependendo do status encontrado na tabela, o endereço da página física pode apontar para o “disco” (status “D”) ou memória física(status “M”).

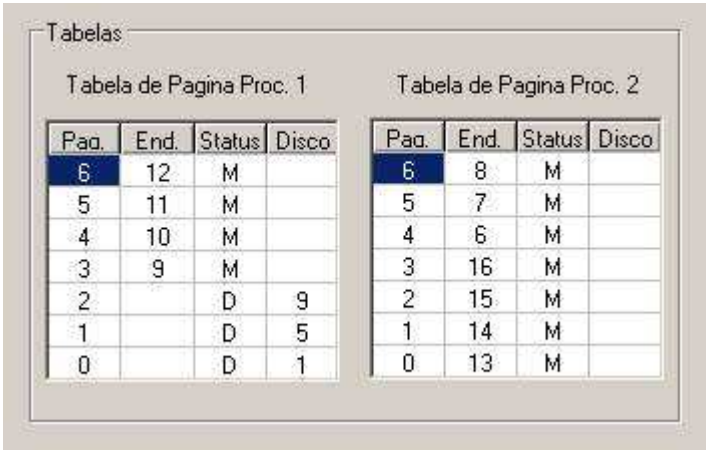


Figura 24 - Tabela de paginas.

A figura 25 representa o disco do sistema , o qual é geralmente usado como memória secundária, para armazenar as páginas que não poderão ser carregadas para a memória principal durante a carga, ou quando o processador requisita uma página que não está em memória e não há mais espaço livre na memória principal. Como método de substituição de páginas foi implementado o método *FIFO*. Como estudado na seção 3.3.6, este método irá verificar qual à página está há mais tempo na memória principal transferindo-a para o disco e transferindo para a memória principal a página solicitada.

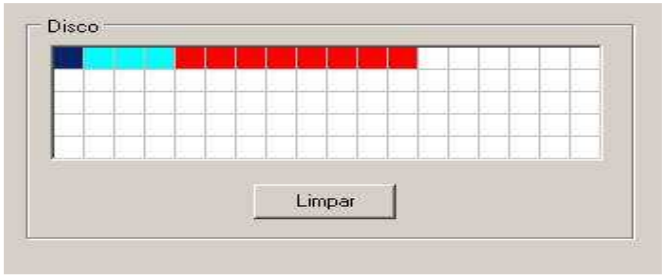


Figura 25 – Disco.

A figura 26 apresenta o diagrama de atividades na execução de um programa num contexto de simulação de gerência de memória virtual. Primeiramente o programa deve estar carregado na memória principal. Depois da carga pode-se executar as instruções. A cada instrução é verificada a tabela de páginas para verificar se a página que se encontra esta instrução está em memória ou no disco. Caso esteja no disco esta página será carregada para memória usando o método de substituição de página FIFO. Caso esteja em memória, a MMU fará a conversão do endereço virtual em real.

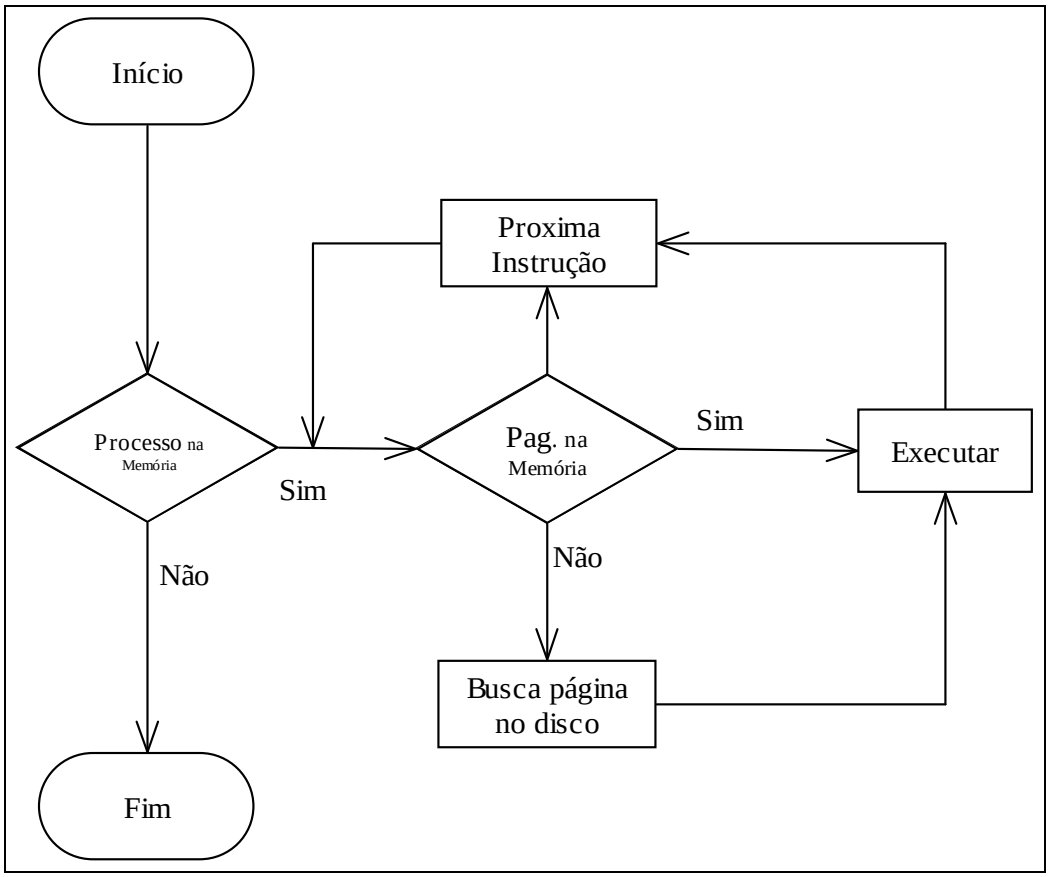


Figura 26 – Lógica da gerência de memória virtual.

No algoritmo de substituição de página implementado (FIFO) a página escolhida como vítima é sempre aquela que está a mais tempo na memória. Para implementação foi definido um *array* (Quadro 2) para guardar o número da página. Quando a página é carregada para a memória o número dela é inserido na ultima posição livre deste *array*. No momento de escolher a página vitima que será retirada da memória , basta verificar o número da página que está no início do *array*.

```
SeqPagMem : Array [0..30] of Integer;
```

Quadro 2 – Declaração do array.

4.3.4 CONFIGURAÇÃO

O Módulo de configuração (figura 27) serve para alterar a identificação de cada página do processo sendo simulado pelos modelos de memória real e memória real com partição fixa. Além da identificação é possível configurar o tempo de execução irá ter entre uma instrução e outra na execução da gerência de memória virtual.

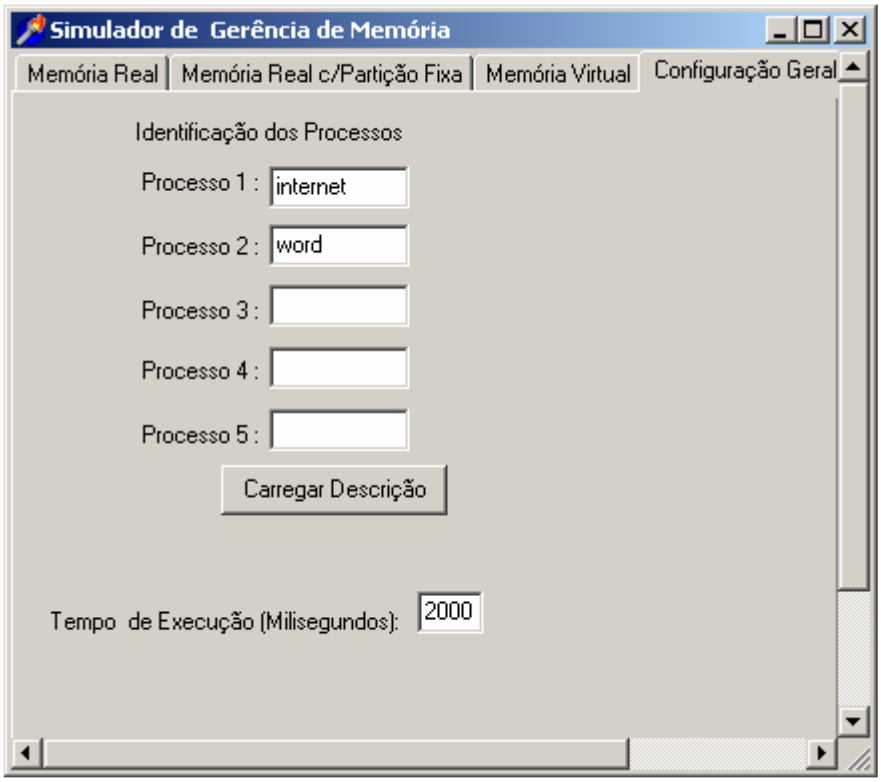


Figura 27 – Configuração.

4.4 OPERACIONALIDADE DA IMPLEMENTAÇÃO

Para um melhor entendimento sobre a execução do simulador será discutido um estudo de caso para cada uma das simulações.

4.4.1 SIMULADOR DE GERÊNCIA DE MEMÓRIA REAL COM CARGA MANUAL

Nesta seção é apresentado um exemplo de utilização do sistema para simular o comportamento do sistema de gerenciamento de memória real com carga manual. Para tanto será considerado o seguinte cenário:

a) modo de alocação FIFO;
b) 5 programas a serem carregados;
c) a memória física possui 22 paginas;
d) o tamanho total dos programas supera o tamanho da memória.

A simulação começa com o usuário fazendo a carga para a memória do processo 2 (figura 28). Deve-se observar que o programa foi carregado nas 4 primeiras páginas da memória, ficando sobrando um bloco de 18 páginas disponíveis.

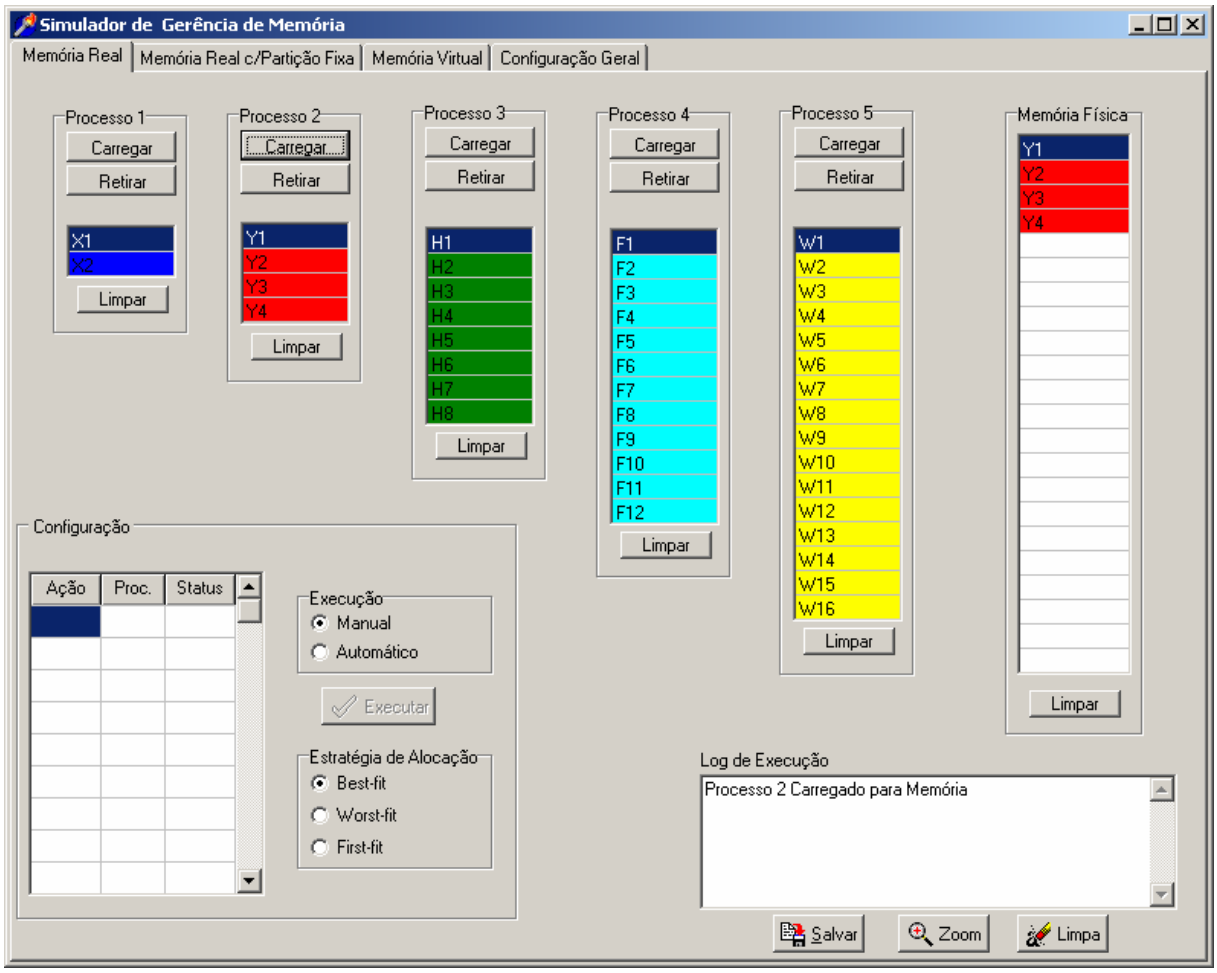


Figura 28 - Carrega processo2.

Ao ser carregado, o processo 4, foi alocado após o processo 2 (figura 29), ficando disponíveis um bloco 6 páginas contiguas.

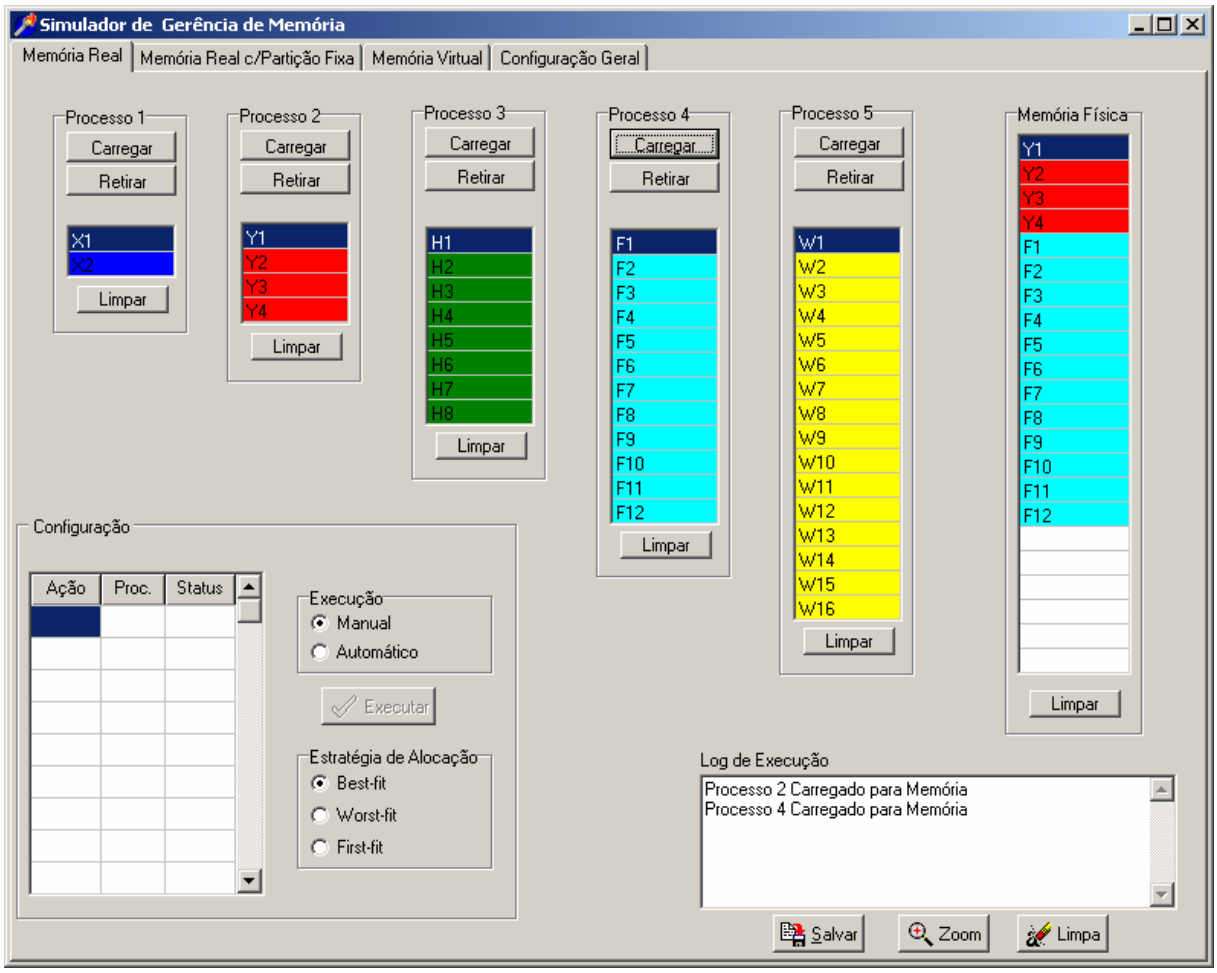


Figura 29 - Carrega processo 4.

Ao ser carregado o processo 3, percebe-se que não foi possível carregar para a memória, como visto na figura 30. O processo 3 tem 8 páginas não havendo bloco de memória livre suficiente para a sua alocação. Este processo fica aguardando até que um dos dois processos que estão executando finalize e libere a memória.

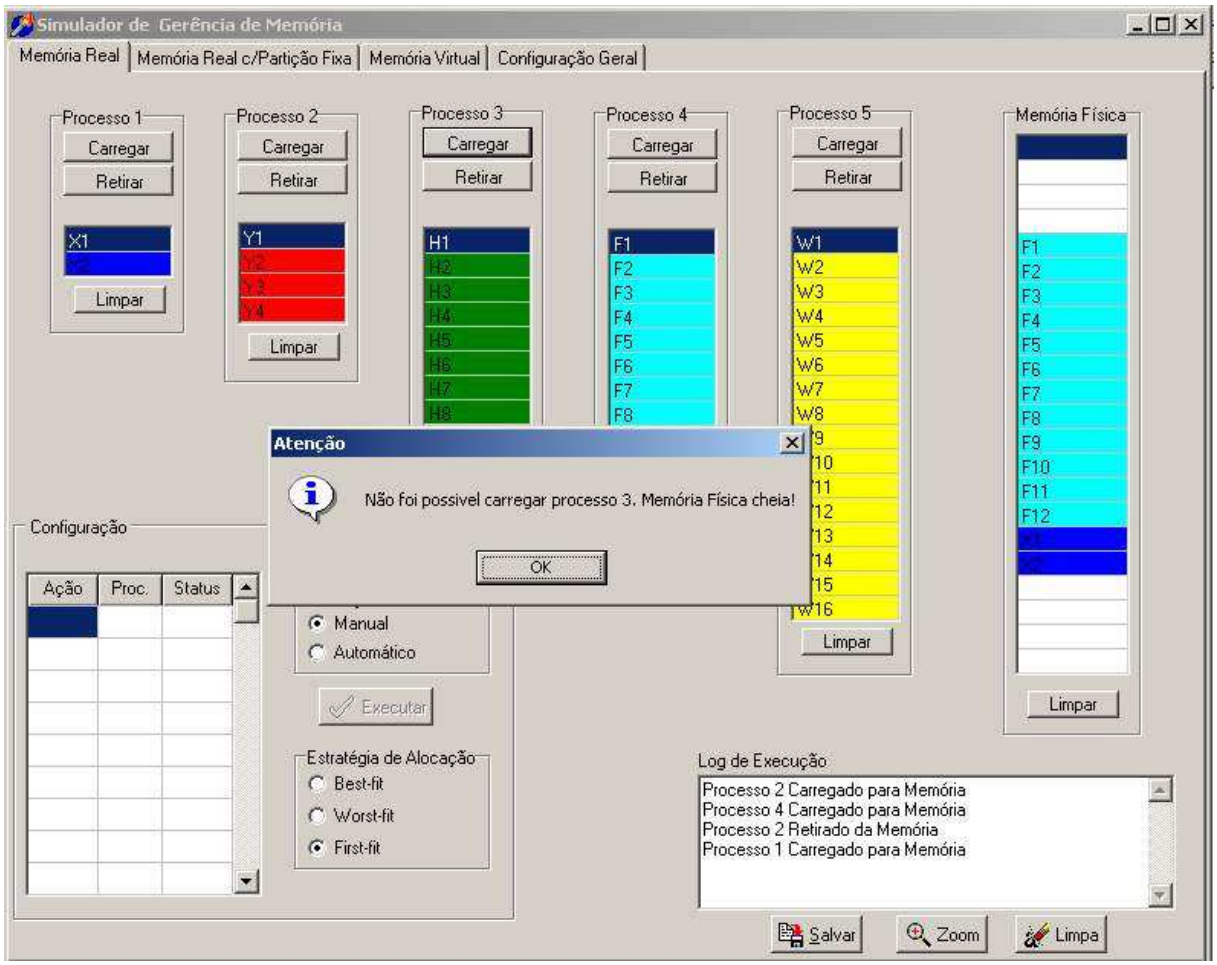


Figura 30 – Memória cheia.

A figura 31 apresenta a situação em que o programa 4 finalizou sua execução, deixando 4 páginas disponíveis no primeiro bloco e 6 páginas disponíveis depois no segundo bloco. Até este momento não importando o tipo de estratégia de alocação de memória a alocação do processo seria o mesma, pois somente existe um bloco de páginas livres na memória.

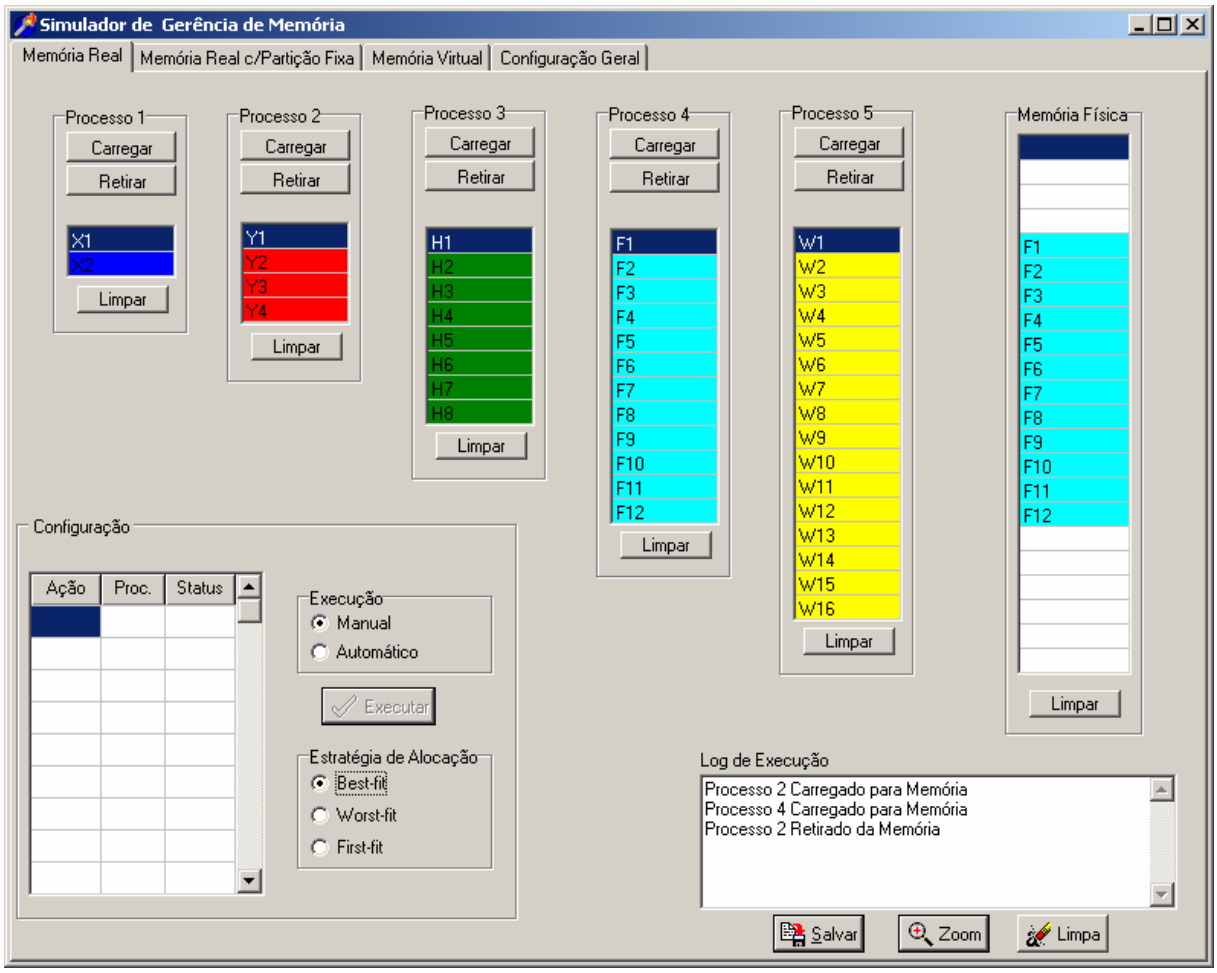


Figura 31 - Retirar processo.

O usuário seleciona o processo 2 para ser carregado para a memória. Agora pode-se perceber o funcionamento da estratégia de alocação.

Se a estratégia de alocação selecionada for *best-fit*, o processo é inserido no primeiro bloco existente livre (figura 32), pois como foi estudado anteriormente o método *best-fit* irá verificar a menor partição livre, ou seja, aquela que o programa deixar menor espaço sem utilização.

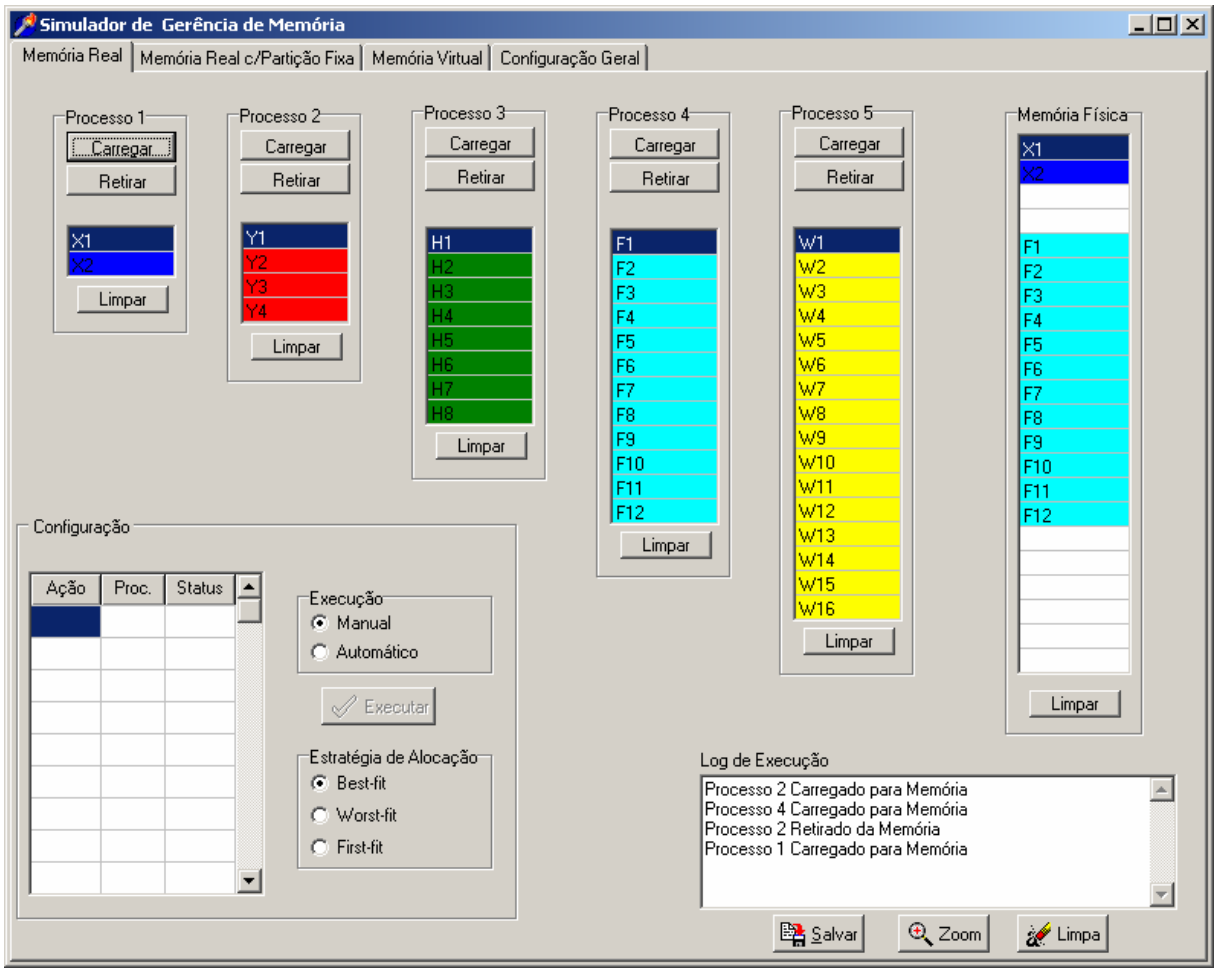


Figura 32 - Método Best-fit.

Se a estratégia de alocação selecionada fosse *worst-fit*, o processo 2 seria inserido no segundo bloco livre existente (figura 33), pois a pior partição é escolhida, ou seja, aquela em que o processo deixar o maior espaço sem utilização.

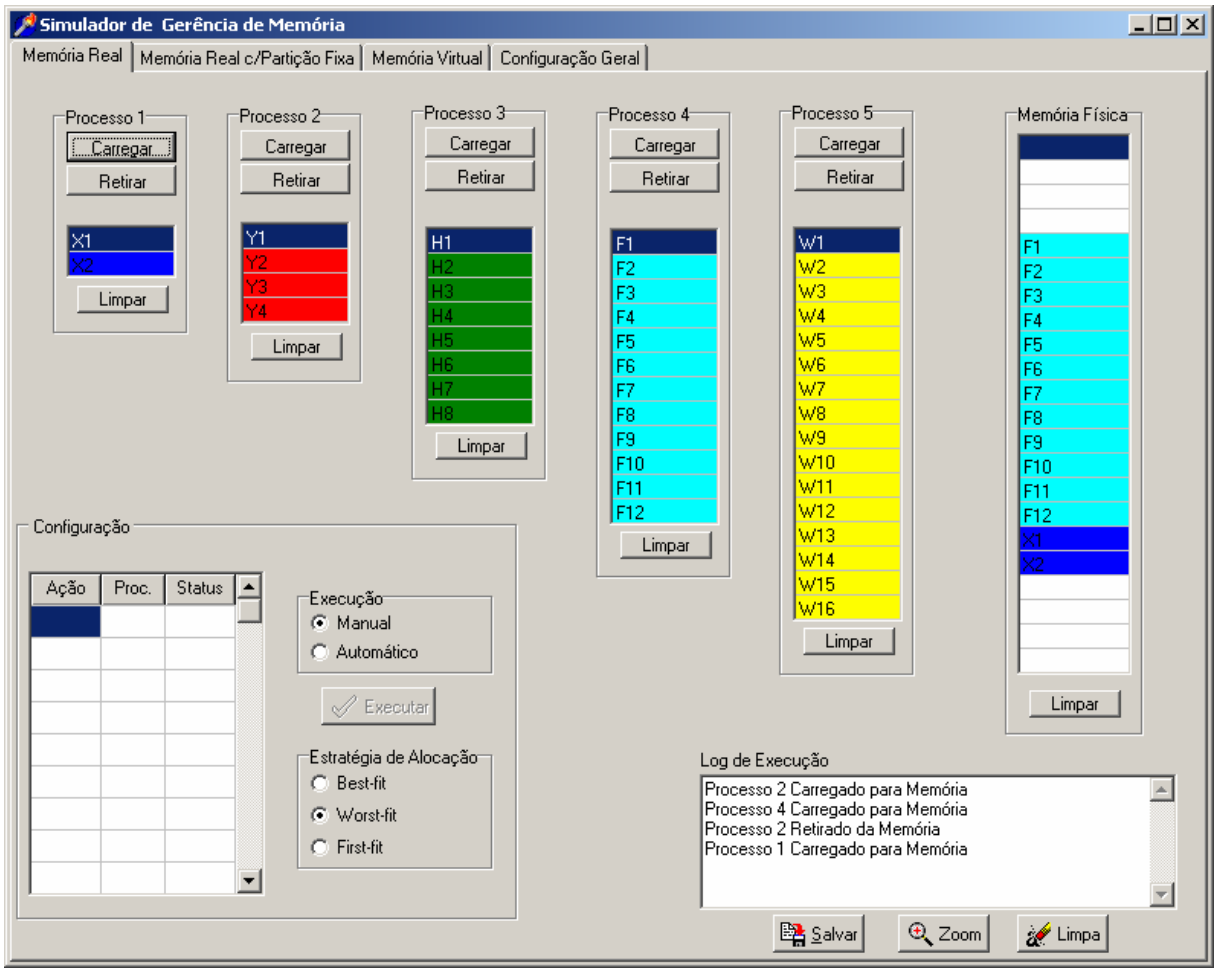


Figura 33- Método Wost-fit.

Se a estratégia de alocação selecionada fosse *first-fit*, o processo 2 seria inserido no primeiro bloco com 4 páginas livre (figura 34), pois este método pega a primeira partição livre de tamanho suficiente para carregar o processo escolhido.

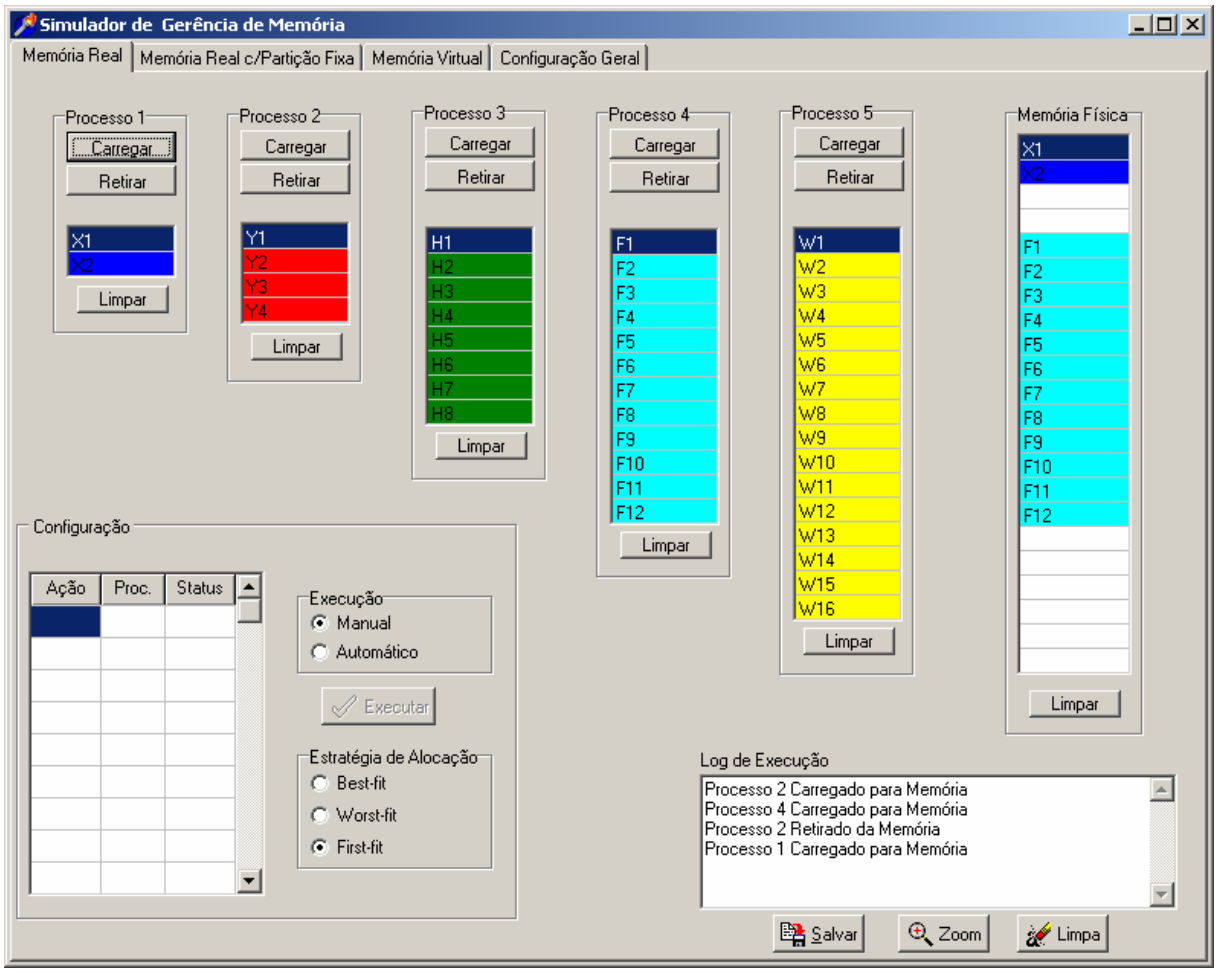


Figura 34- Método First-fit.

4.4.2 SIMULADOR DE GERÊNCIA DE MEMÓRIA REAL COM CARGA AUTOMÁTICA

Nesta seção é apresentado um exemplo de utilização do sistema para simular o comportamento do sistema de gerenciamento de memória real com carga automática. Para tanto será considerado o mesmo cenário do tópico anterior, pois a única diferença é que nesta simulação o usuário não precisa carregar cada processo manualmente pelos botões de carregar, mais sim cadastra uma tabela de execução.

Nesta situação o usuário primeiramente tem que escolher o tipo de execução automática. Após escolha, o simulador ira liberar a tabela existente no painel de configuração (figura 35) para cadastramento.

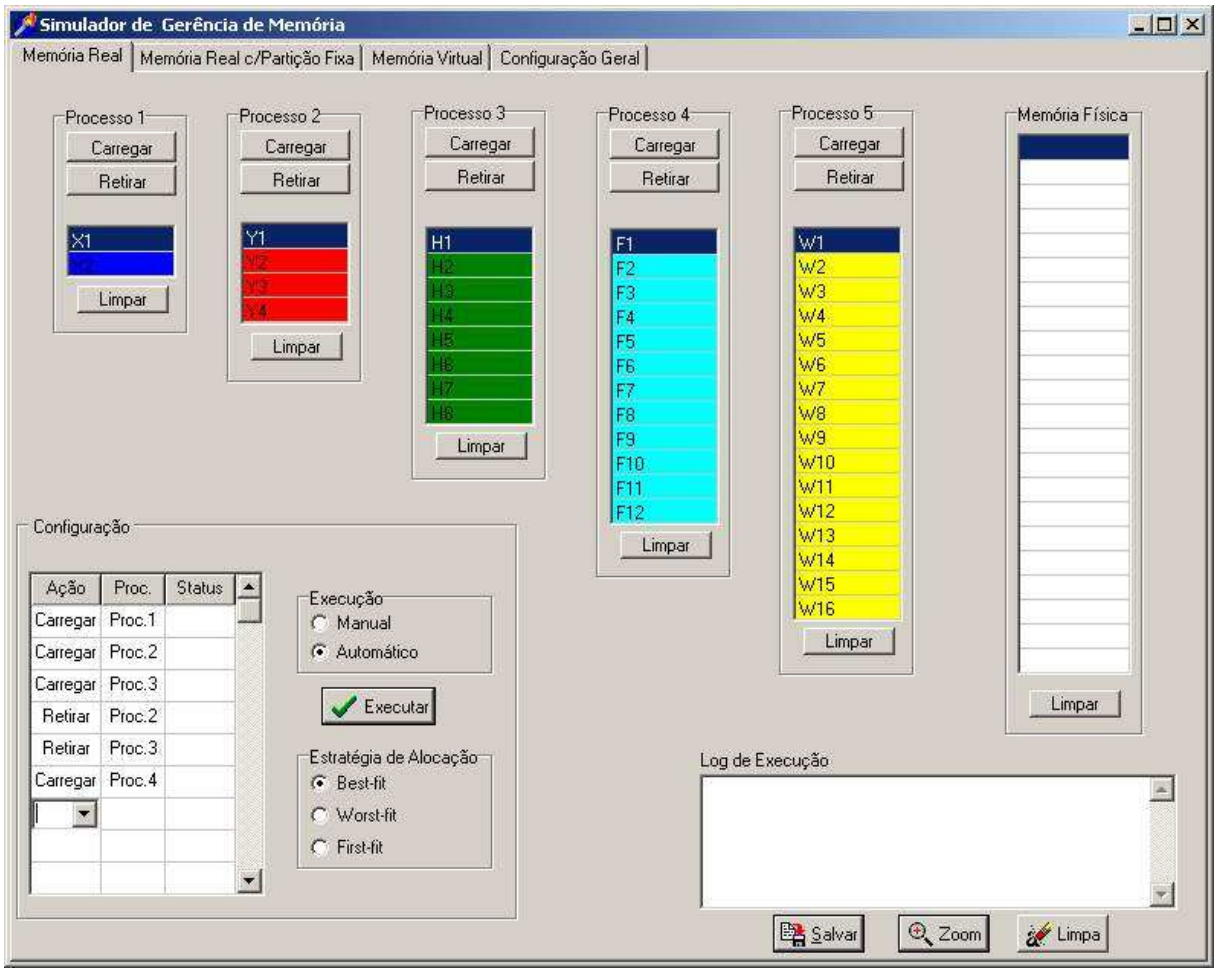


Figura 35 - Cadastro configuração.

Na figura 35 pode-se verificar que foi cadastrada uma seqüência de carga de processos com uma ação específica para cada processo. Após cadastrada a seqüência o usuário pressionará o botão executar e a simulação começará a demonstrar automaticamente a alocação dos processos na memória real.

Percebe-se na figura 36 que as três primeiras ações cadastradas foram executas com sucesso (status do processo informando “OK”). Foram carregados para a memória os processo 1,2 e 3, deixando disponível um bloco de 8 páginas livres na memória. Na simulação automática a estratégia de alocação deve ser escolhida antes de iniciar a execução.

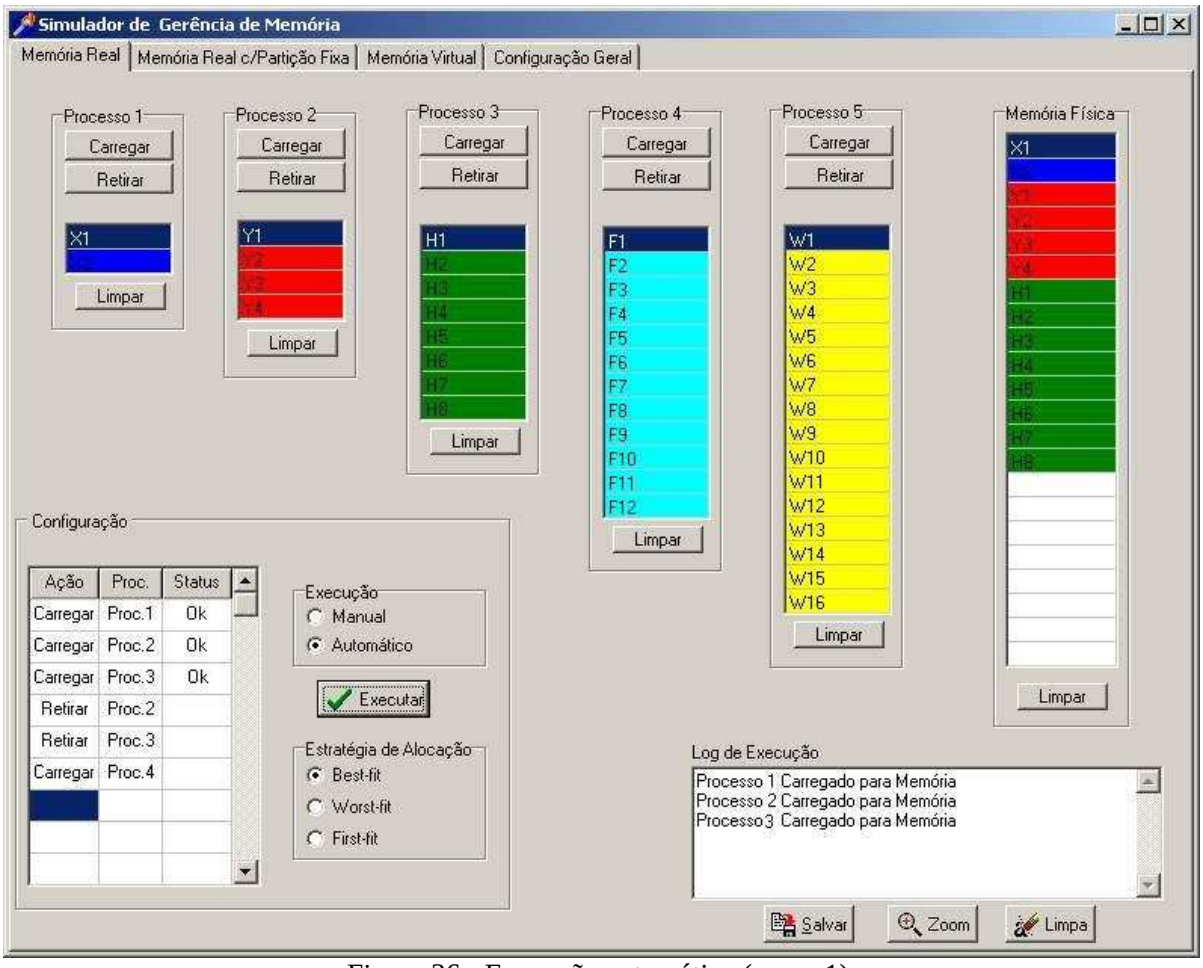


Figura 36 - Execução automática (passo 1).

A figura 37 demonstra mais duas ações executas. O processo 2 é seqüencialmente o processo 3 finalizaram sua execução, deixando disponível um bloco livre de 20 páginas para alocação do próximo processo selecionado na tabela. Os métodos de alocação terão a mesma funcionalidade descrito no item anterior para cada tipo de estratégia selecionada, verificando em qual bloco livre na memória o processo será alocado.

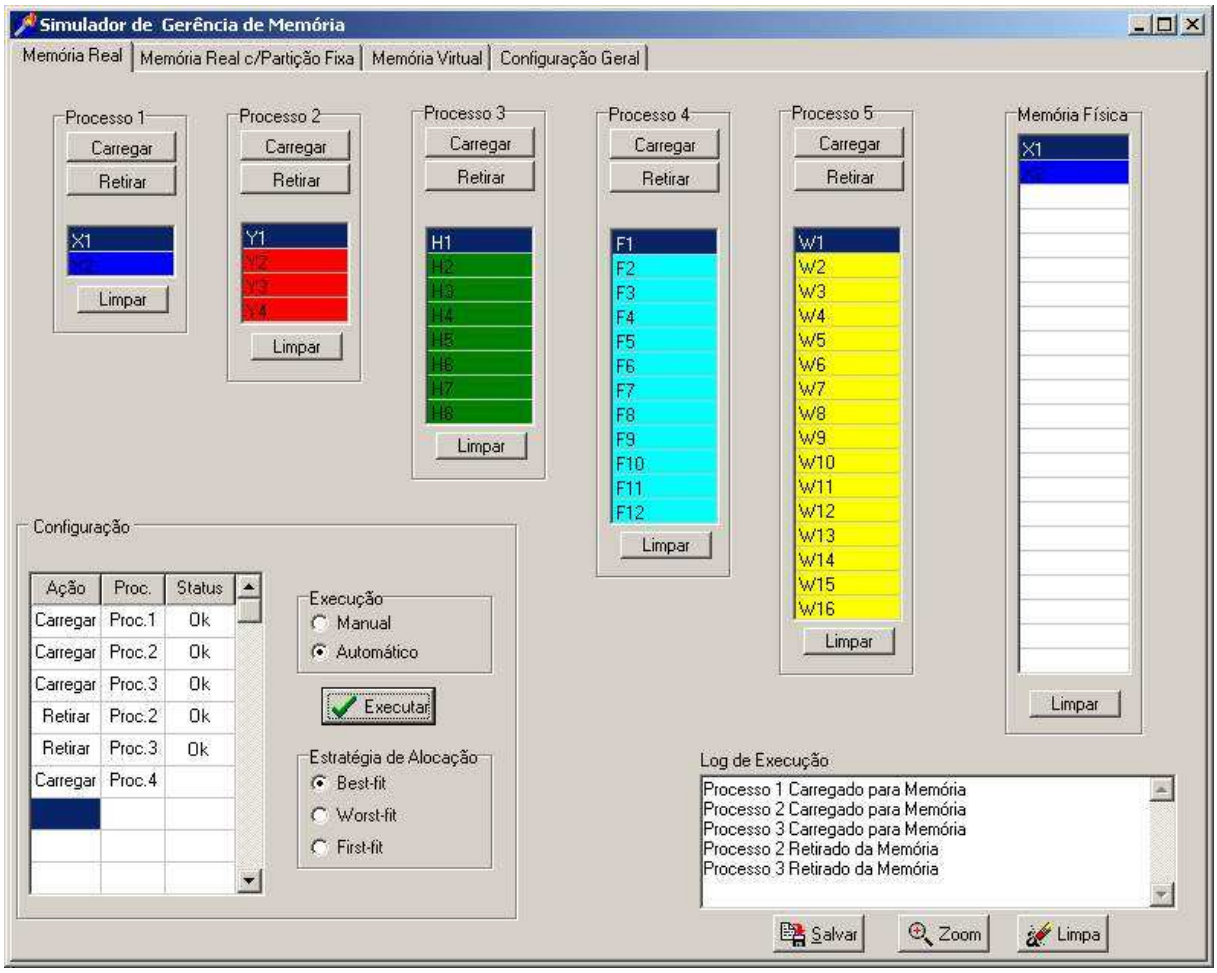


Figura 37 - Execução automática (passo 2).

Na figura 38 observa-se que o processo 4 foi carregado no bloco de páginas livres na memória real. Este processo utilizou 12 páginas de memória livre para a sua alocação, deixando disponíveis 8 páginas livres.

Um histórico de execução é gerado para posterior análise pelo usuário.

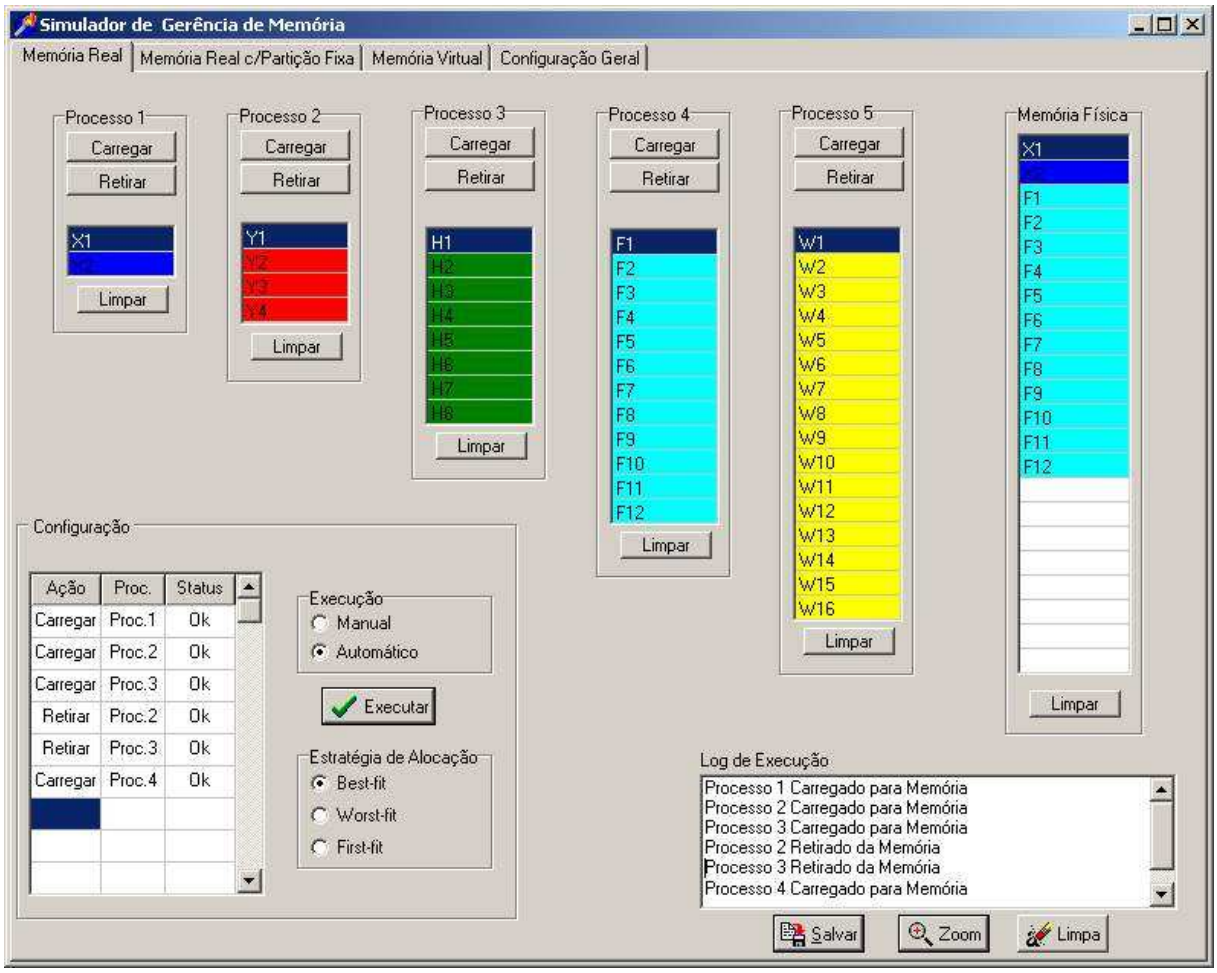


Figura 38 - Execução automática (passo 3).

4.4.3 GERÊNCIA DE MEMÓRIA REAL COM PARTIÇÃO FIXA

Nesta seção é apresentado um exemplo de utilização do sistema para simular o comportamento do sistema de gerenciamento de memória real com partição fixa. Para tanto será considerado o seguinte cenário:

a) modo de alocação FIFO;

b) 5 processos a serem carregados;

c) a memória física esta dividida em duas partições: uma possuindo 12 páginas (chamada de memória alta), outra com 10 páginas (chamada de memora baixa);

d) o tamanho da soma de alguns processos supera o tamanho de cada partição de memória.

A simulação começa com o usuário definindo qual partição o processo será carregado (figura 39). Após definição poderá iniciar a carga dos processo para a memória.

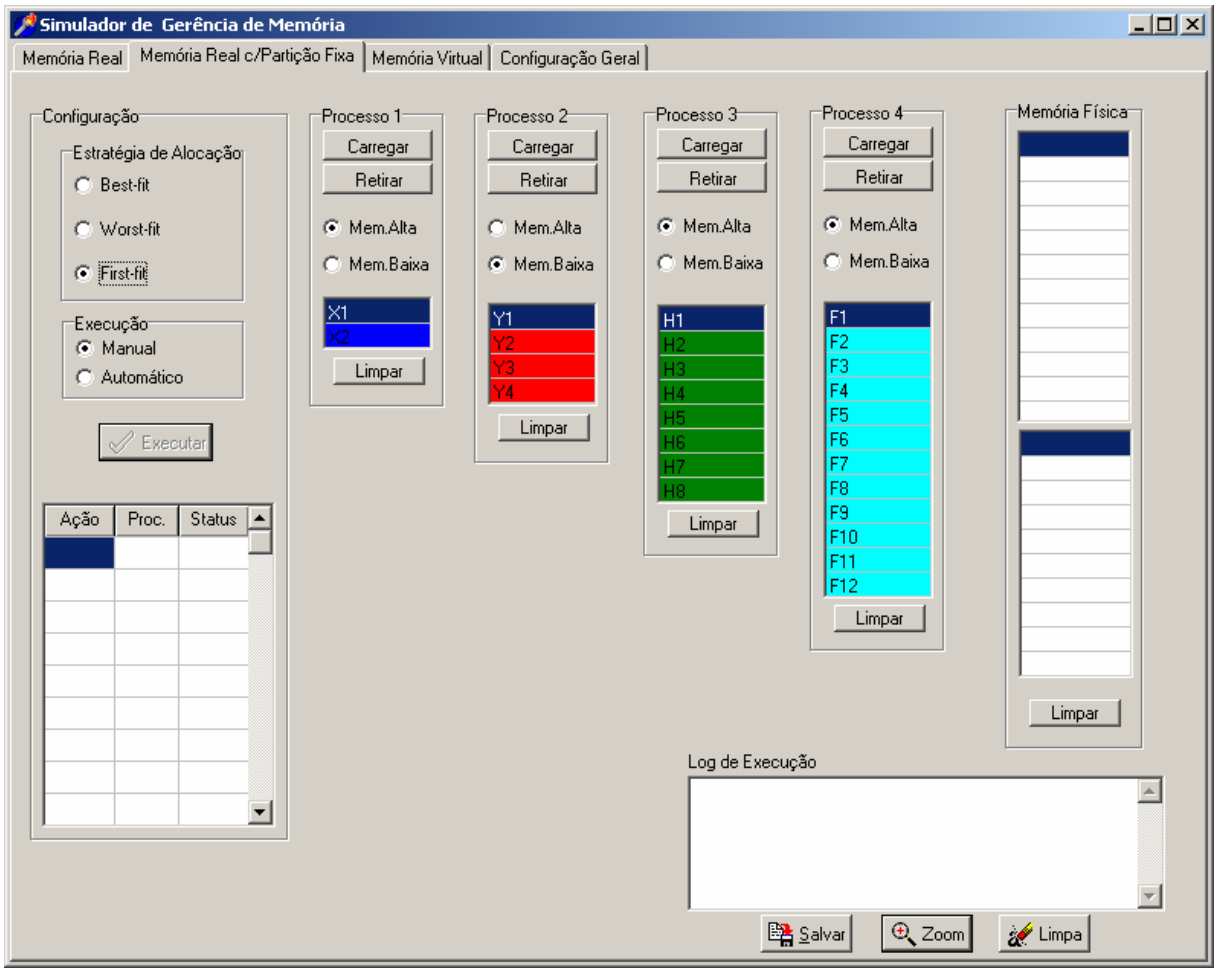


Figura 39 – Escolha da partição.

Pode-se observar na figura 40 que o processo 1 foi carregado para a memória nas primeiras páginas livre da memória alta, ficando disponível um bloco de 10 páginas. Deve-se observar que o processo 2 será carregado na memória baixa (conforme definido anteriormente), deixando um bloco de 6 páginas livres.

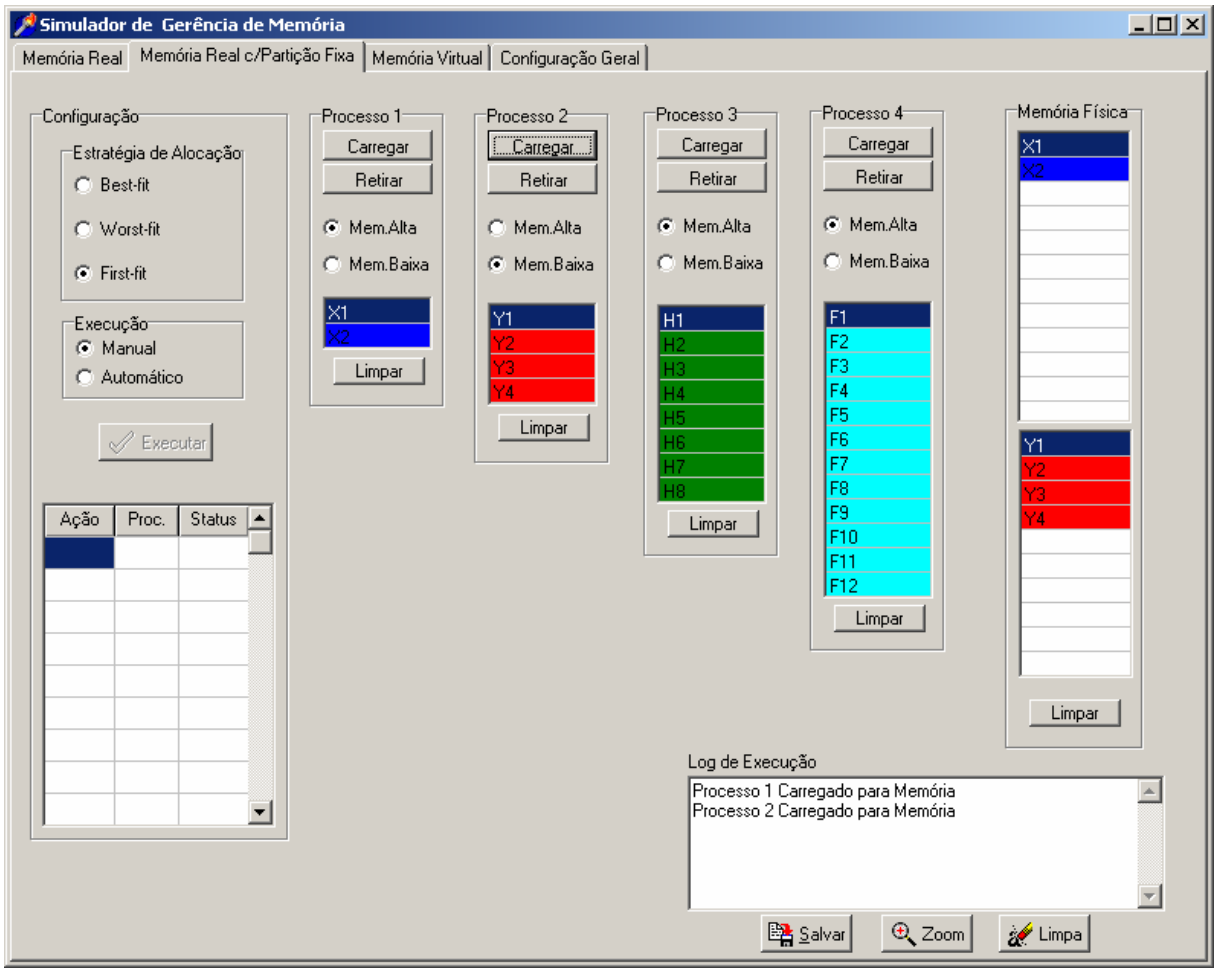


Figura 40 – Carrega processos 1 e 2.

Ao ser carregado o processo 3 (figura 41), o método fisrt-fit selecionada bloco disponível na memória alta, este método ira pegar a primeiro bloco de páginas livres com tamanho suficiente para sua alocação.

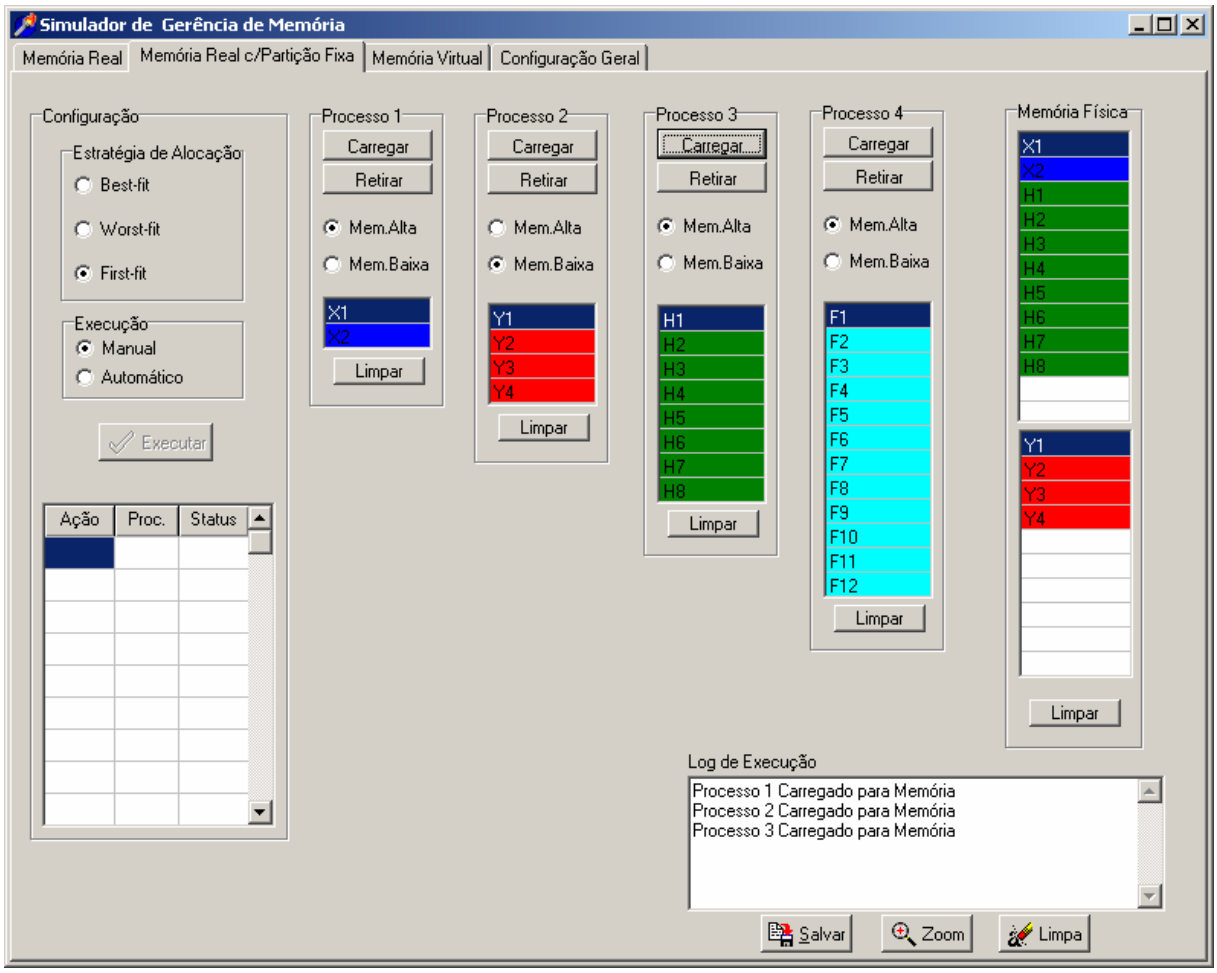


Figura 41 – Carrega processos 3.

Percebe-se que não é mais possível carregar os demais processos porque não há memória livre disponível. Os outros processos poderão ser carregados quando algum, ou todos os processos que estão executando finalizem e liberem a memória para carga dos demais.

Na gerência de memória real com partição fixa os outros métodos de alocação (Best-fit e Worst-fit) terão a mesma funcionalidade descrita no estudo de caso da gerência de memória real.

4.4.4 GERÊNCIA DE MEMÓRIA VIRTUAL

Nesta seção é apresentado um exemplo de utilização do sistema para simular o comportamento do sistema de gerenciamento de memória virtual. Para tanto será considerado o seguinte cenário:

a) 2 processos a serem carregados;

b) memória física possui 18 páginas livres para alocação;

c) tamanho da soma dos programas supera o tamanho do bloco de páginas livre da memória;

d) 2 tabelas de páginas para guardar endereço de localização das páginas na memória real;

e) o disco, onde se encontra o programa.exe e área livre para swap.

A simulação começa com o usuário fazendo a carga para a memória do primeiro processo (figura 42). Deve-se observar que, o programa foi carregado inicialmente da página 6 ate página 12 da memória, ficando sobrando um bloco de 6 páginas disponíveis.

Na tabela de páginas do primeiro programa, pode-se verificar qual o endereço real da página do programa na memória real.

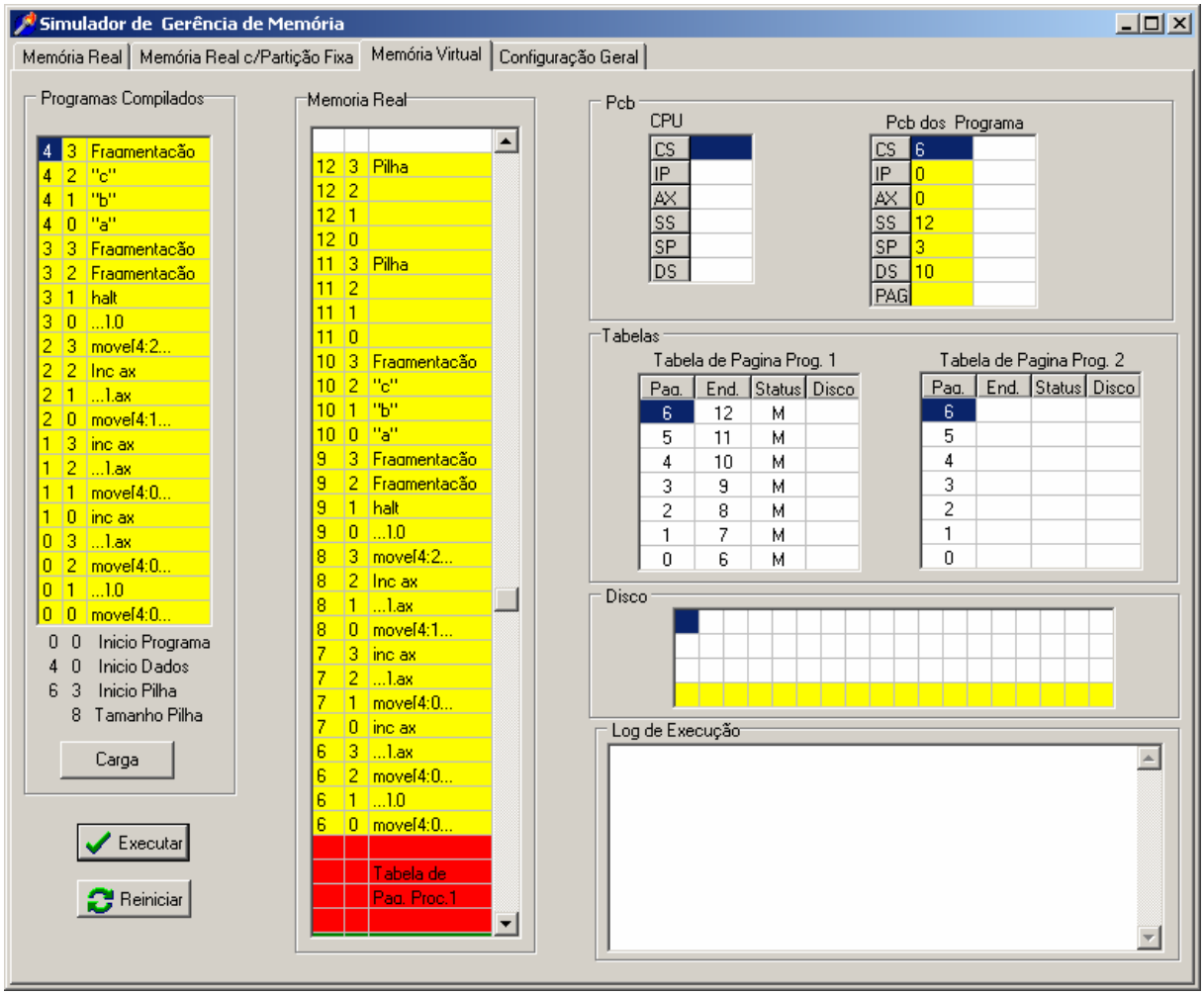


Figura 42 – Carrega do primeiro programa.

Na figura 43 pode-se visualizado a carga do segundo programa para a memória real. Deve-se observar que, não é possível carregar todo o programa para o bloco de 6 páginas

disponíveis na memória real. Pode-se perceber o funcionamento do algoritmo de substituição de página (FIFO).

A página escolhida como vítima é sempre aquela que esta á mais tempo na memória, neste caso a página 1 do primeiro programa que estava alocada na página 6 da memória real. Depois de verificado qual a página vitima, esta será transferida para o disco e transferido para a memória principal a página solicitada.

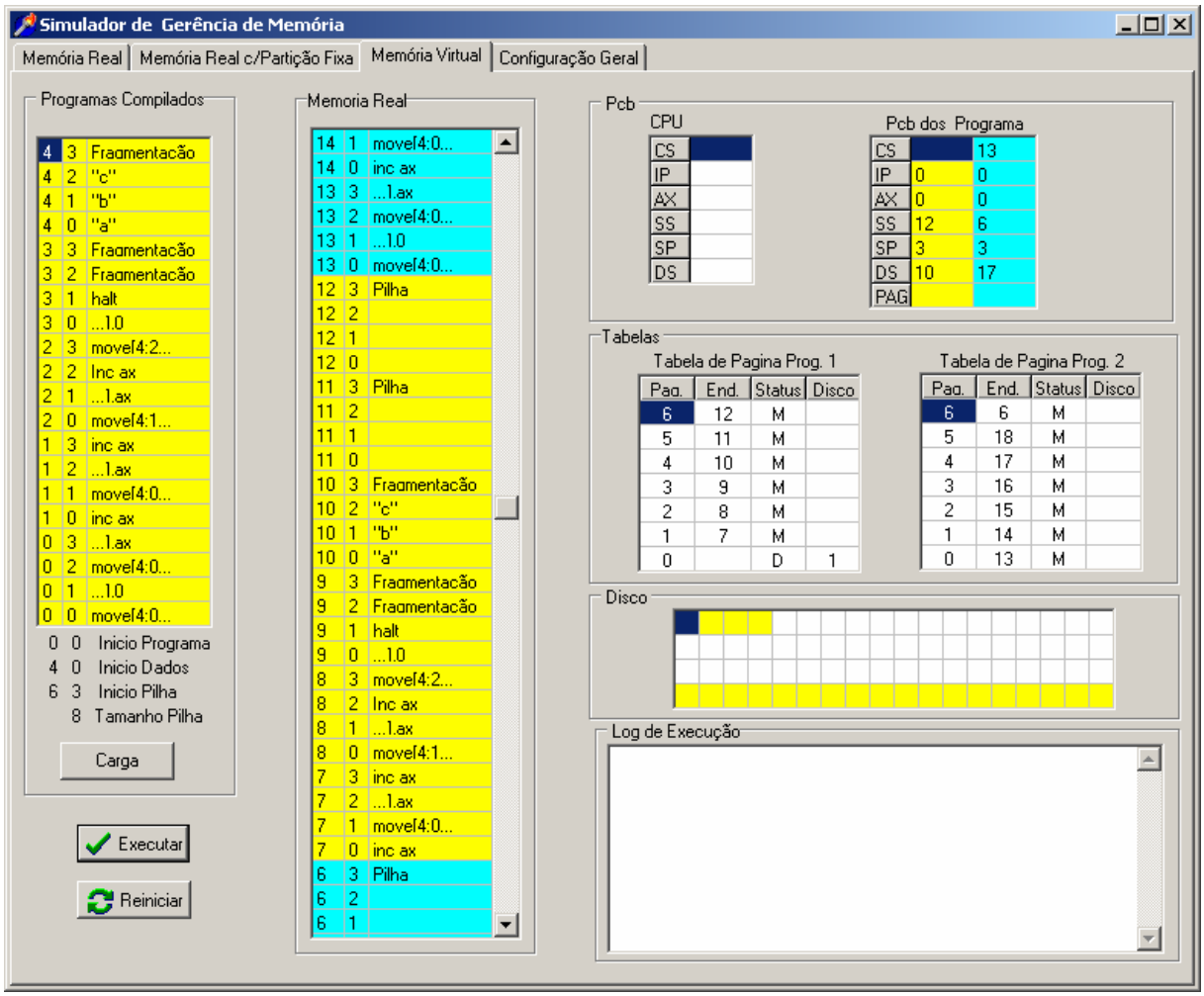


Figura 43 – Carrega do segundo programa.

Um acesso à memória real pode ter dois encaminhamentos distintos. Quando a página lógica acessada pelo processo está marcada como “M” (Memória) no status na tabela de páginas, o endereço lógico é transformado em endereço real, e o acesso transcorre normalmente. Quando a página lógica acessada pelo processo está indicado como “D” (Disco) no status, é gerada uma interrupção por falta de página (*page fault*), e as seguintes ações são tomadas (figura 44) .

a) o processo que gerou a interrupção de falta de página é suspenso e tirado da fila do processador e inserido em uma fila especial (fila dos processos esperando página lógica);
b) uma página física livre é selecionada para receber pagina do disco. Caso não tenha nenhuma página livre, o sistema adota a política de substituição de páginas, selecionado a página escolhida para ser gravada na área de swap;
c) a página lógica acessada deve ser localizada no disco;
d) uma operação de disco deve ser solicitada transferindo a página solicitada que está no disco para a memória principal, atualizando o status e endereço da página na tabela de páginas do processo que esta executando. Este processo é conhecido como *swapping* (seção 3.3.6.).

Enquanto isso, o processo que executava fica suspenso, à espera da página de que ele necessita para continuar sua execução. O sistema pode então selecionar outro processo para executar.

a) a tabela de página do processo é corrigida para indicar que a página lógica causadora da interrupção ou *page fault* esta com status “M” indicando que a página esta alocada na memória principal;
b) o processo é retirado da fila dos processos esperando página lógica e colocado na fila do processador.

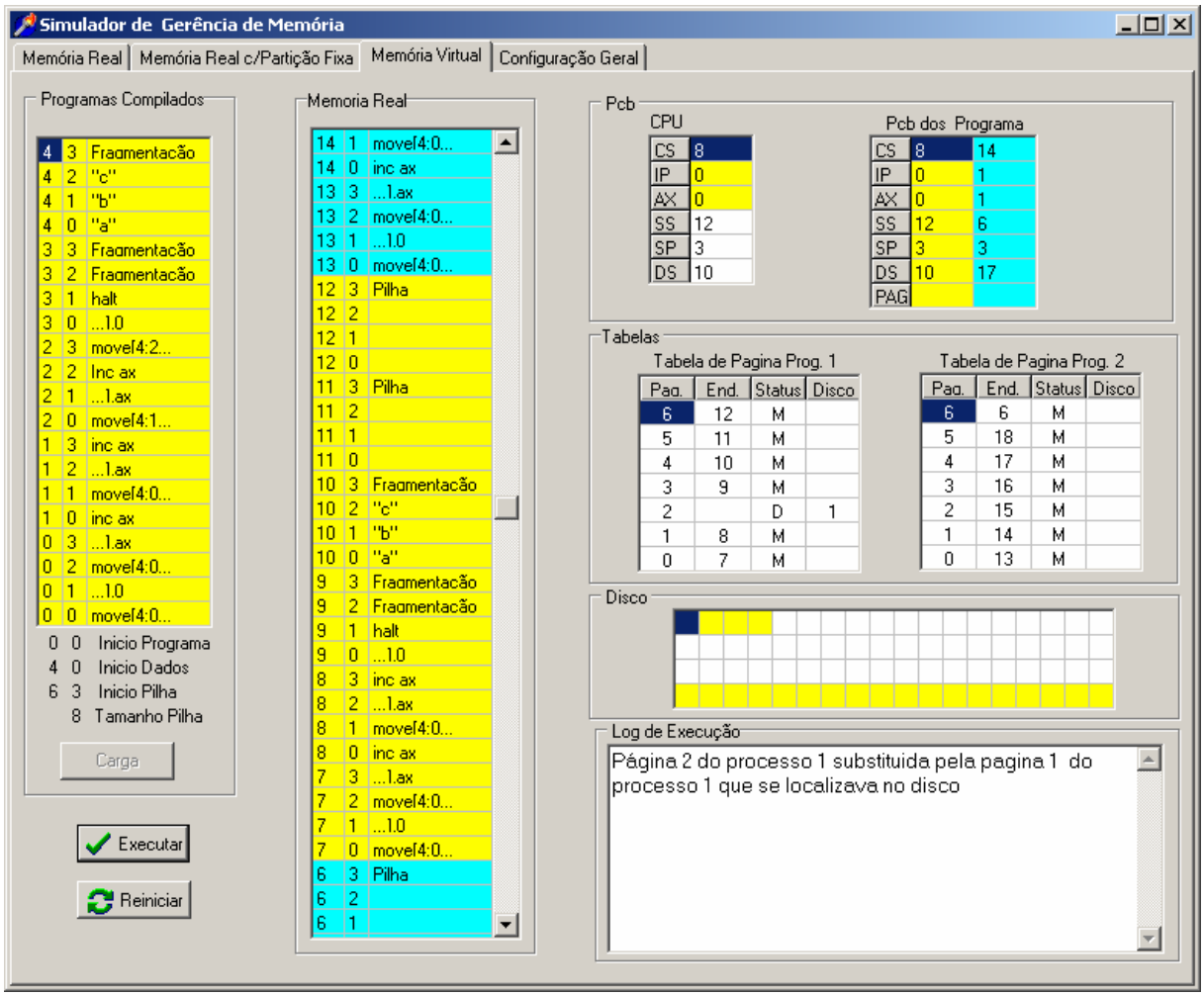


Figura 44 – Substituição de página.

No apêndice B pode-se verificar a descrição completa de uma execução, de um processo alocado na memória real.

4.5 RESULTADOS E DISCUSSÃO

O simulador é um software educacional voltado ao suporte ao ensino de conceitos de gerência de memória em uma ambiente de sistema operacional. Contudo, o software precisa ser utilizado dentro de um contexto pedagógico, o que significa que o software não é um fim em si mesmo, mais um meio de se ampliar as possibilidades de ensino.

5 CONCLUSÕES

A partir dos estudos, constata-se que a ferramenta introduz uma nova alternativa para o aprendizado dos métodos de gerência de memória real e virtual.

Através deste simulador o usuário poderá esclarecer algumas dúvidas sobre o funcionamento de mecanismos de gerência de memória, o que torna o entendimento muito mais prático e amigável. Para operar este simulador, o usuário deve estar familiarizado com alguns conceitos básicos do funcionamento dos sistemas operacionais, principalmente gerência de memória, visto que durante a simulação dos métodos de gerência poderão ser feitos algumas configurações para demonstração dos tipos de algoritmos de estratégia de alocação para que possa ser vista a diferença de cada uma.

Os objetivos e os requisitos previamente definidos foram todos atingidos. Portanto, este protótipo já está apto a ser empregado em disciplinas como Sistemas Operacionais e Arquitetura de Computadores, a fim de facilitar o ensino do sistema operacional, pois além de demonstrar graficamente o funcionamento de alguns mecanismos de gerência de memória o usuário, ainda pode fazer um estudo nos logs gerados durante a execução do processo simulação.

As maiores dificuldades encontradas estão relacionadas com a descoberta de mostrar visualmente estas simulações, pois se tratava de um assunto muito conceitual.

5.1 EXTENSÕES

Como extensões para este trabalho sugere-se:

a) permitir a carga de código diferente na memória;
b) implementar indicadores chamando a atenção do usuário para eventos importantes;
c) aprimorar o controle das mensagens de *log*;
d) implementar os mecanismos de segmentação e segmentação e paginação;
e) implementar outras estratégias de substituição de páginas: LRU, MRU.

REFERÊNCIAS BIBLIOGRÁFICAS

BIZZOTO, Carlos Eduardo Negrão. **O Aprendiz: ambiente extensível para o aprendizado Distribuído.** 2003. 123 f. Dissertação (Pós-Graduação em Engenharia de Produção) – Universidade Federal de Santa Catarina, Florianópolis.

KOMOSINSKI, Leandro José. **Um novo significado para a educação tecnológica fundamentado na informática como artefato mediador da aprendizagem**. 2000. 146 f. Tese de Doutorado (Doutor em Engenharia da Produção) - Programa de Pós- Graduação em Engenharia de Produção da Universidade Federal de Santa Catarina. Universidade Federal de Santa Catarina, Florianópolis.

MACHADO, Francis Berenger; MAIA, Luiz Paulo. **Arquitetura de sistemas operacionais.** Rio de Janeiro: LTC, 2002.

MAIA, Luiz Paulo. **Simulador para ensino de sistemas operacionais**. Rio de Janeiro, [2001]**.** Disponível em: < http://www.training.com.br/sosim >. Acesso em: 20 out. 2004.

MATTOS, Mauro Marcelo. **Sistemas operacionais.** 2003. 50 F. Notas de aula (Disciplina de Sistema de Operacionais, Curso de Ciências da Computação).Centro de Exatas e Naturais, Departamento de Sistema e Computação, Universidade Regional de Blumenau, Blumenau.

MATTOS, Mauro M.; FERNANDES, Andrino; LÓPEZ, Oscar C. Sistema especialista para apoio ao aprendizado de lógica de programação. In: Congresso Ibero-americano de Educação Superior em Computação, 7., 1999, Florianópolis. **Anais...** Assunção: Universidad Autônoma de Asunción, 1999.

OLIVEIRA, Romulo Silva de; CARISSIMI, Alexandre da Silva; TOSCANI, Simão Sirineo. **Sistema operacionais**. Porto Alegre: Sagra Luzzatto, 2000.

SILBERRSCHATZ, Abraham; GALVIN, Peter; GAGNE, Greg. **Sistemas operacionais**: conceitos e aplicações. Rio de Janeiro: Campus, 2004.

SILVA, Carlos Alberto. **Informática na educação**. 2000. 45 f. Trabalho de Conclusão de Curso (Pós Graduação) - Unidade Universitária de Canoínhas, Universidade Regional de Blumenau, Canoínhas.

SLOCZINSKI, Helena; DAL' COL Zene, Carlos Mário; NITSKE, Júlio Alberto; DE LIMA, José Valdeni. **Integração do cd-rom com a internet - ambiente para aprendizagem colaborativa.** [2000]. Disponível em: <http://www.c5.cl/ieinvestiga/actas/ribie2000/demos\ 242>. Acesso em: 20 out. 2004.

TAJRA, Sanmya Feitosa. **Informática na educação:** professor na atualidade. São Paulo: Érica, 1998.

TANENBAUM, Andrew S.**Sistemas operacionais modernos**. São Paulo: Prentice Hall,2003.

WEBER, Raul Fernando. **Fundamentos de arquitetura de computadores.** Porto Alegre: Sagra Luzzatto, 2001.

APÊNDICE A – Algoritmo que verifica onde será alocado o processo na memória conforme estratégia de alocação de partição

```
function TClassAloc.VerificaAloc(SGA: TStringAlignGrid; TamProPri, QtdPro, TpoAloc:
Integer): Integer;
type
  TAloc = record
     Inicio : Integer;
     Fim    : Integer;
     Qtd    : Integer;
     Resto  : Integer;
   end;
var
  X,A : Integer;
  Aloc: array[0..22] of TAloc; // array do tipo da estrutura TAloc
  AlocBol : TAloc;  // variável para utilização no método da bolha
  I,J : integer;
begin
  X := 0;
  A := 0;
  // Limpa e aloca  Array
  FillChar(Aloc,sizeof(Aloc),0);
 // Verifica o se a pagina é menor que o numero Maximo de pagina
  While ( X < TamProPri) do
      begin
        if  ( SGA.Cells[0,X] = '' ) then
          begin
            if   ( Aloc[A].Inicio = 0 ) then
                 Aloc[A].Inicio := X + 1;
            if   ( X = TamProPri - 1 ) then
               begin
                  // Calcula o tamanho de espaço livre existente entre os processos
                  Aloc[A].Fim := X + 1;
                  Aloc[A].Qtd := (Aloc[A].Fim - Aloc[A].Inicio) + 1;
                  Aloc[A].Resto := Aloc[A].Qtd - QtdPro;
                  // Verifica se o espaço suporta o processo que ira ser inserido
                  if   ( Aloc[A].Resto < 0 ) then
                       FillChar(Aloc[A],1 * sizeof(TAloc),0);
               end;
           end
        else
           if  (( Aloc[A].Fim = 0 ) and ( Aloc[A].Inicio <> 0 )) then
             begin
                  // Calcula o tamanho de espaço livre existente entre os processos
               Aloc[A].Fim := X - 1;
               Aloc[A].Qtd := (Aloc[A].Fim - Aloc[A].Inicio) + 1;
               Aloc[A].Resto := Aloc[A].Qtd - QtdPro;
                // Verifica se o espaço suporta o processo que ira ser inserido

               if   ( Aloc[A].Resto < 0 ) then
```

```
                    FillChar(Aloc[A],1 * sizeof(TAloc),0)
                else  Inc(A);
            end;
        Inc(X);
    end;
// Verifica tipo de Estratégia de Alocação
if  ( TpoAloc = 0 ) then
     // Retorna posição inicial onde será inserido  o processo
     Result := Aloc[0].Inicio - 1
else
   begin
     // Feito método da bolha para ordenar crescente o resultado obtido
     if  ( TpoAloc = 1 ) then
       begin
         I := A;
         While  ( I > 0 ) do
              begin
                j := 0;
                While  ( J < I ) do
                     begin
                       if  ( Aloc[j].Resto > Aloc[j+1].Resto ) then
                         begin
                           AlocBol := Aloc[j];
                           Aloc[j] := Aloc[j+1];
                           Aloc[j+1] := AlocBol;
                         end;
                       Inc(j);
                     end;
                dec(i);
              end;
       end
     else
       // Feito método da bolha para ordenar decrescente o resultado obtido
       begin
         I := A;
         While  ( I > 0 ) do
             begin
               j := 0;
               While  ( J < I ) do
                   begin
                     if  ( Aloc[j].Resto < Aloc[j+1].Resto ) then
                       begin
                         AlocBol := Aloc[j];
                         Aloc[j] := Aloc[j+1];
                         Aloc[j+1] := AlocBol;
                       end;
                     Inc(j);
                   end;
               dec(i);
             end;
```

```
          end;
        // Retorna posição inicial onde será inserido  o processo
        Result := Aloc[0].Inicio - 1;
      end;
end;
```

APÊNDICE B – Seqüência de execução de um único processo carregado na memória em gerência de memória virtual

O programa começa a executar e faz *fetch* do endereço 6:0. A unidade de controle identifica que é uma instrução de 2 bytes e incrementa IP para buscar o segundo byte da instrução.O segundo byte é recuperado da memória e a unidade de controle identifica que a instrução é: move [4:0],0, ou seja move o valor zero para a posição 4:0 no espaço de endereços virtuais do processo. Ao executar a instrução, a cpu envia o endereço 4:0 para a MMU, a qual acessando a tabela de páginas do processo identifica que a página 4 está na realidade no endereço real 10 da memória e que o deslocamento é zero em relação ao início da página (variável "a"). A instrução move [10:0], 0 é executada fazendo com que o valor da posição de memória 10:0 seja atualizada para 0.

O apontador de instruções IP, é automaticamente incrementado para apontar para a instrução seguinte, neste caso: 6:2. A unidade de controle identifica que é uma instrução de 2 bytes e incrementa IP para buscar o segundo byte da instrução. O segundo byte é recuperado da memória e a unidade de controle identifica que a instrução é: move [4:0],ax , ou seja move o conteúdo do registrador ax da CPU para a posição 4:0 no espaço de endereços virtuais. Ao executar a instrução, a cpu envia o endereço 4:0 para a MMU, a qual acessando a tabela de páginas do processo identifica que a página 4 está na realidade no endereço real 10 da memória e que o deslocamento é zero em relação ao início da página (variável "a"). A instrução move ax,[10,0] é executada fazendo com que o conteúdo da posição [10:0] seja transferido para o registrador ax da CPU.

O apontador de instruções IP, é automaticamente incrementado para apontar para a instrução seguinte. Neste caso: 7:0 uma vez que o próximo byte de memória está no segmento 7 e deslocamento 0. Uma nova fase de fetch inicia-se e a cpu recupera a instrução "inc ax". A unidade de controle identifica que esta é uma instrução de 1 byte e a executa fazendo com que o valor do registrador ax seja incrementado e passe a armazenar o valor 1.

Automaticamente o apontador de instruções é incrementado para apontar para a próxima instrução, ou seja 7:1. Uma nova fase de fetch inicia-se e a cpu recupera o primeiro byte da instrução move [4:0],ax. O segundo byte é recuperado da memória e a unidade de controle identifica que a instrução é: move [4:0],ax , ou seja move o conteúdo do registrador ax da CPU para a posição 4:0 no espaço de endereços virtuais. Ao executar a instrução, a cpu envia o endereço 4:0 para a MMU, a qual acessando a tabela de páginas do processo identifica

que a página 4 está na realidade no endereço real 10 da memória e que o deslocamento é zero em relação ao início da página (variável "a").

O apontador IP é automaticamente incrementado para apontar para a instrução seguinte. Uma nova fase de fetch inicia-se e a cpu recupera a instrução "inc ax". A unidade de controle identifica que esta é uma instrução de 1 byte e a executa fazendo com que o valor do registrador ax seja incrementado e passe a armazenar o valor.

Automaticamente o apontador de instruções é incrementado para apontar para a próxima instrução, ou seja 7:0. Uma nova fase de fetch inicia-se e a cpu recupera o primeiro byte da instrução move [4:1],ax. O segundo byte é recuperado da memória e a unidade de controle identifica que a instrução é: move [4:1],ax , ou seja move o conteúdo do registrador ax da CPU para a posição 4:1 no espaço de endereços virtuais. Ao executar a instrução, a cpu envia o endereço 4:1 para a MMU, a qual acessando a tabela de páginas do processo identifica que a página 4 está na realidade no endereço real 10 da memória e que o deslocamento é um em relação ao início da página (endereço da variável "b").

O apontador IP é automaticamente incrementado para apontar para a instrução seguinte. Uma nova fase de fetch inicia-se e a cpu recupera a instrução "inc ax". A unidade de controle identifica que esta é uma instrução de 1 byte e a executa fazendo com que o valor do registrador ax seja incrementado e passe a armazenar o valor.

Automaticamente o apontador de instruções é incrementado para apontar para a próxima instrução, ou seja 7:3. Uma nova fase de fetch inicia-se e a cpu recupera o primeiro byte da instrução move [4:2],ax. O segundo byte é recuperado da memória. Para tanto a CPU precisa incrementar o registrador de segmento e zerar o registrador de deslocamento ficando apontando para 8:0. A unidade de controle identifica que a instrução é: move [4:2],ax , ou seja move o conteúdo do registrador ax da CPU para a posição 4:2 no espaço de endereços virtuais. Ao executar a instrução, a cpu envia o endereço 4:2 para a MMU, a qual acessando a tabela de páginas do processo identifica que a página 4 está na realidade no endereço real 10 da memória e que o deslocamento é 2 em relação ao início da página (endereço da variável "c").

O apontador IP é automaticamente incrementado para apontar para a instrução seguinte em 8:1. Observe que, a instrução halt em um processador real faz com que o mesmo congele.

No simulador, esta instrução está sendo utilizada para representar uma chamada de sistema que encerraria o processo.