Simulação do Processo de Substituição de Páginas em Gerência de Memória Virtual

Fagner do Nascimento Fonseca[a], Orientador(a): Flávia Maristela S. Nascimento[b]

[a]*Instituto Federal da Bahia*
[b]*Instituto Federal da Bahia - GSORT*

Resumo

Este trabalho apresenta uma ferramenta de apoio ao processo de ensino aprendizagem da disciplina de sistemas operacionais. A ferramenta desenvolvida apresenta a simulação do funcionamento da memória virtual, com o foco nos algoritmos de substituição de páginas, demonstrando o funcionamento dos algoritmos e como ocorrem as interações entres os principais componentes deste mecanismo. Para melhorar o entendimento do assunto e garantir que o sistema atenda a seu objetivo, a utilização de recursos visuais é de fundamental importância. No sentido de apoiar o processo de ensino-aprendizagem, a aplicação conta não apenas com a simulação visual dos algoritmos de substituição de páginas, mas permite também que os alunos possam implementar outros algoritmos, através de classes pensadas para este propósito.

Palavras Chave: Simulação, Gerência de Memória Virtual, Ferramenta de Ensino-Aprendizagem, Sistemas Operacionais

1. Introdução

A memória é de fundamental importância para o funcionamento de um sistema de computação, pois é nela que são armazenados os dados necessários para a execução dos programas, inclusive do sistema operacional. O ideal seria que o sistema tivesse a sua disposição uma grande quantidade de memória, que fosse barata e não-volátil, ou seja, que os dados não fossem perdidos quando o computador fosse desligado [1, 2].

A tecnologia atual ainda não nos permite essa realidade, portanto, os sistemas operacionais obedecem uma hierarquia de memória, que é dividida nas seguintes categorias: registradores, cache, memória principal, armazenamento secundário e armazenamento permanente. Conforme pode ser visto na Figura 1, quando avançamos para o topo da pirâmide, a memória fica menor (em termos de capacidade), mais cara e mais rápida; quando descemos na hierarquia, a memória fica maior (em termos de capacidade), mais barata e mais lenta. [3, 2].

A memória mais rápida é proporcionalmente a mais cara (Figura 1). Devido a isso, os sistemas são construídos com pequenas quantidades de memória rápida. Além disso, memórias mais rápidas como por exemplo, registrador e *cache*, são voláteis. Sendo assim, faz-se necessário outros tipos de tecnologias de memória que mantenham os dados armazenados mesmo sem a presença de energia elétrica [3].

O gerenciador de memória é a parte do sistema operacional que cuida da hierarquia de memória. O foco dele está na memória principal, uma vez que esta possui o meio termo em relação a tamanho, velocidade e preço. É papel do gerenciador de memória conhecer a disponibilidade dos espaços de memória, alocar e desalocar processos para execução, transferir processos entre a memória principal e o armazenamento secundário quando a memória não for suficiente para alocar os processos [3, 4, 5, 2].

Com o passar do tempo, o computador foi exigindo cada vez mais espaço da memória. Isso ocorreu não só pelo aumento na complexidade e no tamanho dos programas, mas também devido ao avanço das técnicas utilizadas para gerenciamento de

memória. A memória principal ainda era um recurso muito caro para a época e embora seu preço estivesse diminuindo e o seu tamanho aumentando, a quantidade de programas que demandavam grandes quantidades de memória aumentava muito mais, agravando ainda mais o problema [3, 4, 5, 2, 6].

Para tentar resolver esta questão de memória escassa, foi criado o conceito de memória virtual. Com a memória virtual é possível criar a ilusão de que o sistema operacional possui uma quantidade maior de memória disponível do que a que realmente possui. Assim, programas muito grandes que antes eram impossibilitados de executar, por causa do tamanho da memória disponível, agora podem executar [4].

Para um programa executar, não é necessário que ele seja carregado todo na memória. Existem trechos que podem nunca ser chamados, como por exemplo, trechos que tratam erros do sistema. Algumas rotinas também nunca serão executadas se o usuário não solicitar. Mesmo se o programa executar todas as suas rotinas, elas não serão executadas e consequentemente carregadas ao mesmo tempo. Com as técnicas de memória virtual, somente as partes do programa que são necessárias para ele executar serão carregadas na memória [7, 1].

Entender como tudo isso funciona, no entanto, não é uma tarefa das mais simples. Em geral, nos cursos da área de Computação, os alunos de disciplinas como Arquitetura de Computadores e/ou Sistemas Operacionais apresentam dificuldade para entender a interação entre *hardware* e *software* que envolve o processo de gerência de memória, mais especificamente os algoritmos responsáveis pela administração e funcionamento da memória virtual. Um dos fatores que pode levar a isto é o alto grau de abstração requerido por esta temática, bem como o elevado nível de conhecimento acerca do funcionamento de cada um dos elementos envolvidos. Por estas razões, muitas vezes pode ser difícil para o aluno conseguir visualizar de maneira clara como todo esse processo funciona.

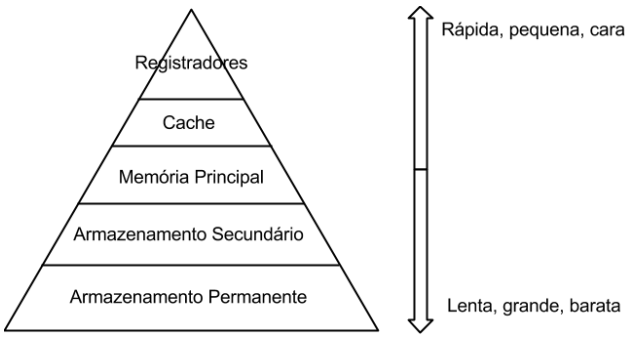

Figura 1: Hierarquia de Memória [3].

O presente trabalho simula o processo de gerência de memória virtual, especificamente os algoritmos de substituição de páginas, demonstrando graficamente como todo o mecanismo funciona. O objetivo do trabalho é servir como ferramenta de apoio para disciplinas que abordam essa temática.

2. Motivação

A disciplina Sistemas Operacionais é obrigatória na maioria nos cursos de computação. Os temas que são abordados por ela e a forma como são ministrados torna a compreensão desses assuntos um tanto complexa. Experiências observadas entre professores e alunos mostra como é difícil a compreensão dos assuntos dessa disciplina, bem como a aplicação prática dos mesmos [8].

A forma como a disciplina é ensinada, utilizando somente teoria, dificulta o entendimento e em alguns casos pode até desmotivar o aluno, que pela falta de ferramentas capazes de mostrar na realidade os conceitos vistos na teoria, fica bastante longe do objeto de estudo. Um curso de sistemas operacionais com apenas aulas teóricas não colabora na compreensão dos conceitos abordados. Pode-se observar que muitos alunos concluem a disciplina conhecendo diversos conceitos e algoritmos, mas sem saber como integrar tudo o que foi aprendido [8, 9].

Nos casos em que o curso possui uma abordagem prática, normalmente as práticas de laboratórios são feitas modificando código de algum sistema operacional de código aberto, como por exemplo o Linux. Podem ser feitas implementações de algoritmos clássicos, como por exemplo, jantar dos filósofos, barbeiro dorminhoco e caixeiro viajante, esses algoritmos permitem que os alunos ponham em prática os conceitos de comunicação e sicronização entre processos. Para essas implementações podem ser utilizadas algumas linguagens de programação como C ou Java, ou pode ser utilizado o próprio shell do sistema operacional. Outra forma de prática é utilizar

simuladores [8].

Simuladores envolvem a criação de modelos dinâmicos e simplificados do mundo real. O uso deles pode aumentar o aprendizado comparado com o uso de outras técnicas. Na área de computação existem simuladores que auxiliam no ensino de várias disciplinas, como redes de computadores, técnicas de programação, arquitetura de computadores e sistemas operacionais [8].

Como pode ser observado, o uso de ferramentas práticas nessa disciplina é de fundamental importância. Os objetivos que devem ser buscados nas aulas práticas são mostrados a seguir [9]:

1. consolidar a compreensão dos mecanismos e estruturas básicas do funcionamento de um núcleo de sistema operacional típico;
2. compreender claramente como as diversas partes constituintes de um sistema operacional interagem e se integram;
3. observar o impacto das políticas internas do núcleo no funcionamento das aplicações;
4. compreender o impacto nas escolhas de implementação;
5. desenvolver a capacidade de propor e testar suas próprias soluções.

Para a elaboração deste trabalho, foi feita uma pesquisa para saber o interesse dos alunos em uma ferramenta de simulação que auxiliasse a disciplina de sistemas operacionais, se eles já utilizaram alguma ferramenta desse tipo e se a ferramenta ajudou. Sessenta e duas pessoas responderam ao questionário, alguns resultados obtidos são mostrados a seguir:

1. sobre a importância de uma ferramenta de simulação, 46,77% das pessoas acredita que é importante, 50% acredita que é muito importante e 3,23% acredita que tem pouca importância;
2. 48,39% das pessoas não usou nenhum tipo de ferramenta de simulação quando fez a disciplina de sistemas operacionais, 48,39% usou e afirma que a ferramenta ajudou no aprendizado e 3,23% usou algum tipo de ferramenta e afirma que não ajudou no aprendizado;
3. 98,39% dos alunos gostaria de usar ou gostaria de ter usado uma ferramenta de simulação de gerência de memória virtual e 1,61% não gostaria de usar.

Foram realizadas pesquisas em busca de aplicações que auxiliassem de maneira eficaz a entender a gerência de memória virtual e a visualizar como esse mecanismo funciona através da simulação, mas poucas aplicações a respeito foram encontradas.

Baseado nestes aspectos, este trabalho busca preencher esta lacuna através de uma ferramenta visual, que apresenta as etapas da gerência de memória virtual e ainda permite que o aluno possa implementar outros algoritmos, através das estruturas de dados e classes criadas com este propósito. O que certamente agrega aspectos práticos às disciplinas de Sistemas Operacionais e/ou Arquitetura de Computadores.

3. Estrutura do trabalho

O presente trabalho está organizado conforme a estrutura mostrada a seguir:

- a Seção 4 trata sobre como e porque gerenciar memória; explica a hierarquia de memória, a organização da memória e mostra os principais algoritmos de alocação de memória;
- a Seção 5 explica como funciona a memória virtual, quais são os desafios, trata sobre a tradução de endereços virtuais para endereços reais, trata sobre a paginação e explica os principais algoritmos de substituição de páginas;
- a Seção 6 mostra o trabalhos correlatos, são citados exemplos de simuladores tanto de gerência de memória virtual quanto de gerência de memória simples;
- a Seção 7 trata sobre a ferramenta em si. Explica a arquitetura do sistema, mostra quais são os principais componentes, mostra como a simulação foi implementada e explica como criar novos algoritmos de substituição de páginas;
- a Seção 8 apresenta os testes realizados e os resultados obtidos.

- a Seção 9 mostra a conclusão do trabalho e os possíveis projetos que podem ser iniciados tendo como base a ferramenta proposta.

4. Gerência de Memória

A memória não possui divisões uniformes onde cada processo pode ser armazenado, ela nada mais é do que um vetor de *bytes*. Portanto, é preciso saber como alocar a memória disponível para os processos que precisam executar. O método de gerência de memória que será empregado em um determinado sistema depende também do *hardware* que está disponível [3, 7, 1].

Nos primeiros sistemas, somente um processo poderia ser executado por vez e ele permanecia na memória até terminar a sua execução. Uma parte da memória era dedicada ao sistema operacional e o restante era todo disponível para o processo que estivessse em execução (Figura 2). O programador era totalmente responsável por gerenciar os espaços de memória utilizados pelo processo. Se um processo fosse grande demais para ser carregado, a memória principal deveria ser aumentada ou então o programa deveria ser modificado de maneira que ele coubesse na memória. Esses sistemas são chamados de monousuário, pois trata de um processo de somente um usuário (Figura 2) [4, 5, 7, 2].

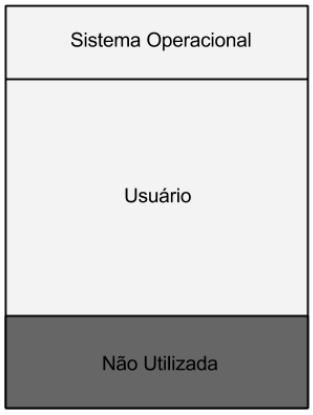

Figura 2: Alocação de memória contígua em sistema monousuário [4].

Para tentar resolver o problema de somente um processo poder ser executado por vez, foram criadas as partições fixas. Desta forma, a memória era dividida de maneira lógica em várias partes, que foram chamadas de partições. Essas partições poderiam possuir tamanhos iguais ou variados, no entanto, uma vez o tamanho fosse definido, este não poderia mais ser modificado. Cada processo poderia ser alocado a uma determinada partição caso ele coubesse nela e caso ela não estivesse ocupada por outro processo [4, 5, 7].

Assim, surgiu um novo conceito, o de multiprogramação. A multiprogramação permite que vários processsos sejam mantidos na memória principal simultaneamente. No esquema de partições fixas os processos continuam sendo alocados em um espaço de memória contíguo e o processo precisa ser carregado todo na memória e deve permanecer carregado até o final de sua execução [4, 5, 7].

Nesse novo esquema surgiram alguns problemas, o primeiro deles é o de proteger o espaço de memória de um processo. No esquema anterior que somente um processo podia ser executado por vez, bastava garantir a proteção do espaço de memória do sistema operacional. No esquema de partições fixas é preciso garantir também que um processo não invada a área de outro processo acidentalmente ou intencionalmente [4, 5, 7].

Outro problema que surgiu nesse esquema foi a fragmentação interna. Como as partições são de tamanho fixo, caso o processo não ocupe toda a partição, ficará sobrando espaço de memória que não poderá ser utilizado (Figura 3). Esses espaços que sobram nas partições geram a fragmentação interna. Com a fragmentação interna, será possível que um processo que precisa ser executado fique impossibilitado de executar por falta de partições livres, apesar da soma das sobras oriundas das partições serem o suficiente para o mesmo executar. Como as partições são de tamanho fixo, não é possível reorganizar a memória juntando os fragmentos em uma única partição [3, 4, 5, 7].

Um outro esquema utilizado para permitir a execução de vários programas é o de partições dinâmicas. Nesse esquema, o tamanho da partição é definido quando o processo é carregado. O problema da fragmentação interna é resolvido, mas aparece o problema da fragmentação externa. Apesar das partições serem adequadas ao tamanho do processo, pode acontecer que conforme os processos

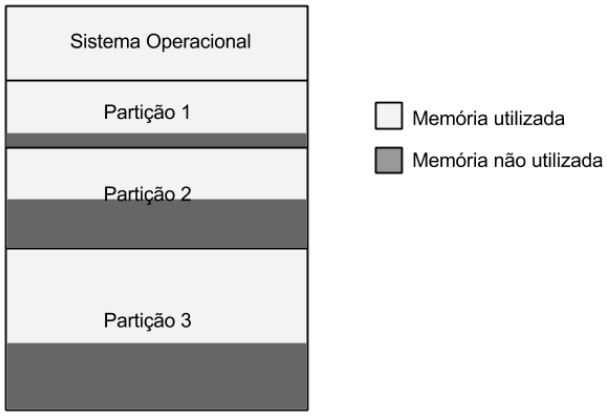

Figura 3: Fragmentação interna em um esquema de partição fixa [4].

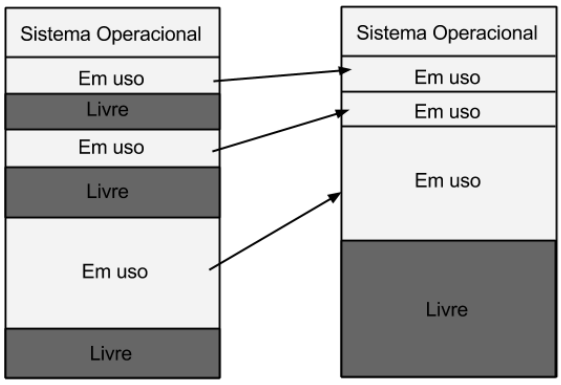

Figura 4: Compactação de memória [4].

sejam alocados e desalocados, partições pequenas e vazias fiquem entre uma partição ocupada e outra. Se um processo grande precisar ser carregado, ele pode ser impedido de executar, apesar do tamanho da soma das partições livres serem maior que o seu tamanho. No esquema de partições dinâmicas o processo ainda precisa estar em espaço contíguo e precisa estar na memória até o fim de sua execução [3, 4, 5].

Uma forma de resolver o problema da fragmentação externa é aplicar uma técnica chamada de compactação. Nessa técnica, o gerenciador realoca todos os espaços de memória livre em uma das extremidades da memória, criando assim uma grande partição (Figura 4). Essa técnica gera uma certa sobrecarga para o sistema, além disso o sistema precisa parar todos os processos para poder reorganizar a memória e isso pode gerar tempos de respostas maiores para os programas [4, 5, 1].

Quando um processo terminar sua execução, será necessário desalocá-lo, pois outros processos podem estar aguardando por alguma partição livre. Se o esquema for de partição fixa, a tarefa é simples, basta atualizar o status da partição para livre ou desocupada. Se o esquema for de partição dinâmica, a tarefa pode ser um pouco mais complexa. Podem acontecer três casos diferentes: um bloco pode ser desalocado adjacente a outro bloco livre; pode ser desalocado entre dois blocos livres; pode ser desalocado isolado de outros blocos livres. Caso o bloco desalocado esteja próximo a algum outro bloco livre, esses blocos deveram se tornar somente um. Se não houver blocos livres adjacentes, simplesmente aparecerá um novo bloco livre entre dois ocupados [5, 7].

Independente se o esquema é de partições fixas ou dinâmicas, o sistema operacional precisa manter uma lista que guarda quais partições estão desocupadas. Quando um novo processo precisar ser carregado, o gerenciador precisa colocá-lo em alguma partição livre. Existem alguns algoritmos para percorrer a lista em busca de uma partição que possua tamanho suficiente para alocar o processo [5].

1. *First-Fit*: nessa estratégia, o gerenciador utiliza a primeira partição encontrada que caiba o processo. O gerenciador pode alocar o processo rapidamente pois precisa procurar o mínimo possível (Figura 5) [3, 4, 5, 7, 1, 2];
2. *Best-Fit*: nessa estratégia, o processo é alocado no melhor espaço encontrado, ou seja, naquele que resulte na menor fragmentação interna. Há

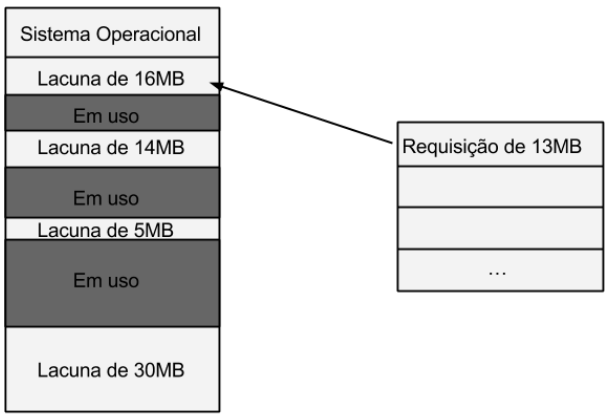

Figura 5: First Fit [4].

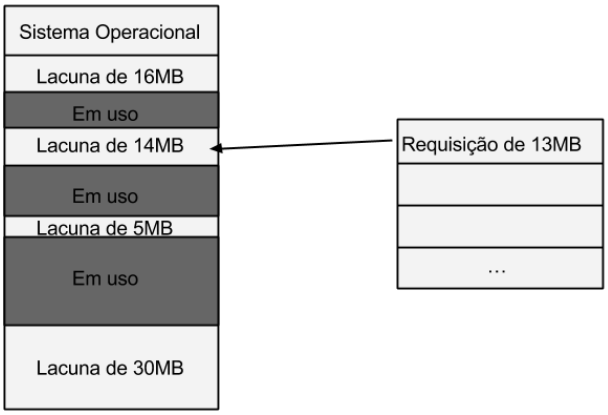

Figura 6: Best Fit [4].

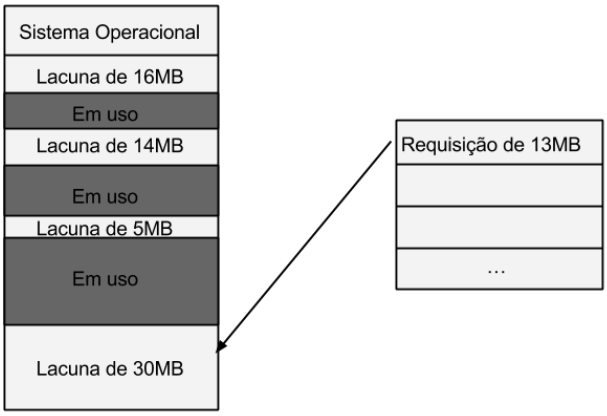

Figura 7: Worst Fit [4].

uma sobrecarga para realizar essa tarefa pois o gerenciador precisa antes de alocar o processo pesquisar pelo melhor espaço. O problema dessa estratégia é que conforme os processos forem alocados, muitas sobras pequenas serão geradas. Se o esquema utilizado for o de partição variável será gerada muita fragmentação externa (Figura 6) [3, 4, 5, 7, 1, 2];

3. *Worst-Fit*: nessa estratégia o processo é alocado na partição que resulte na maior fragmentação interna. O *worst-fit* possui a mesma sobrecarga da estratégia anterior. A ideia de alocar na partição que resulte na maior fragmentação interna, é justamente o fato da fragmentação gerada poder alocar um outro processo, diferentemente do problema da estratégia anterior (Figura 7) [3, 4, 7, 1, 2];
4. *Next-Fit*: essa estratégia é semelhante ao *first-fit*, mas ao invés de iniciar a busca sempre do início da lista, ele guarda a posição da última partição alocada e continua a busca de onde parou. Essa estratégia também pode ser chamada de *circular-fit* [3, 7, 2].

A memória sozinha não tem capacidade para suportar todos os programas. Se um processo precisar executar e todas as partições da memória estiverem ocupadas, ele só poderá executar depois que algum processo termine a execução e se a partição desocupada tenha tamanho suficiente para alocar o novo processo. Para solucionar esse problema foi criada a técnica de *swapping*. Nessa técnica, um processo pode ser transferido temporariamente para o disco, dando lugar para outro processo executar, posteriormente o processo que está no disco volta para a memória e pode terminar a sua execução [3, 4, 7, 1, 2].

A ação de transferência do processo da memória para o disco é chamada de *swap out*, a ação de transferência de volta para a memória é chamada de *swap in*. A sobrecarga utilizada para fazer essa transferência é muito alta, já que o processo inteiro é movido para o disco e depois novamente para a memória. É necessário deixar o processo um tempo razoável no disco para justificar tal operação [7].

5. Memória Virtual

Apesar das diversas técnicas citadas anteriormente otimizarem o uso da memória principal, alguns problemas ainda permanecem. Um processo precisa estar em um espaço contíguo para poder executar e precisa estar em sua totalidade carregado na memória. Outro problema é o número de processos executando em paralelo, que dependerá do tamanho da memória e do tamanho dos processos. Os processos que podem ser executados possuem um limite de tamanho, que é o tamanho da memória. Um processo que exceda esse tamanho nunca poderá ser executado. Uma primeira solução para esses problemas seria aumentar o tamanho da memória. Mas como já foi dito, o tamanho dos processos vem aumentando numa taxa muito maior que a memória física [7, 2].

A técnica de memória virtual surge para solucionar estes problemas. Sistemas baseados em memória virtual dão aos processos a ilusão de que o computador possui mais memória do que a que ele de

fato possui. Pois o gerenciador disponibiliza como espaço de endereçamento tanto a memória principal quanto a memória secundária (disco). Com isso, cria-se a ilusão de que o processo está o tempo inteiro carregado na memória principal [4, 5].

5.1. Endereçamento

Em sistemas que utilizam memória virtual existem dois tipos de endereços: os endereços que os processos referenciam são os endereços virtuais e os endereços disponíveis na memória física são os endereços reais. Quando um processo referencia um endereço virtual, esse endereço precisa ser traduzido para um endereço real. O programador não precisa se preocupar se partes do processo está na memória principal ou secundária, ele só precisa referenciar o espaço de endereçamento virtual do processo e o gerenciador se encarregará de mapear esse endereço para sua localização correta. O programador também não precisa saber qual o tamanho da memória física, o que será disponibilizado para ele será somente o espaço de endereçamento virtual [4, 1].

O espaço de endereçamento virtual é a faixa de endereços virtuais que o processo pode referenciar. Esse espaço de endereçamento virtual normalmente é maior que o espaço de endereçamento físico, que é a faixa de endereços reais disponíveis do sistema. Para o sistema disponibilizar um espaço de endereçamento maior do que a que ele de fato possui, ele precisa fornecer um meio de armazenamento secundário, que normalmente é o disco rígido, para guardar os dados que não cabem na memória principal, já que a mesma é limitada [4, 2].

Quando um processo está executando ou está pronto para executar, os dados que são necessários serão carregados do armazenamento secundário para a memória principal. O processo referencia um enderço virtual e esse endereço é traduzido para um endereço físico. O mapeamento do endereço virtual para o endereço físico é feito enquanto o processo executa e deve ser feito rapidamente. Para que isso seja possível os sistemas utilizam um hardware de propósito especial que é chamado de MMU (Unidade de Gerenciamento de Memória), caso contrário o processador perderia muito tempo fazendo esse mapeamento resultando na perda de desempenho do sistema operacional [4, 7].

Os mecanismos de tradução dinâmica de endereços devem manter a informação de quais regiões do espaço de endereçamento virtual de um processo estão na memória principal e qual a sua localização. Se cada entrada de endereço do espaço de armazenamento do processo fosse carregado na memória, não haveria espaço suficiente para fazer o mapeamento de todos os processos. A solução mais usada é fazer o mapeamento em blocos, onde as informações são agrupadas em blocos e o sistema necessita mapear apenas esses blocos. Quanto maior for o bloco, menor será a quantidade de entradas no mapeamento. Os blocos podem ter tamanhos fixos ou variados. Os blocos de tamanho fixo são chamados de páginas e a organização de memória virtual associada de paginação. Os blocos de tamanhos variados são chamados de segmentos e a organização de memória virtual associada de segmentação [4, 2].

Em um sistema de memória virtual com mapeamento de bloco, o sistema representa os endereços como pares ordenados, onde dado par ordenado $v = (b, d)$, b é o número do bloco em que o item referenciado reside e d é o deslocamento em relação ao início do bloco. O sistema mantém uma tabela de mapas de blocos, essa tabela possui uma entrada para cada processo e essas entradas são mantidas em ordem sequencial. Quando o processo referencia uma página, o sistema pega o endereço de memória da tabela de mapas de blocos do processo e soma com o número do bloco, esse local contém o entrada do início do processo na memória principal, o valor de deslocamento é então adicionado a esse endereço para se chegar no endereço real correspondente ao endereço virtual da página referenciada (Figura 8)[4].

Existem várias técnicas para a tradução dinâmica de endereços, a técnica descrita anteriormente é a tradução de endereço de paginação por mapeamento direto. Outra forma de tradução dinâmica é através de páginas multiníveis. Com a tradução por mapeamento direto, as tabelas de páginas precisam estar todas carregadas na memória sequenciamente, essas tabelas podem ocupar uma grande quantidade de memória e isso pode limitar o uso da memória para os processos. O esquema de páginas multiníveis permite carregar na memória somente as tabelas de

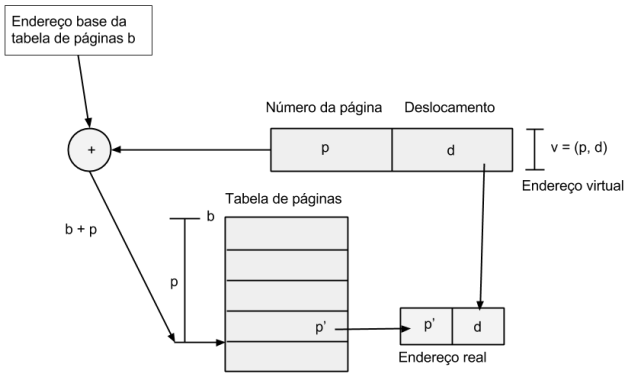

Figura 8: Tradução de endereço de paginação por mapeamento direto [4].

páginas que forem mais utilizadas. As tabelas de páginas são criadas sob uma hierarquia, cada nível possui um ponteiro para uma tabela de nível mais baixo, a tabela de nível mais baixo possuirá o ponteiro para a página [4].

A implementação da memória virtual não é trivial e deve-se tomar todo o cuidado para o desempenho do sistema não piorar ao invés de melhorar. Normalmente os sistemas utilizam mais a paginação do que a segmentação, pois a implementação da segmentação é muito mais complexa pelo fato dos segmentos possuírem tamanhos variados. O uso da paginação simplifica bastante o gerenciamento da memória virtual e torna possível se obter as vantagens propostas pela mesma sem grande perda de desempenho do sistema. Nesse texto somente será abordada a organização de memória virtual com paginação [7, 1].

5.2. Paginação

A paginação permite que o processo seja alocado em um espaço não-contíguo. A memória não precisa mais possuir partições de diversos tamanhos, pois as páginas são de tamanho fixo. O bloco da memória principal que guarda a página é denominado moldura de página e possui o tamanho exato da página. Quando um processo está pronto para carregar, as páginas que são necessárias são carregadas em qualquer espaço de memória que esteja disponível. Nesse tipo de esquema não existe fragmentação externa, mas pode ocorrer fragmentação interna, nesse caso a fragmentação será reduzida [1].

É preciso saber de alguma forma se determinada página está carregada na memória principal ou se está no disco. Uma solução empregada é o uso do bit válido-inválido. O gerenciador mantém uma tabela de páginas que mapeia todas as páginas, bem como se ela está carregada na memória principal ou não (Figura 9) [1].

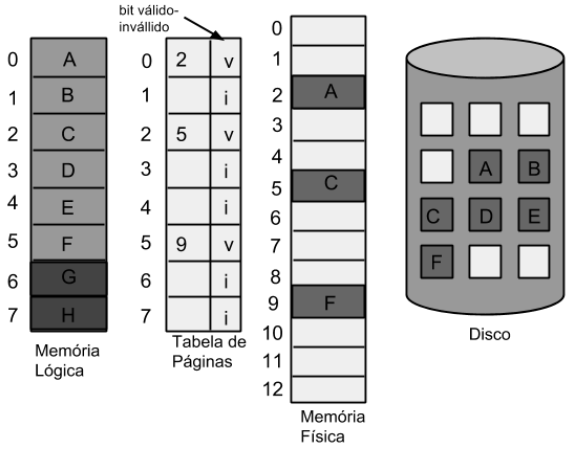

Figura 9: Tabela de página quando algumas páginas não estão na memória principal [1].

Existem dois tipos de estratégias para paginação: a paginação por demanda e a paginação antecipada. Na paginação por demanda, é utilizada a busca por demanda onde o gerenciador espera que o processo referencie a página para então carregá-la na memória principal. Na paginação antecipada, o gerenciador tenta prevê quais páginas o processo precisará e tenta carregá-las na memória antecipadamente [4, 7].

Nas diversas heurísticas utilizadas no processo de paginação e substituição de páginas, um conceito que é amplamente usado é o de localidade. A localidade pode ser manifestada no tempo e no espaço. Segundo [4] “a localidade temporal é a localidade ao longo do tempo. Por exemplo, se o dia estiver ensolarado em uma certa cidade às 14h, haverá uma boa chance (mas certamente nenhuma garantia) de que o dia naquela cidade esteja ensolarado às 14h30 e continuará às 15h30. Localidade espacial significa que itens próximos tendem a ser semelhantes. Por exemplo, se o sol estiver brilhando em uma cidade, será bem provável, mas não garantido, que também estará brilhando em cidades próximas”[4, 7].

Comparando paginação por demanda com a paginação antecipada, podemos observar que na paginação por demanda, apenas as páginas que

de fato serão utilizadas pelos processos serão trazidas para a memória principal. Essas páginas serão carregadas uma por vez, conforme os processos as necessitam, esse fator aumenta o grau de multiprogramação, já que mais processos poderão ser carregados na memória principal. O problema é que o tempo que o processo espera por páginas que não estão carregadas pode ser muito caro [4].

Fazendo o cálculo do produto espaço-tempo pode-se observar que na paginação por demanda, o processo gasta mais tempo esperando por páginas que não estão na memória principal do que executando algo produtivo. Segundo [4], "o produto espaço-tempo de um processo, é uma medida de seu tempo de execução multiplicado pela quantidade de espaço da memória principal que o processo ocupa. O produto espaço-tempo ilustra não somente quanto tempo um processo gasta esperando, mas quanta memória principal não pode ser usada enquanto ele espera"(Figura 10) [4].

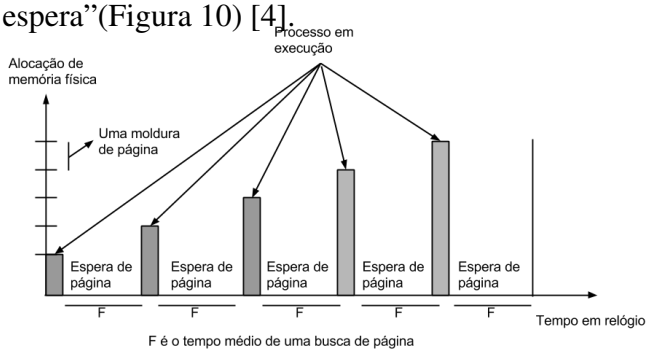

Figura 10: Produto espaço-tempo sob paginação por demanda [4].

Na paginação antecipada tenta-se reduzir o tempo em que o processo executa ou fica ocioso na memória eliminando o tempo de espera por páginas, carregando essas páginas antes que o processo as referencie. Segundo [4], "os critérios que determinam o sucesso de uma paginação antecipada são:

1. alocação pré-paginada: a quantidade de memória principal alocada à pré-paginação;
2. o número de páginas carregadas previamente de uma só vez;
3. a política: a heurística que determina quais páginas são pré-carregadas."

É preciso de uma boa heurística para carregar de uma só vez na memória um certo número de páginas que o processo possivelmente irá referenciar. O tempo em que um processo fica ocioso na memória de fato diminui, pois o número de E/S para o disco diminui. Entretanto, se for utilizada uma heurística inadequada, o sistema pode apresentar um desempenho pior do que um sistema com paginação por demanda, pois as páginas que serão carregadas antecipadamente poderão não ser as próximas páginas referenciadas pelo processo. O número de páginas que serão carregadas de uma só vez também é um fator muito importante, pois quanto mais páginas forem carregadas, menos processos poderão ser executados em paralelo [4].

Como várias páginas são carregadas de uma só vez, existe o tempo maior de espera para que essas páginas sejam carregadas do armazenamento secundário. Normalmente as técnicas de paginação antecipada carregam páginas que são contíguas no espaço de endereçamento virtual. O fato dessas páginas serem contíguas no espaço de endereçamento virtual, não significa que elas são contíguas no disco. Portanto, haverá uma sobrecarga no carregamento dessas páginas [4].

O tempo que o processo espera na paginação antecipada e na paginação por demanda são quase os mesmos. Porém a paginação antecipada pode apresentar resultados melhores que a paginação por demanda, tudo vai depender dos critérios citados anteriormente para um bom desempenho desse tipo de paginação. Normalmente a estratégia de paginação antecipada é combinada com a paginação por demanda [4].

5.3. Substituição de páginas

Quando um processo referencia uma página, pode ser que todas as molduras de páginas na memória principal já estejam ocupadas. A memória principal não é utilizada apenas para carregar dados e instruções de programas, outro uso dela, por exemplo, são os buffers de operações de *Input/Output*, que por sinal ocupam uma considerável quantidade de memória. O evento em que um processo referencia uma página que não está carregada na memória é chamado de *page-fault* (falta de página) [1].

Quando isso ocorre, o gerenciador de memória deverá transferir alguma página carregada na memória para o disco e então carregar a página referenciada.

Esse processo é chamado de substituição de páginas. O algoritmo a seguir descreve como esse processo funciona [1, 5]:

1. Localizar a posição da página desejada no disco.
2. Encontrar um quadro livre.
   (a) Se houver um quadro livre, use-o.
   (b) Se não houver quadros livres, use um algoritmo de substituição de página para selecionar um quadro vítima.
   (c) Gravar a página vítima no disco; alteras as tabelas de página e quadro de acordo.
3. Ler a página vítima no disco; alteras as tabelas de páginas e quadro de acordo.
4. Retornar o processo de usuário.

Para realizar a substituição de páginas são necessárias duas transferências e isso aumenta o tempo de resposta do programa. Uma solução para esse problema é utilizar um bit de modificação. Se a página sofrer algum tipo de alteração esse bit será ativado. No momento da substituição de uma página, será verificado através do bit de modificação se a página sofreu alguma alteração. Caso ela não tenha tido alterações, ela poderá ser removida da memória, pois ela já está armazenada no disco. Com esse artifício conseguimos reduzir a sobrecarga dessa operação, visto que a página só será copiada da memória para o disco caso ela tenha sido modificada [1].

Para substituir uma página o sistema primeiro deve decidir qual página será substituída. Existem diversas estratégias de substituição. Para analisar o desempenho dessas estratégias elas são comparadas com a estratégia de substituição ótima. Essa estratégia diz que a página ideal a ser substituída é aquela que não será referenciada no futuro mais distante possível. A estratégia ótima serve apenas como parâmetro para analisar as outras estratégias. É impossível implementar o algoritmo ótimo, pois não tem como se prever o comportamento dos processos, ou seja, não tem como prever qual será a ordem exata em que ele referenciará as páginas e nem quais processos serão executados no futuro. As estratégias de substituição de página devem balancear a redução de falta de páginas com a sobrecarga gerada para o algoritmo funcionar [4, 1].

Pelo fato da memória ser utilizada para carregar dados e instruções dos processos e os buffers de Input/Output, os algoritmos de substituição de páginas podem aumentar a carga de execução. Alguns sistemas possuem porcentagem fixa de armazenamento para buffers de Input/Output e processos. Outros permitem que ambos concorram por todo espaço de armazenamento da memória [1].

5.4. Estratégias de substituição de páginas

Os principais algoritmos de substituição de páginas são enumerados a seguir.

5.4.1. Substituição Aleatória de Páginas

Nessa estratégia, a página que será substituída é escolhida aleatoriamente, portanto todas as páginas têm a mesma probabilidade de serem substituídas. Essa estratégia é de fácil implementação e possui baixa sobrecarga no processamento. Seu problema é que a página que for selecionada poderá ser justamente a próxima página que será referenciada. Esse caso dificilmente irá acontecer, devido ao grande número de molduras de páginas presente na memória, mas ainda assim essa estratégia não é utilizada por causa de sua abordagem da tentativa e erro [4].

5.4.2. FIFO - First In, First Out

Nessa estratégia a página que estiver a mais tempo na memória será aquela que será substituída. Esse algoritmo é implementado utilizando uma fila. A Figura 11 mostra como esse processo funciona. A tabela da esquerda mostra a sequência que as páginas foram referenciadas. A tabela da direita mostra o estado da fila toda vez que uma página é referenciada. A página que estiver no início da fila será substituída. O problema dessa estratégia é que páginas que são intensamente utilizadas também serão substituídas, o fato delas estarem a mais tempo na memória não significa que não serão referenciadas em seguida. Nesse caso a FIFO irá resultar em mais faltas de páginas. Essa estratégia não é comumente utilizada [3, 4, 5, 7, 1, 2].

5.4.3. MRU - Menos Recentemente Usada

Nessa estratégia, a página que passou mais tempo na memória principal sem ser referenciada será a

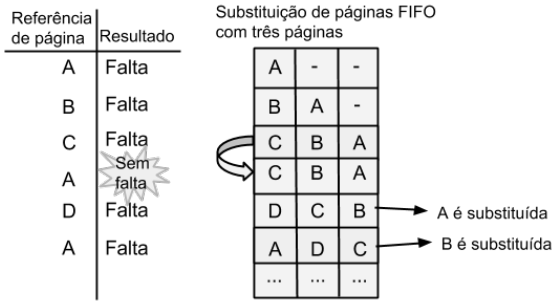

Figura 11: Substituição de páginas FIFO [4].

próxima página a ser substituída. Essa estratégia se baseia no princípio da localidade temporal onde, se a página não foi referenciada nesse momento, muito provavelmente ela continuará sem ser referenciada em um determinado espaço de tempo futuro. Essa abordagem é falha, pois apesar de uma determinada página ser a menos referencida, ela pode ser a próxima página a ser referencida logo depois de ser substituída. A MRU pode ser implementada utilizando uma lista, se uma página for referenciada, essa página deverá ser transferida para o final da lista (Figura 12). A MRU pode ter um resultado melhor que a FIFO, mas possui uma sobrecarga para manter quais são as páginas que são menos recentemente usadas [3, 4, 5, 7, 1, 2].

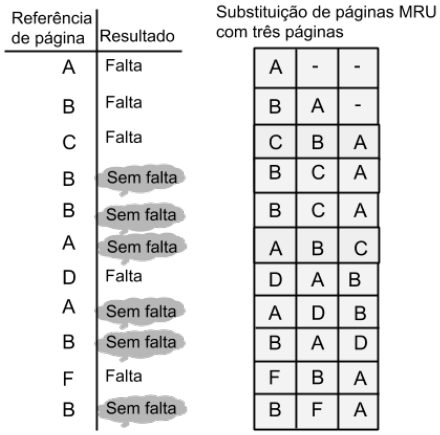

Figura 12: Estratégia de substituição de página "menos recentemente usada"(MRU) [4].

5.4.4. MFU - Menos Frequentemente Usada

Nessa estratégia a página que será substituída será aquela que foi menos frequentemente referenciada. Essa estratégia se baseia na heurística de que se a página não é frequentemente referenciada, significa que ela continuará não sendo referenciada num futuro próximo. Essa estratégia pode ser implementada utilizando um contador para cada página. Toda vez que uma página for referenciada seu contador deverá ser incrementado. Semelhante a MRU essa estratégia também é falha, pois a página menos frequentemente referenciada pode ser a próxima a ser referenciada. A MFU possui uma sobrecarga parecida com a da MRU, pois também precisa fazer uma atualização sempre que uma página for referenciada. Mas por causa da forma de implementação, pode possuir uma sobrecarga menor, já que enquanto ela atualiza o valor de um contador, a MRU precisar mudar a posição da página referenciada [3, 4].

5.4.5. NUR - Não Usada Recentemente

Semelhante a MRU, porém implementada de forma diferente e com pouca sobrecarga. Cada entrada de tabela de página possui dois bits de hardware: bit referenciado e bit modificado. Se a página não foi referenciada o bit referenciado será 0 e se a página foi referenciada o bit referenciado será 1; se a página não foi modificada o bit modificado será 0 e se a página não foi modificada o bit modificado será 1.

Essa estratégia divide as páginas em quatro grupos (Figura 13). O grupo 1 são as páginas que não foram modificadas e nem foram referenciadas; o grupo 2 são as páginas que não foram referenciadas, mas foram modificadas (esse caso ocorre quando o sistema atualiza os bits referenciados para 0, isso deve acontecer periodicamente, caso contrário chegará um momento em que todas as tabelas terão seus bits referenciados como 1); o grupo 3 são as páginas que foram referenciadas, mas não foram modificadas; o grupo 4 são as páginas que foram referenciadas e foram modificadas. Os grupos com menor número serão escolhidos primeiro que os de maior número, as tabelas que fizerem parte de um mesmo grupo serão escolhidas aleatoriamente [4, 2].

5.4.6. Modificação FIFO - "Segunda Chance"

Como na estratégia FIFO pode ocorrer da próxima página a ser substituída ser a próxima página a ser referenciada, nessa estratégia cada página possuirá um

Grupo	Referida	Modificada	Descrição
Grupo 1	0	0	Melhor opção para substituir
Grupo 2	0	1	
Grupo 3	1	0	
Grupo 4	1	1	Pior opção para substituir

Figura 13: Tipos de páginas sob NUR [4].

bit referenciado, onde 0 significa que a página não foi refereniada e 1 significa que a página foi referenciada. Quando chegar a vez da página ser substituída, se o bit dela for 1, significa que a página foi referenciada recentemente, então, ao invés de ser substituída a página será movida para o final da fila e o bit será modificado para 0. Se ao chegar no início da fila o bit for 0, será verificado se a página foi modificada, caso tenha sido, ela também será movida para o final da fila e ela não poderá ser substituída enquanto não for copiada para o armazenamento secundário. Isso é feito para evitar um dispendiosa operação para copiar a página para o disco no momento da substituição. Caso a página chegue no início da fila com o bit 0 e não foi modificada ela será substituída naturalmente [3, 4, 7, 1, 2].

5.4.7. Modificação FIFO - "Relógio"

Semelhante ao FIFO sengunda chance e produz essencialmente os mesmos resultados. Nessa estratégia é utilizada uma lista circular ao invés de uma linear. Um ponteiro de lista indicará qual página deverá ser substituída, se o bit referenciado for 1 ou se a página estiver esperando para ser atualizada no armazenamento secundário, o ponteiro passará para o próximo elemento (Figura 14). Essa estratégia possui menor sobrecarga que o FIFO segunda chance, pois enquanto este precisa remover e adicionar a página na fila toda vez que a página não puder ser substituída, aquele precisa somente mudar o valor de um ponteiro [3, 4, 2].

5.4.8. Substituição de Página Longínqua

Nessa estratégia, será gerado um grafo que caracteriza a ordem que o processo irá referenciar as páginas. Esse grafo é gerado com base em padrões previsíveis que os programas tendem a referenciar funções e dados. Cada nó do grafo possuirá um bit referenciado, a próxima página a ser substituída

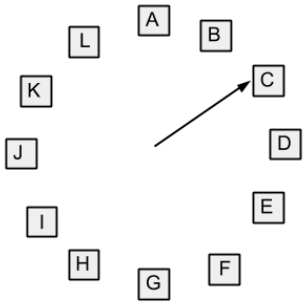

Figura 14: Algoritmo de substituição de página relógio [2].

será aquela que tiver o bit referenciado como 0 e estiver mais distante no grafo da última página referenciada. A Figura 15 mostra um exemplo desse grafo. Cada vértice do grafo de acesso representa uma das páginas do processo. Os vértices significam que o processo pode, por exemplo, referenciar a página E depois de referenciar a página B e não pode referenciar a página L antes de referenciar a página G. O grafo mostra como o processo pode referenciar as páginas durante sua execução. Conforme as páginas forem referenciadas, o sistema vai marcando as mesmas. A página que estiver mais longe da ultima página referenciada no grafo, muito provavelmente será a página a ser referenciada no futuro mais distante. Essa estratégia consegue resultados quase ótimos, mas devido a complexidade do algoritmo para gerar o grafo e da sobrecarga do tempo de execução devido também ao grafo, essa estratégia não é utilizada em sistemas reais [4].

5.4.9. Modelo de Conjunto de Trabalho

"Localidade de referência implica que um programa pode executar eficientemente mesmo que apenas um subconjunto relativamente pequeno de suas páginas resida na memória principal em qualquer dado instante"[4]. A teoria do conjunto de trabalho para comportamento do programa de Denning tenta descobrir justamente qual é esse subconjunto do processo para mantê-lo na memória principal. Quanto mais molduras de páginas o processo possuir, menor será o número de falta de páginas; quanto menos molduras de páginas o processo possuir, maior será o

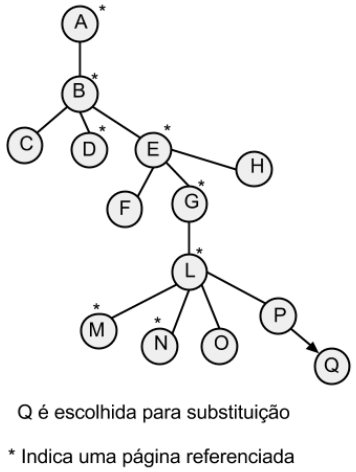

Figura 15: Grafo de acesso da estratégia de substituição de página longínqua [4].

número de falta de páginas. A teoria do conjunto de trabalho para o comportamento do programa de Denning diz que um programa funciona eficientemente bem, ou seja, possui um número de falta de páginas razoável e estável, se um determinado subconjunto de páginas do processo permanecer na memória, essas páginas são as páginas necessárias para o programa executar bem. O conjunto de trabalho de um processo muda conforme o processo vai executando. Se as páginas que são necessárias para o processo executar forem substituídas, o sistema apresentará uma grande quantidade de paginação. [3, 4, 5, 2].

5.4.10. FFP - Substituição de Página por Frequência de Falta de Página

A taxa de falta de páginas de um processo mostra se ele está executando bem ou não. A FFP também baseia-se em carregar na memória principal um conjunto de páginas. O conjunto de páginas que ficará residente na memória principal é escolhido com base na frequência de falta de páginas do processo. Processos que apresentam muitas falta de páginas possuem poucas molduras de páginas, enquanto processos que apresentam poucas falta de páginas possuem muitas molduras de páginas e pode estar impedindo outros processos de executarem adequadamente ou diminuindo o grau de multiprogramação. O conjunto de trabalho do processo deve possibilitar a execução do processo no intermediário desses dois casos. Ao ocorrer uma falta de página, o gerenciador calculará quanto tempo se passou após a útlima falta de página. Se esse tempo for maior que um determinado tempo máximo estabelecido, as páginas não referenciadas do conjunto de trabalho do processo nesse intervalo de tempo serão descartadas. Se o tempo for menor que o tempo mínimo estabelecido a página será adicionada ao conjunto de trabalho. A FFP calcula o conjunto de trabalho a cada falta de página e ajusta-o dinamicamente conforme o processo vai executando [3, 4].

6. Trabalhos Correlatos

Esta seção apresenta alguns sitemas que simulam o processo de gerência de memória virtual ou a gerência de memória simples.

6.1. SOSIM

O SOSIM é uma ferramenta que foi criada para auxiliar professores e alunos nas disciplinas que abordam conceitos de sistemas operacionais. Esse sistema permite simular os conceitos implementados em sistemas operacionais multiprogramados. Através da ferramenta é possível visualizar de forma animada a gerência de processos e a gerência de memória virtual. Sobre gerência de memória virtual, o sistema permite [10]:

1. escolher a política de busca de página, se é por demanda ou antecipada;
2. visualizar as páginas que estão na memória principal e na memória secundária;
3. visualizar as páginas sendo trocadas entre a memória principal e a memória secundária conforme os processos vão executando;
4. visualizar uma janela com dados estatísticos, um deles mostra a quantidade de falta de páginas desde a vinda do primeiro processo para execução;
5. visualizar quantos por cento da memória física está livre.

O SOSIM é capaz de auxliar no entendimento de diversos conceitos, no entanto, o sistema em si pode se apresentar confuso. Várias janelas são exibidas e executam ações ao mesmo tempo, o usuário pode ficar perdido sem saber para onde olhar. A forma

como a gerência de memória virtual é tratada pode ser aprofundada e aprimorada. O usuário é capaz de ver que páginas estão o tempo todo sendo trocadas entre a memória e o disco, mas não tem como entender porque isso está acontecendo, nem como a memória virtual foi implementada. O sistema só permite simular o algoritmo de substituição de páginas FIFO. Uma *screenshot* de algumas janelas do sosim pode ser visualizada na Figura 16.

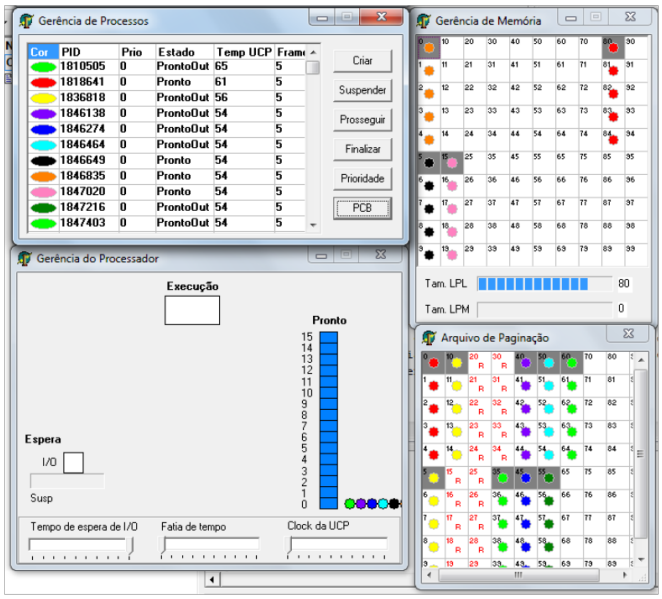

Figura 16: Sosim.

Através do sistema é possível visualizar em que ordem as páginas são substituídas de acordo com os algoritmos disponíveis e quantas faltas de páginas ocorreram. Com isso já é possível fazer uma boa análise dos algoritmos. Poder executar dois algoritmos em paralelo também é bastante interessante, pois podemos observar durante a execução a diferença no comportamento da substituição de ambos. Esse sistema auxilia no entendimento, mas é bastante simples e limitado.

Entretanto, torna-se complexo o entendimento do funcionamento da memória virtual como um todo, sendo possível apenas entender a lógica dos algoritmos demonstrados. Não é possível identificar como esses algoritmos foram implementados, nem visualizar detalhes de todo esse mecanismo. *Screenshots* do sistema podem ser visualizadas nas Figuras 17 e 18.

Figura 17: SimulaRSO.

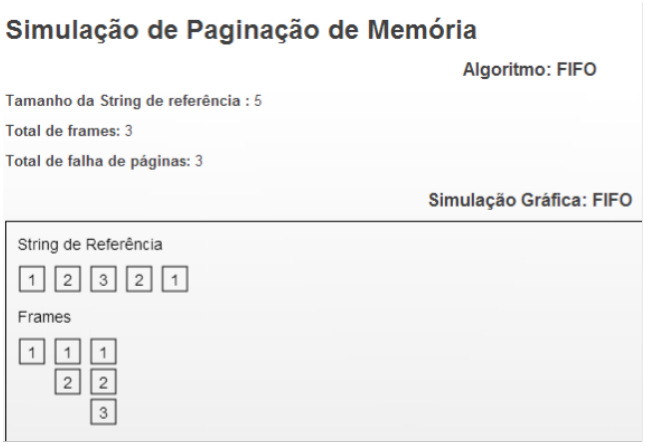

Figura 18: SimulaRSO.

6.2. *SimulaRSO*

Esse sistema simula o escalonamento de processos, escalonamento de discos e a paginação de memória. Na simulação de paginação de memória o sistema permite [11]:

1. escolher entre as estratégias de substituição FIFO, algoritmo ótimo e a menos recentemente utilizada;
2. escolher o número máximo de molduras de páginas na memória;
3. escolher quantas páginas serão referenciadas e qual a sequência que elas serão referenciadas;
4. comparar a execução de dois algoritmos;
5. visualizar graficamente o estado das molduras de páginas conforme as páginas vão sendo referenciadas.

6.3. *SSOG*

O SSOG é um software que foi proposto para auxiliar nas aulas de Sistemas Operacionais. Esse

software simula alguns conceitos como gerência de processos, gerência de processador e gerência de memória. No esquema de memória utilizado pelo software, não é utilizado o conceito de memória virtual. A memória é dividida em 100 partições, as 20 primeiras são reservadas ao sistema operacional. Quando o usuário cria um processo, ele pode escolher quantos frames esse processo possuirá, para o sistema poder alocar na memória a quantidade adequada. É possível também visualizar a porcentagem de memóra que está sendo utilizada e a quantidade de processos que estão carregados [12]. (Figura 19).

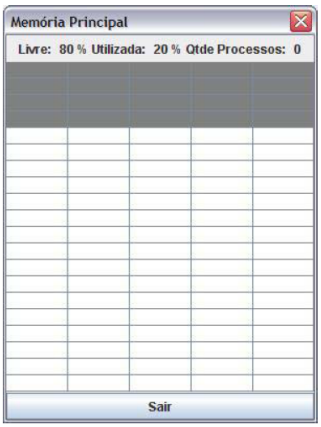

Figura 19: Gerência de Memória no SSOG [12].

6.4. Simulador de Memória Real e Virtual

Esse trabalho propõe simular a gerência de memória real e virtual. O usuário pode inserir ou remover processos de maneira manual ou automática. Esses processos serão alocados na memória com base em algum algoritmo de alocação que pode ser escolhido. Na simulação de memória virtual, o sistema permite a visualização (Figura 20) [13]:

1. conteúdo do programa a ser carregado;
2. memória física;
3. CPU;
4. controle dos processos em execução;
5. tabelas de páginas;
6. disco rígido;
7. log de execução.

O algoritmo de substituição de página implementado é o FIFO. Somente dois processos podem ser

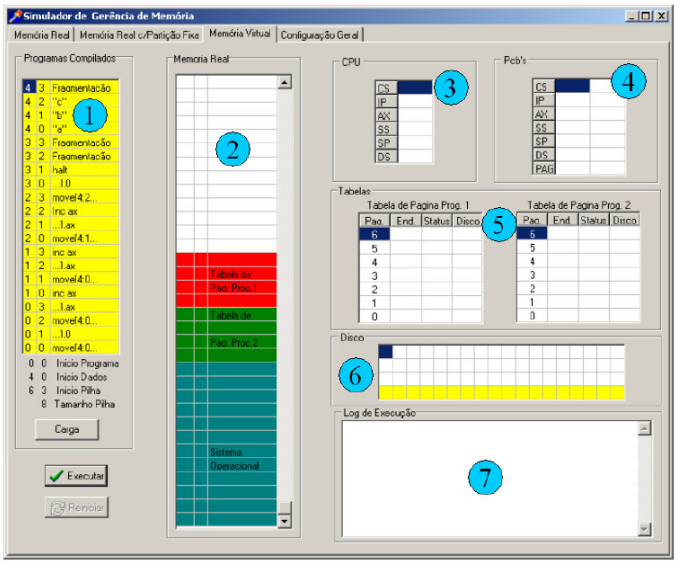

Figura 20: Gerência de Memória Virtual [13]

carregados e antes de carregá-lo é preciso realizar uma configuração de qual será seu identificador e o tempo de execução entre uma instrução e outra. A tabela de página informa os enderços virtuais e reais da página e se ela está na memória física ou no disco. A memória física mostra que está sendo ocupada também pelo sistema operacional e pelas tabelas de páginas dos processos.

7. Modelo Proposto

Na ferramenta desenvolvida, o usuário além de visualizar o funcionamento dos algoritmos de substituição de páginas e as interações que acontecem entre os diversos componentes que compõem esse mecanismo, poderá também implementar seus próprios algoritmos e colocá-los para executar juntamente com a aplicação. A ideia é que depois de estudar o algoritmo o aluno possa por em prática o que ele aprendeu. Normalmente o aluno precisa ter um enorme trabalho para simular esse processo, muitas vezes demora muito mais tempo implementando abstrações do sistema operacional do que se preocupando com a lógica do algoritmo em si. Com o sistema que está sendo proposto, só será preciso se preocupar com a lógica do algoritmo de substituição, pois o sistema já disponibilizará abstrações dos principais recursos necessários

para a gerência de memória virtual, como por exemplo, memória principal, disco e tabela de páginas.

Ao final, o aluno poderá visualizar o que foi feito através de animações. É possível também interagir com o gerenciador simulado, adicionando e removendo processos e principalmente referenciando as páginas. Essa última funcionalidade é um diferencial comparado com outras ferramentas. Nas ferramentas correlatas analisadas, o usuário somente interage com o mecanismo de gerência de memória adicionando e removendo processos. Após adicionar um processo, ele fica apenas observando qual será o comportamento. Essa abordagem é de difícil compreensão, pois o aluno precisa prestar bastante atenção para não perder nenhum detalhe,já que o processo executa e referencia as páginas sem a intervenção do usuário.

Este trabalho propõe uma maior interação entre o usuário e o processo. Além de adicionar o processo, o usuário também vai ditar a ordem com que as páginas vão ser referenciadas em tempo de execução. Isso permite que o usuário tenha um melhor controle sobre o que está acontecendo e consiga criar diversas situações ao mesmo tempo em que já está analisando-as. Os recursos gráficos representam um ponto muito importante para este trabalho, pois ele foi idealizado para ser um sistema com visual agradável.

O ambiente de simulação proposto foi feito considerando a gerência de memória virtual com paginação sob demanda, tendo as páginas com tamanho fixo. O foco do trabalho em si, é somente os algoritmos de subsituição de páginas e as interações entre o gerenciador, memória física e disco quando os processos são carregados e demandam espaço de memória. Por tanto, toda a parte de endereçamento e tradução dinâmica de endereço foi simplificada e não será mostrada na visualização da simulação, sendo sugerida como trabalho futuro.

O sistema é composto por um módulo gerenciador, que possui os principais componentes necessários para prover a simulação da gerência de memória virtual. Outra camada também de fundamental importância é a *view*, que irá representar graficamente tudo o que esta acontecendo no módulo gerenciador. Essa camada view utiliza o componente JavaFX, o qual provê os recursos necessários para que a visualização desejada seja possível (Figura 21).

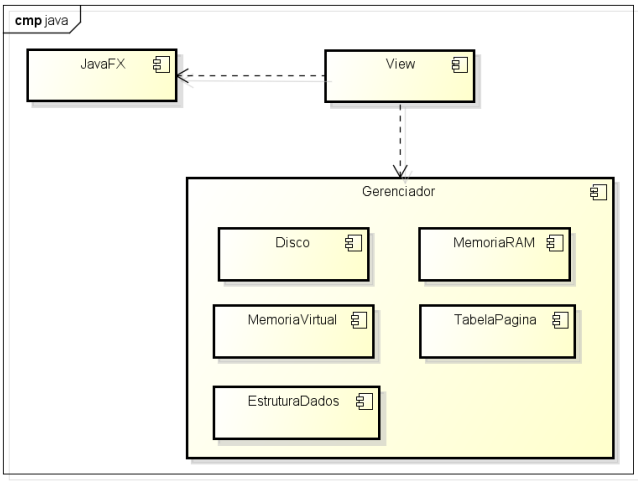

Figura 21: Arquitetura do trabalho.

As tecnologias utilizadas para desenvolver a ferramenta foram basicamente a linguagem Java e a biblioteca JavaFX. O JavaFX permite a criação de ambientes gráficos de maneira simplificada, foi escolhido pois foi a melhor ferramenta para criação de layout de tela encontrada. Utilizando o JavaFX, automaticamente faz-se necessário também utilizar a linguagem Java, visto que o JavaFX é genuinamente código Java.

7.1. Módulo gerenciador

A Figura 22 mostra um diagrama com as principais classes do módulo gerenciador. Esse módulo foi construído de maneira que o seu funcionamento independe da camada *view*. Podemos executar os testes realizando chamadas diretas ao módulo via console. O aluno pode ainda criar uma outra implementação da camada *view* utilizando outros componentes para representar o gerenciador, se assim desejar.

Na Figura 22 podemos observar uma interface chamada *EstruturaDados*. No desenvolvimento do trabalho, foi identificado que o determinante para o funcionamento correto de certa estratégia de subsituição, é a forma como as dados relevantes para a mesma são mantidos. Normalmente, os algoritmos funcionam em cima de alguma estrutura de dados, mais comumente, filas, listas, árvores e matrizes. Essas estruturas normalmente guardam os endereços das páginas e sempre que uma página é referenciada elas são atualizadas. Essa atualização pode ser

somente uma reorganização dos endereços ou pode ser a adição de algum dado que será utilizada depois para determinar qual página será substituída.

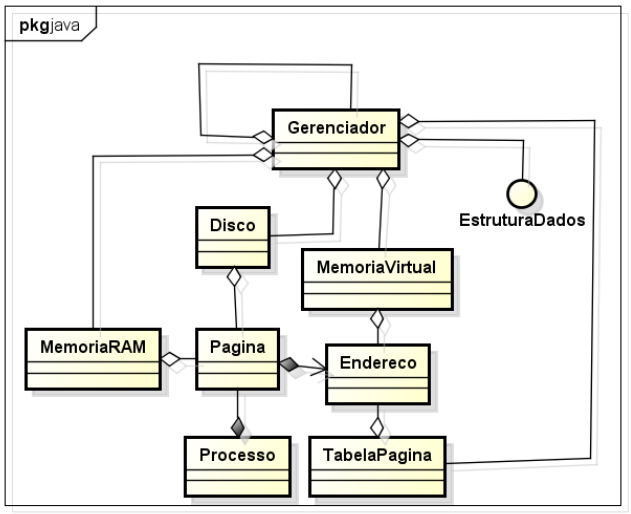

Figura 22: Diagrama de classe do módulo gerenciador.

Normalmente a sequência de passos utilizada pelo gerenciador para substituir uma página é sempre a mesma. Essa sequência pode ser visualizada no diagrama de sequência presente na Figura 23. A página que será retornada no método *paginaParaSubstituir* da *EstruturaDados*, vai depender de qual estratégia de substituição de páginas a estrutura de dados está atendendo. Para entender melhor o que foi proposto, será mostrada a implementação do algoritmo FIFO sobre essa ótica. A interface *EstruturaDados* é definida conforme a Figura 24.

```
public void adicionarReferencia(Endereco endereco);
public void removerReferencia(Endereco endereco);
public Endereco paginaParaSubstituir();
public void removerReferencias(int idProcesso);
public Iterator<Endereco> createIterator();
```

Figura 24: Interface Estrutura Dados.

Para que o algoritmo FIFO funcione (Seção 5.4.2), é necessário que o objeto que implemente a interface *EstruturaDados* possua uma fila. Assim, sempre que uma página for referenciada, o endereço da mesma será adicionado na fila, exceto se esse endereço já tiver sido adicionado. Sempre que o método paginaParaSubstituir for chamado, o primeiro endereço da fila será retornado. Conforme pode ser observado, o bom funcionamento do algoritmo depende inteiramente da implementação da estrutura.

Se um usuário desejar adicionar uma nova estratégia, basta ele implementar essa interface EstruturaDados de maneira que ela mantenha os dados necessários e que retorne o endereço apropriado da próxima página a ser substituída no momento que ele for solicitado. O mesmo fluxo definido no diagrama de sequência da Figura 23 servirá para qualquer algoritmo baseado em paginação sob demanda. O módulo gerenciador também suporta outra implementação desse fluxo, mas na maioria dos casos isso não será necessário.

Para que o software reconheça a nova estratégia implementada será necessário adicioná-la na classe *EstrategiaEnum* mostrada na Figura 25. A lista de estratégias disponibilizadas para o usuário escolher será carregada baseada nesse *enum*. O campo *estrutura* deverá ser o nome da classe que representa essa estrutura e o campo *estrategia* será a implementação da sequência de passos utilizada pelo gerenciador. Essas classes serão instanciadas via reflexão computacional. No caso do campo *estrategia*, normalmente será o mesmo para todas as estruturas.

```
public enum EstrategiaEnum {

    FIFO        ("FIFO", "Fila", "DefaultStrategy"),
    MRU         ("MRU", "ListaMRU", "DefaultStrategy");

    private String nome;
    private String estrutura;
    private String estrategia;
```

Figura 25: EstrategiaEnum.

As interações entre o usuário e o sistema são descritas no diagrama de casos de uso da Figura 26. Os diagramas de sequências para os casos de uso adicionar processo, remover processo e referenciar página podem ser vistos respectivamente nas Figuras 27, 28 e 29.

Para adicionar um processo, o fluxo seguido é bastante simples (Figura 27):

1. o usuário solicita a criação de um processo e diz quantas páginas esse processo possuirá;
2. o método *adicionarProcesso* da classe *Disco*, adiciona as páginas no respectivo objeto, como essas páginas normalmente não estão próximas, a classe *Disco* foi implementada utilizando um

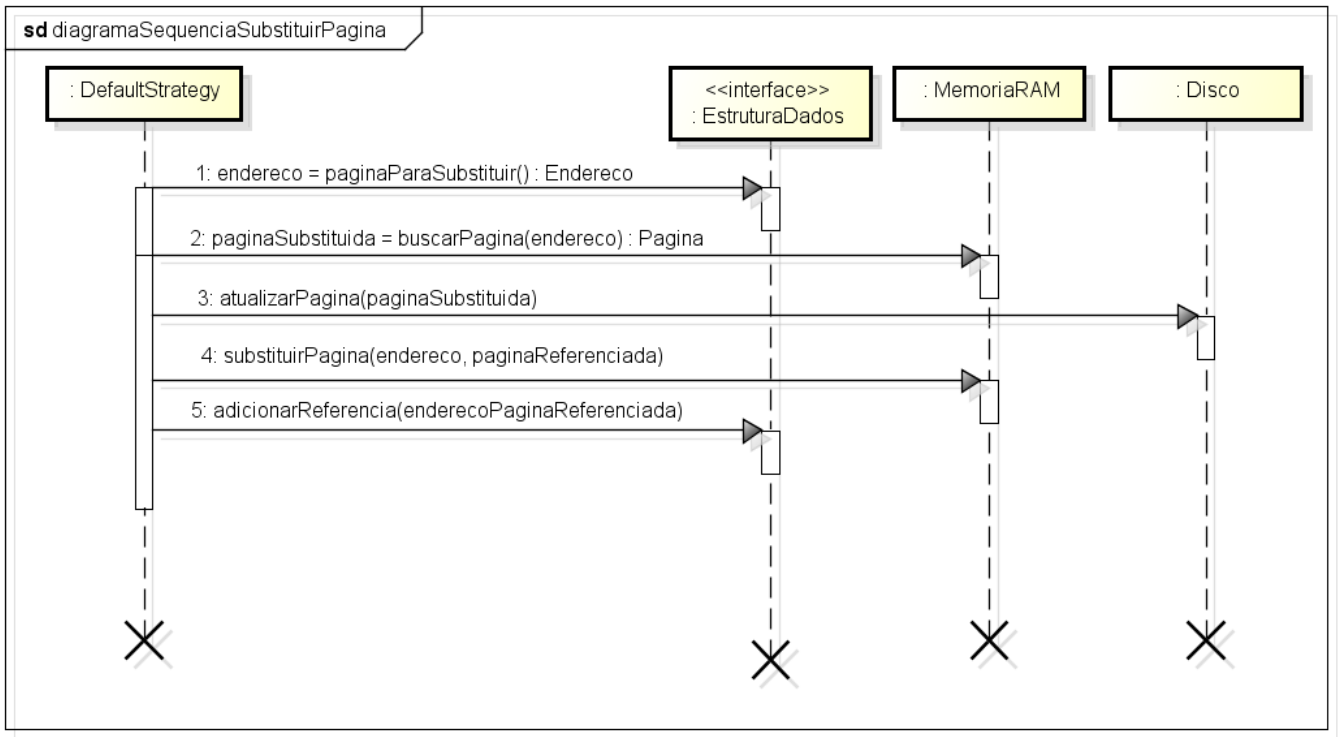

Figura 23: Diagrama de Sequência para o fluxo de substituir página

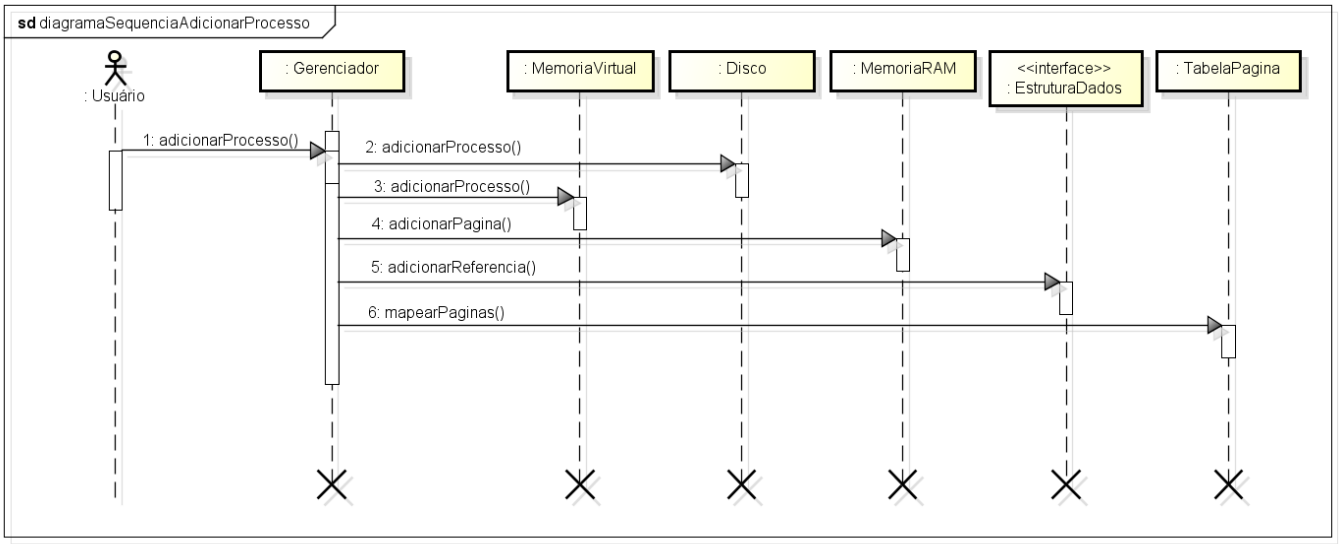

Figura 27: Diagrama de Sequência para o caso de uso adicionar processo

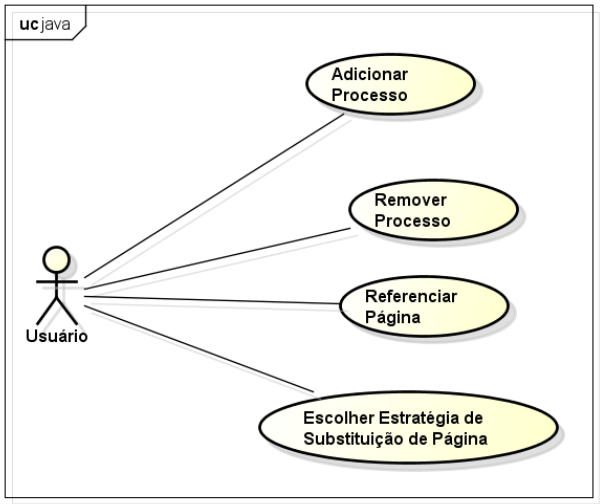

Figura 26: Casos de Uso.

HashMap, de maneira que as páginas não são guardadas contiguamente;

3. o método *adicionarProcesso* da classe *MemoriaVirtual*, adiciona os endereços das páginas do processo no objeto em questão. Essa classe representa o enderaçamento virtual dos processos. Os endereços são guardados nesse objeto de maneira contígua;
4. a primeira página do processo é carregada na memória, essa ação foi realizada para simbolizar o fato de que quando um processo é carregado na memória, as páginas necessárias para que seja realizado o seu "*start*"já são alocadas na memória principal;
5. é adicionado na estrutura de dados a referência da página que foi carregada na memória, para que ela já seja considerada no método de substituição utilizado;
6. o objeto *TabelaPagina* é atualizado com o mapeamento das páginas do novo processo carregado.

O diagrama de sequência da Figura 28 é o inverso do fluxo anterior:

1. o usuário solicita a remoção do processo;
2. o gerenciador solicita a remoção dos endereços das páginas do processo do objeto *MemoriaVirtual*;
3. as páginas em seguida são removidas do *Disco*, se houver páginas carregadas na *MemóriaRAM*, elas também são removidas;
4. os endereços presentes na *EstruturaDados* são removidos para que o algoritmo não os considere mais;
5. o mapeamento feito no objeto *TabelaPagina* também é removido.

O diagrama da Figura 29 mostra o fluxo seguido para referenciar uma página:

1. o usuário executa a ação de referenciar a página;
2. o *Gerenciador* verifica se ela já não está carregada, caso esteja, o método de adicionar referência da *EstruturaDados* é chamado. Se a página não está carregada, ela primeiramente será buscada no *Disco*;
3. o *Gerenciador* verifica se a *MemoriaRAM* está cheia, caso estiver cheia a página é adicionada na *MemoriaRAM*, caso contrário, o fluxo de substituição de página será acionado (Figura 23);
4. o mapeamento da *TabelaPagina* é atualizado;
5. a referência da página carregada é adicionada na *EstruturaDados*.

7.2. Camada view

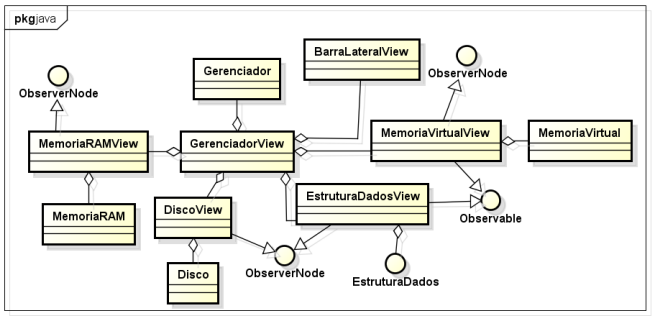

Figura 30: Diagrama de classe da camada view.

Na camada de visão, existe um componente para representar cada componente do módulo gerenciador. Toda vez que algum componente do módulo gerenciador sofrer alguma alteração a camada de visão será atualizada para representar o estado atual do gerenciador. O diagrama de classe principal da camada *view* pode ser visualizado na Figura 30.

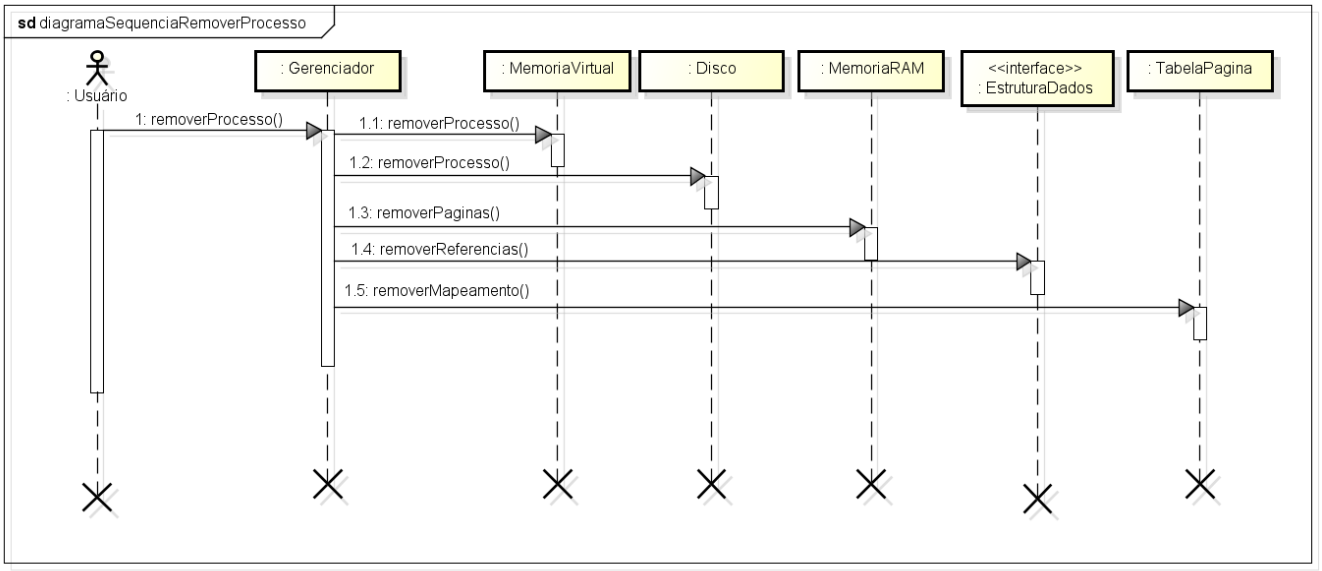

Figura 28: Diagrama de Sequência para o caso de uso remover processo

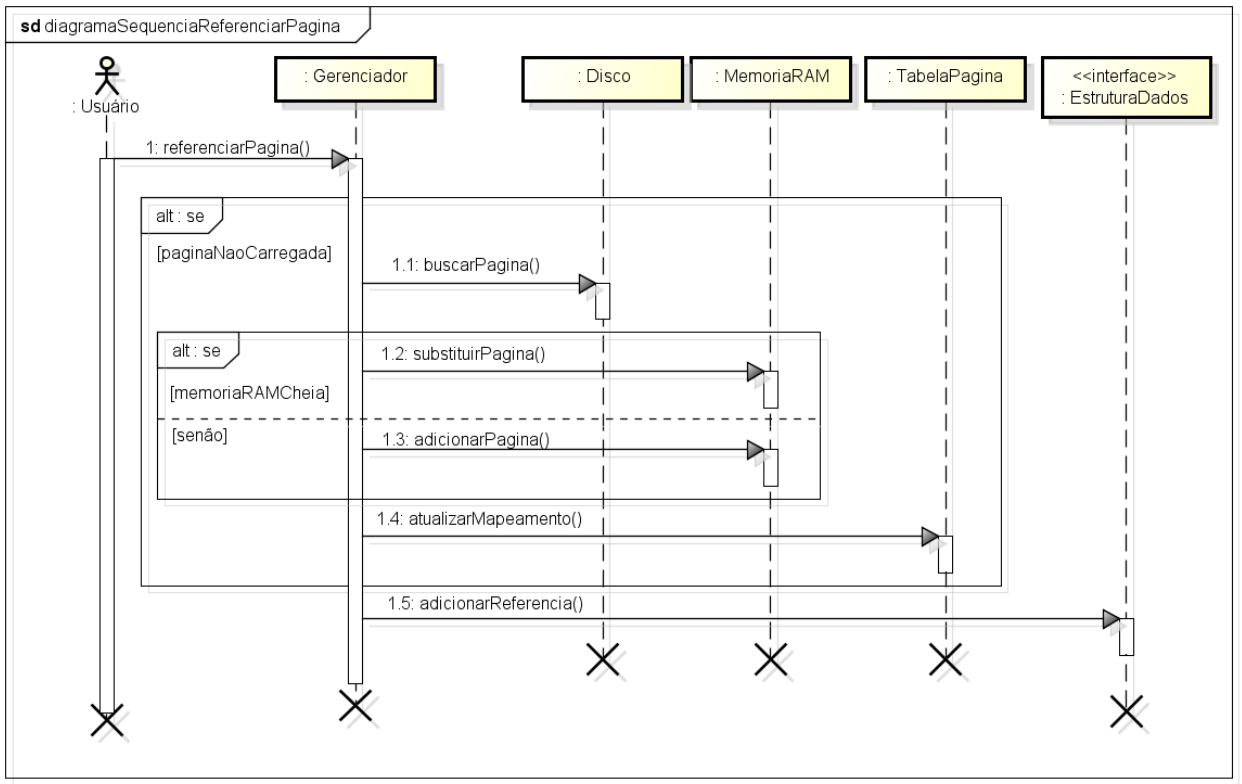

Figura 29: Diagrama de Sequência para o caso de uso referenciar página

Para que as classes de visão sejam atualizadas sempre que o estado do gerenciador mude e sempre que eventos oriundos de alguma ação do usuário ocorra, foi implementado o padrão de projeto *Observer*. Esse padrão define duas interfaces, uma para objetos observáveis e outra para objetos observadores. Um objeto observável pode ser observado por vários objetos observadores. Sempre que o objeto observado mudar de estado, os objetos que o observam são atualizados de maneira apropriada caso a atualização seja relevante para ele.

Na Figura 30, podemos verificar que as classes de visão implementam *ObserverNode* ou *Observable*. Essas interfaces implementam o padrão *Observer*. No momento de criação do *GerenciadorView*, é configurado quem observa quem. Dessa maneira é possível atualizar os componentes adequadamente sempre que for necessário. Uma *screenshot* do sistema é mostrada na Figura 31.

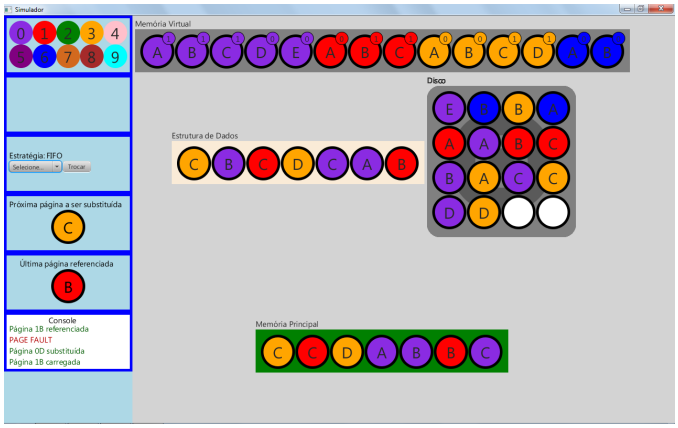

Figura 31: Screenshot do sistema.

A barra lateral em azul presente na Figura 31, é representada pela classe *BarraLateralView* (Figura 30). Essa classe agrega várias outras classes de visão para poder compor essa barra de visualização. A Figura 32 mostra como foi modelada essa parte.

O objeto *MenuProcessosView* mostra os processos que podem sem carregados na memória. O *AdicionarProcessoView* é responsável por chamar o método que adiciona o processo no *Gerenciador*. O objeto *EstrategiaPaginaView* carrega a lista de algoritmos implementados presentes em *EstrategiaEnum* (Figura 25). Conforme estratégia escolhida pelo usuário, ele instancia a estrutura apropriada e efetua a troca da mesma no *Gerenciador*. Os objetos

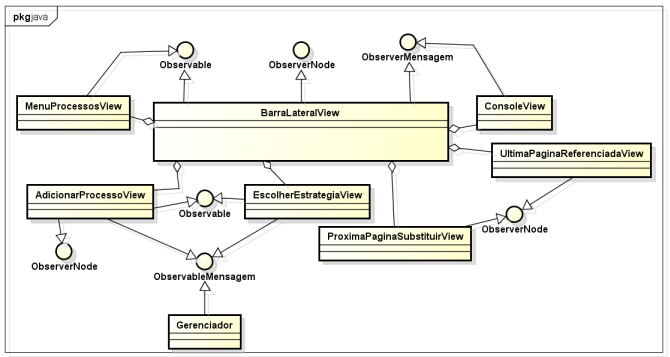

Figura 32: Diagrama de classe dos objetos que compõem a barra lateral.

ProximaPaginaSubstituirView e *UltimaPaginaReferenciadaView* mostram respectivamente, qual será a próxima página a ser substituída e qual foi a ultima página a ser referenciada. Esses objetos existem para poder facilitar o entendimento do que está sendo processado na tela. Por último, temos o objeto *ConsoleView*, que como o próprio nome diz é a representação de uma console. O *ConsoleView* serve para mostrar o log do que está sendo feito na simulação, é uma forma de melhorar ainda mais o entendimento do que está sendo simulado.

Essas classes também utilizam o artifício proposto pelo padrão *Observer*. O objeto *ConsoleView* em especial, implementa outra interface chamada *ObserverMensagem*. Essa interface segue o mesmo princípio da *ObserverNode*. Sua implementação foi necessária porque o papel da *ConsoleView* é somente mostrar o que está acontecendo na simulação. Então, sempre que alguma ação importante acontece na tela, por exemplo, uma página sendo referenciada ou um processo sendo adicionado, a *ConsoleView* é acionada para imprimir o que foi feito. A interface *ObservableMensagem* define os objetos que podem ser observados pelo *ObserverMensagem*.

8. Validação e Testes

Para a validação do modelo proposto, foi aplicada uma aula de gerência de memória virtual utilizando a ferramenta em uma turma de primeiro semestre do curso de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal da Bahia, Campus Salvador, inscritos na disciplina de ”Arquitetura

de Computadores e Software Básico", que trata do assunto de Gerência de Memória, de forma introdutória. Os conceitos relacionados à gerência de memória virtual foram abordados utilizando a ferramenta como recurso visual, uma vez que o assunto já tinha sido abordado previamente em sala de aula. Após a explicação foi realizada a demonstração dos algoritmos, ao passo em que a teoria envolvida também era explicada. No final da aula os alunos responderam a um questionário sobre o que acharam da ferramenta. Apenas sete alunos estavam presentes na aula. Os resultados obtidos do questionário podem ser visualizados a seguir:

1. O simulador te ajudou a entender a gerência de memória virtual? (Sim/Não) - 85,71% dos alunos respondeu que sim e 14,29% respondeu que não.
2. O simulador te ajudou a entender o funcionamento dos algoritmos de substituição de páginas propostos? (Sim/Não) - 71,43% dos alunos respondeu que sim e 28,57% dos alunos respondeu que não.
3. O que você achou da visualização da simulação? (Ruim/Regular/Boa/Ótima) - 71,43% dos alunos respondeu que a visualização é boa e 28,57% respondeu que é ótima.
4. Existe algum ponto da gerência de memória virtual que você acredita ter faltado na simulação? (Sim/Não) - 100% dos alunos respondeu não.
5. O simulador é fácil de usar? (Sim/Não) - 85,71% dos alunos respondeu que sim e 14,29% respondeu que não.

Além dessas respostas, os alunos fizeram alguns comentários sobre a ferramenta, os mais relevantes podem ser vistos a seguir:

1. "Gostei do simulador. Como a área de informática se utiliza muito de abstrações, todos os tipos de simuladores são importantes para o auxílio no aprendizado".
2. "Bem didático e de fácil compreensão, principalmente para os alunos iniciantes".
3. "O programa é excelente e ajuda ao aluno ter algo para auxiliar na memorização dos assuntos em sala de aula".

Através dos resultados obtidos foi possível observar que a ferramenta de fato auxilia no entendimento do assunto. Os alunos que responderam a este questionário, ainda não tinham estudado os algoritmos de substituição de páginas, mas 71,43% dos alunos conseguiu entender o funcionamente dos algoritmos simulados. Sendo que os 28,57% que não conseguiram entender o assunto, são alunos que ou chegaram atrasados ou não estavam cursando a disciplina.

A ferramenta inicialmente não foi pensada para ser utilizada como recurso facilitador no ensino de Gerência de Memória. O intuito inicial era que o aluno assistisse a aula e depois usasse a ferramenta. Mas a forma como o teste foi realizado se mostrou muito interessante, o próprio professor pode utilizar a ferramenta para ensinar o assunto e depois o aluno pode explorar com calma a ferramenta, solidificando ainda mais o conhecimento.

Com os resultados obtidos é possível notar a importância de ferramentas desse tipo, conforme pode ser observado no comentário de um dos alunos: "...Como a área de informática se utiliza muito de abstrações, todos os tipos de simuladores são importantes para o auxílio no aprendizado".

9. Conclusão e Trabalhos Futuros

Este trabalho apresenta uma ferramenta desenvolvida para servir de apoio ao processo de ensino-aprendizagem da disciplina de Sistemas Operacionais. O principal objetivo é apresentar o funcionamento dos algoritmos de substituição de páginas implementados, FIFO e MRU, e a intervenção entre os principais componentes envolvidos na gerência de memória virtual.

A ferramenta foi projetada de maneira que facilitasse a implementação de outros algoritmos pelos alunos. Além da implementação de outros algoritmos de substituição de páginas, como por exemplo, MFU e FIFO-segunda chance, outros trabalhos podem ser desenvolvidos neste contexto, porque o simulador proposto é o ponto de partida para a criação de uma ferrementa que simule o funcionamento de um sistema operacional completo. Algumas sugestões podem ser vistas a seguir:

1. simulação da tradução dinâmica de endereço virtual para o endereço real.

2. simulação da gerência de processos
3. simulação da gerência de disco
4. simulação da gerência de arquivos
5. simulação da concorrência entre processos

Todos esses trabalhos propostos podem ser implementados em projetos menores, visto que cada um deles envolve diversos assuntos e conceitos.

Referências

[1] A. Sílberschatz, P. Galvin, G. Gagne, Sistemas Operacionais, Campus, 2004.

[2] A. S. Tanenbaum, Sistemas Operacionais Modernos, Pearson, 2010.

[3] B. L. Stuart, Princípios de Sistemas Operacionais, Cengage Learning, 2011.

[4] Deitel, Deitel, Choffnes, Sistemas Operacionais, Pearson Education Brasil, 2005.

[5] I. M. Flynn, A. M. McHoes, Introdução aos Sistemas Operacionais, Thomson, 2002.

[6] W. Stallings, Arquitetura e Organização de Computadores, Prentice Hall, 2003.

[7] R. S. de Oliveira, S. S. Toscani, A. da Silva Carissimi, Sistemas Operacionais, Bookman, 2010.

[8] F. B. Machado, L. P. Maia, Um framework construtivista no aprendizado de sistemas operacionais - uma proposta pedagógica com o uso do simulador sosim (2001).

[9] C. Maziero, Reflexões sobre o ensino prático de sistemas operacionais (2002).

[10] L. P. Maia, SOSIM: Simulador para o Ensino de Sistemas Operacionais, Master's thesis, Universidade Federal do Rio de Janeiro, 2001.

[11] A. de Araújo Rodrigues, C. R. Pereira, Simulador de recursos de sistemas operacionais, 2011. Acessado em 10/02/2014.

[12] E. Y. Kioki, P. P. Santiago, A. C. Soares, Um simulador didático como ferramenta de apoio ao ensino da disciplina de sistemas operacionais (2008).

[13] G. Moritz, Simulador de mecanismos de gerência de memória real e virtual, 2004.