CENTRO UNIVERSITÁRIO LUTERANO DE PALMAS

Recredenciado pela Portaria Ministerial nº 1.162, de 13/10/16, D.O.U. nº 198, de 14/10/2016
AELBRA EDUCAÇÃO SUPERIOR - GRADUAÇÃO E PÓS-GRADUAÇÃO S.A.

Rodrigo Vasconcelos Figueiredo

OSLIVE: DESENVOLVIMENTO DO MÓDULO DE SIMULAÇÃO DA PAGINAÇÃO POR DEMANDA COM SUBSTITUIÇÃO DE PÁGINAS

Palmas – TO

2021

Rodrigo Vasconcelos Figueiredo

OSLIVE: DESENVOLVIMENTO DO MÓDULO DE SIMULAÇÃO DA PAGINAÇÃO POR DEMANDA COM SUBSTITUIÇÃO DE PÁGINAS

Projeto de Pesquisa elaborado e apresentado como requisito parcial para aprovação na disciplina de Trabalho de Conclusão de Curso I (TCC I) do curso de bacharel em Ciência da Computação pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).

Orientador: Profa. M.e Madianita Bogo Marioti.

Palmas – TO

2021

Rodrigo Vasconcelos Figueiredo

OSLIVE: DESENVOLVIMENTO DO MÓDULO DE SIMULAÇÃO DA PAGINAÇÃO POR DEMANDA COM SUBSTITUIÇÃO DE PÁGINAS

Projeto de Pesquisa elaborado e apresentado como requisito parcial para aprovação na disciplina de Trabalho de Conclusão de Curso II (TCC II) do curso de bacharel em Ciência da Computação pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).

Orientador: Profa. M.e Madianita Bogo Marioti.

Aprovado em: _____/_____/_____

BANCA EXAMINADORA

_____

Profa. M.e Madianita Bogo Marioti

Orientador

Centro Universitário Luterano de Palmas – CEULP

_____

Prof. M.e Fabiano Fagundes

Centro Universitário Luterano de Palmas - CEULP

_____

Prof. Esp. Fábio Castro Araújo

Centro Universitário Luterano de Palmas - CEULP

Palmas – TO

2021

RESUMO

FIGUEIREDO, Rodrigo Vasconcelos. **OSLIVE: DESENVOLVIMENTO DO MÓDULO DE SIMULAÇÃO DA PAGINAÇÃO POR DEMANDA COM SUBSTITUIÇÃO DE PÁGINAS.** 2021. Trabalho de Conclusão de Curso (Graduação) - Curso Ciência da Computação, Centro Universitário Luterano de Palmas, Palmas/TO, 2021.

A evolução da tecnologia disponibiliza recursos que estão mudando a forma de ensino e aprendizagem na área educacional. Aproveitando desses recursos o Grupo de Estudos em Novas Tecnologias para processos de Ensino e aprendizagem (GENTE) do Centro Universitário Luterano de Palmas está desenvolvendo a plataforma OSLive, que visa complementar o aprendizado dos alunos por meio de simulações e exercícios sobre os complexos e abstratos mecanismos vistos na disciplina de SO. A gerência de memória é uma área do sistema operacional responsável por implementar mecanismos que controlam o acesso à memória RAM. Um dos mecanismos presente nessa área é a paginação por demanda com substituição de páginas. A partir desse contexto, o objetivo deste trabalho foi desenvolver um módulo de simulação do mecanismo de paginação por demanda com substituição de páginas para a plataforma OSLive. O módulo desenvolvido permite aos alunos, por meio das simulações, complementar os conceitos vistos em sala de aula, pois possibilita a visualização e interação com o mecanismo da paginação por demanda com substituição de páginas.

Palavras-chave: Sistemas Operacionais, Gerência de Memória, Paginação por Demanda, Substituição de Páginas, OSlive

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

CPU - Unidade Central de Processamento

E/S - Entrada e Saída

FIFO - Primeiro a Entrar Primeiro a Sair (*First-in-First-out*)

GENTE - Grupo de Estudos em Novas Tecnologias para processos de Ensino

MMU - Unidade de Mapeamento de Memória (*Memory Management Unit*)

RAM - Memória de Acesso Aleatório (*Random Access Memory*)

SO - Sistema Operacional

SUMÁRIO

1 INTRODUÇÃO

O uso de recursos tecnológicos na educação vem inovando os métodos de ensino e proporcionando melhores resultados no processo de aprendizagem dos alunos. Nesse processo, além do uso de computadores, outras diversas ferramentas computacionais são disponibilizadas para ajudar o aluno a absorver melhor o conteúdo.

A simulação é um dos recursos tecnológicos disponíveis e tem sido amplamente utilizada para auxiliar na aprendizagem dos estudantes. Algumas das vantagens de sua utilização são: tornar mais concretos os conceitos abstratos; engajar estudantes em tarefas interativas; aperfeiçoar a compreensão de conceitos e apresentar conceitos abstratos de forma simplificada (MEDEIROS E MEDEIROS, 2002).

As simulações computacionais estão presentes no OSLive, uma plataforma *online* com o objetivo de facilitar o ensino e aprendizagem de alunos da disciplina de Sistemas Operacionais do Centro Universitário Luterano de Palmas (CEULP-ULBRA). A disciplina de Sistemas Operacionais, segundo as diretrizes curriculares do MEC (2012), é uma disciplina indispensável em alguns cursos da área da computação, dessa forma a plataforma OSLive serve como ferramenta de apoio no ensino dessa disciplina.

Na plataforma OSLive, os alunos têm acesso a diversos módulos com simulações dos mecanismos de gerência do sistema operacional, além de exercícios que permitem melhorar e auxiliar na compreensão dos conceitos vistos em sala de aula. Esse projeto faz parte do Grupo de Estudos em Novas Tecnologias para Processos de Ensino e Aprendizagem (GENTE) do Centro Universitário Luterano de Palmas.

Para Tanenbaum e Bos (2016), um Sistema Operacional (SO) é responsável por gerenciar todos os itens que compõem um computador, como os discos, processadores, memórias, dispositivos de entrada e saída. Ou seja, o SO disponibiliza uma interface amigável para o usuário, deixando oculta a complexidade presente na comunicação com o hardware e no gerenciamento de recursos que é realizado em baixo nível.

Para compreensão dos mecanismos presentes nos sistemas operacionais, a disciplina aborda as quatro áreas de atuação de um SO: gerência de arquivos, gerência de entradas e saídas, gerência de processos e gerência de memória. A gerência de memória usa vários mecanismos com o objetivo de manter o maior número possível de processos residentes na memória RAM, mesmo em momentos em que não tenha espaços livres, ela aplica mecanismos para permitir que novos processos sejam aceitos e executados, aumentando ao máximo o tempo de compartilhamento da CPU (MACHADO e MAIA, 2007).

A paginação é um dos mecanismos implementados pela gerência de memória. Na paginação, ocorre uma abstração da memória, em que são criados dois espaços de endereçamento: o espaço de memória lógica do processo, que é um conjunto de endereços que o processo usa para endereçar a memória física; e o espaço de endereçamento físico , que é o conjunto de endereços que a memória física disponibiliza (TANENBAUM E BOS, 2016). Os espaços de endereçamento lógico e físico são divididos em blocos do mesmo tamanho, respectivamente, chamados de páginas e quadros. No mecanismo de paginação, quando um programa é executado todas as páginas do processo são carregadas para os quadros da memória física.

Carregar um programa inteiro na memória física, mesmo quando apenas alguns recursos serão utilizados, pode ter um custo muito alto no que se refere ao compartilhamento de memória na multiprogramação. Para Machado e Maia (2007), uma estratégia alternativa é a paginação por demanda, que permite carregar na memória principal apenas as páginas que estão sendo acessadas pelo programa e, ao decorrer da execução, ir carregando as páginas de acordo com a necessidade dos processos.

Mesmo em um sistema com paginação por demanda, pode ocorrer de todos os espaços livres da memória serem ocupados, portanto, para que um novo quadro seja ocupado na memória física a solução mais comum é utilizar algoritmos de substituição de páginas. Segundo Tanenbaum e Bos (2016), a substituição de páginas ocorre quando o sistema operacional tem que escolher uma página para remover da memória física (página vítima) para ceder espaço a um novo conteúdo.

O mecanismo de paginação por demanda com substituição de páginas, como outras ações de gerência do SO, é executado em baixo nível. Portanto, as ações são realizadas pelo SO sem a visualização e percepção dos usuários e faz com que esses conceitos se tornem complexos e difíceis de serem compreendidos pelos alunos.

Nesse contexto, o objetivo deste trabalho foi a implementação do módulo da paginação por demanda com substituição de páginas para complementar a compreensão de conceitos do mecanismo estudado na disciplina de sistemas operacionais. Esse módulo faz parte da plataforma OSLive e oferece uma interface gráfica que permite aos alunos visualizar e interagir, por meio das simulações e da entrada de dados, com o mecanismo da paginação por demanda com substituição de páginas.

Por fim, este trabalho é estruturado da seguinte forma: seção 2, Referencial Teórico, são apresentados os conceitos de Sistemas Operacional, Gerência de Memória, Paginação, Paginação por Demanda e os Algoritmos de Substituição de Páginas; seção 3, Metodologia, é

dividida em Desenho de Estudo e Materiais; seção 4, Resultados e Discussão, apresenta de forma detalhada os resultados obtidos na implementação do módulo e é dividida em Arquitetura de Software, Interface da aplicação e Demonstração da Simulação; seção 5, Considerações Finais, apresenta as conclusões referentes a execução deste trabalho e, posteriormente, são listadas as referências que serviram como base para a construção do trabalho.

2 REFERENCIAL TEÓRICO

Nesta seção é apresentada uma visão geral dos sistemas operacionais e suas quatro áreas: gerência de arquivos, gerência de entrada e saída, gerência de memória e gerência de processos. Na apresentação do conteúdo, há um aprofundamento na área de gerência de memória, mais especificamente no mecanismo de paginação por demanda com algoritmos de substituição de páginas, por ser o mecanismo que será simulado na aplicação.

2.1 SISTEMA OPERACIONAL

Um Sistema Operacional, segundo Silberschatz, Galvin e Gagne (2012), é responsável por gerenciar os recursos de *software*, *hardware* e dados e, através desse gerenciamento, oferecer o ambiente para que os demais programas possam ser executados. De acordo com Machado e Maia (2007), o Sistema Operacional atua como interface entre os usuários e os recursos disponíveis no SO e torna transparente as tarefas realizadas em baixo nível. O Sistema Operacional é responsável por intermediar e facilitar a comunicação entre o *hardware* e os usuários, uma vez que seria muito complexo para grande maioria dos usuários compreender e executar as ações de baixo nível que são atribuídas ao SO.

Para lidar com numerosas solicitações de recursos e decidir a maneira de alocá-los de forma eficiente e justa, sem que haja conflito, o sistema operacional trabalha como um alocador de recursos (SILBERSCHATZ, GALVIN E GAGNE, 2012), dividido em quatro áreas:

- **Gerência de Processos**: para Tanenbaum e Bos (2016), um processo é basicamente a instância de um programa em execução, que pode conter outros processos e recursos em sua hierarquia. Diversos processos são executados de forma concorrente em um SO é função da gerência de processos gerenciar os processos durante todos seus ciclos de vida e implementar mecanismos de escalonamento para garantir que todos os processos sejam atendidos pela CPU;
- **Gerência de Memória:** nos sistemas operacionais vários processos são mantidos na memória RAM para garantir o desempenho. A gerência de memória tem a função de fornecer mecanismos que permitam que todos os processos tenham acesso a memória física. As políticas implementadas pela gerência de memória são responsáveis por manter diversos programas na memória, controlar espaços utilizados decidir quais processos devem permanecer, alocar e desalocar espaços na memória quando necessário (SILBERSCHATZ, GALVIN E GAGNE, 2012);

- **Gerência de Dispositivos de E/S:** segundo Machado e Maia (2007), essa é uma das partes mais complexas do sistema operacional, pois possui uma estrutura em camadas para separar e ocultar as informações de baixo nível das camadas superiores. A camada de dispositivo de E/S de baixo nível e a camada de subsistema de E/S, responsável por comunicar com sistema de arquivo e aplicações, se comunicam por meio da camada do *device driver*. Essa camada permite conexões de variados dispositivos de forma simples para o usuário;
- **Gerência de Arquivos:** área também conhecida como sistema de arquivos, a gerência de arquivos realizada pelo SO tem como objeto fornecer acesso de leitura, gravação e manipulação aos diversos dispositivos de armazenamento de forma amigável para os usuários (MACHADO E MAIA, 2007).

A seção seguinte aborda a gerência de memória mais detalhadamente, apresentando conceitos importantes para a implementação da simulação da paginação por demanda com substituição de páginas, que é o objetivo deste trabalho.

2.2 GERÊNCIA DE MEMÓRIA

Os sistemas operacionais multitarefas executam diversos processos simultaneamente, através da divisão do tempo do processador. Para garantir que o processador alcance o maior desempenho possível, é necessário manter vários processos na memória (RAM), prontamente disponíveis para serem acessados e executados. Portanto, a gerência de memória tem a função de fornecer os mecanismos para possibilitar que diversos programas possam utilizar e compartilhar a memória de forma segura e eficiente (OLIVEIRA, CARISSIMI E TOSCANI, 2009).

Segundo Tanenbaum e Bos (2016), nos primeiros computadores os programas acessam diretamente a memória física, não havendo como identificar áreas que estavam sendo utilizadas por outros programas ou pelo próprio sistema operacional. Dessa forma, permitir que processos acessem a memória física diretamente pode trazer desvantagens, como causar parada total ou derrubar o sistema operacional, além de tornar praticamente impossível a execução de múltiplos programas ao mesmo tempo. Por exemplo, se um programa está usando um espaço de endereçamento da memória física e outro programa é executado, como ele desconhece os espaços da memória física que já estão ocupados, possivelmente ele substituirá endereços do programa que já está em execução, causando uma falha nos dois programas.

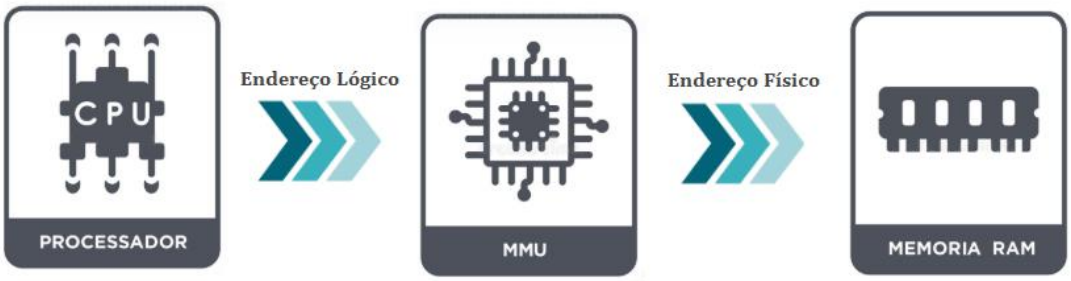
Uma alternativa para contornar o problema de acessar a memória diretamente foi criar uma abstração de memória utilizando dois espaços de endereçamento, memória lógica e

memória física. A memória lógica é formada por todos os endereços lógicos de um processo e cada processo tem a memória lógica independente. A memória física é o conjunto de endereços físicos disponibilizado pela memória RAM, ou seja, o endereço físico corresponde a uma posição real na memória (OLIVEIRA, CARISSIMI E TOSCANI, 2009).

Os processos conhecem apenas os endereços da memória lógica e desconhecem os endereços da memória física. Quando um processo deseja acessar um endereço na memória física, é necessário que ocorra um mapeamento do endereço lógico conhecido para o respectivo endereço físico.

A MMU (*Memory Management Unit*) é o dispositivo de hardware responsável por mapear endereços lógicos em endereços físicos em tempo de execução. Ao executar uma instrução, a CPU não acessa diretamente os endereços da memória física, de forma que passa o endereço lógico para a MMU que se encarrega de mapear e localizar o endereço requisitado na memória física, como mostra a figura 1.

Figura 1. MMU - Conversão de endereço lógico em endereço físico



Fonte: Elaborada pelo autor

A maioria dos computadores modernos oferecem o recurso de memória virtual, o qual permite que programas maiores que a memória física livre sejam executados, por meio da troca dinâmica de partes do processo entre memória virtual e memória física (TANENBAUM E BOS 2016). A memória virtual usa o armazenamento secundário (disco) para armazenar as partes do processo que não estão em execução, o fato de usar o disco faz com que se tenha poucas limitações de armazenamento.

Na literatura são definidos diversos mecanismos para Gerência de Memória, entre eles os mecanismos de paginação e de paginação por demanda, que é utilizado como base para as políticas de segurança do Windows e do Linux. Esses mecanismos serão discutidos mais detalhadamente a seguir, por serem o foco do projeto.

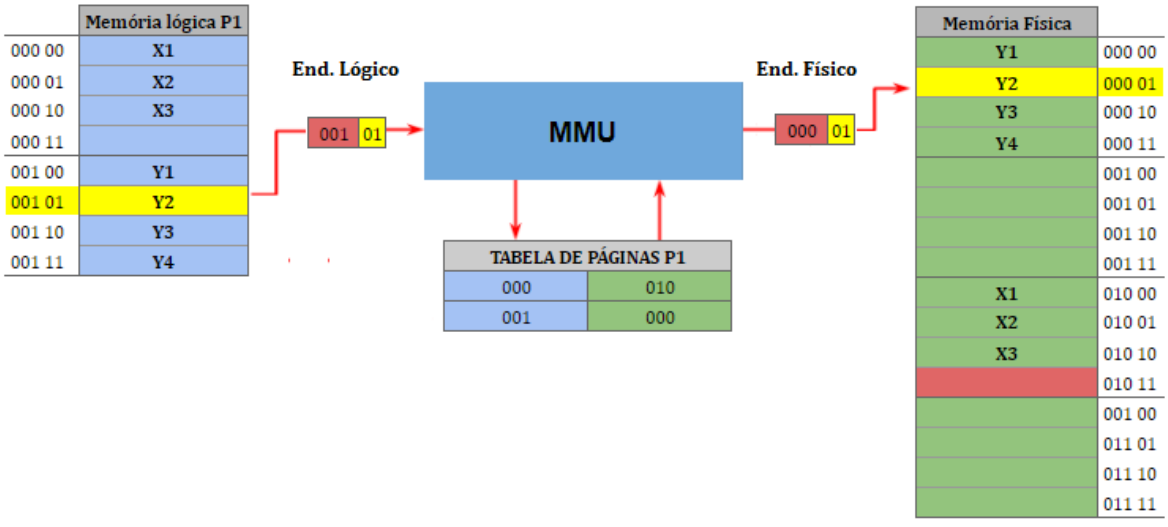
2.2.1 PAGINAÇÃO

Na paginação, a memória lógica do processo é dividida em blocos de tamanho fixo, chamados de páginas, e tem suas unidades correspondentes na memória física, chamados de quadro de páginas (TANENBAUM E BOS 2016). Geralmente, páginas e os quadros possuem tamanhos iguais definidos pela MMU, além disso, cada bloco e página possui um número de endereço que é usado como índice na tabela de páginas.

No mecanismo de paginação, quando um programa é executado todas as páginas da memória lógica do programa são carregadas nos quadros da memória física, de forma que as páginas podem ser carregadas em qualquer quadro livre na memória. É dispensável a carga de forma contígua, ou seja, na paginação não é necessário que as páginas ocupem um espaço sequencial de memória física, podendo ocupar qualquer quadro disponível.

Toda carga de página lógica para a memória física gera uma entrada na tabela de páginas composta pelo número da página lógica e o número quadro correspondente na memória física. A tabela de páginas é responsável por associar endereços lógicos e físicos e auxiliar a MMU no processo de mapeamento desses endereços.

Figura 2. Mecanismo básico de paginação



Fonte: Elaborada pelo autor

No exemplo do mecanismo de paginação da figura 2, a memória física é composta por 16 bytes e está dividida em 4 quadros. A memória lógica do processo P1 é composta por 8 bytes e foi dividida em 2 páginas. Ambas as memórias foram divididas em blocos com tamanho fixo

de 4 bytes, sendo o endereço dos bytes formados por duas partes: um número da página e o seu deslocamento dentro dessa página.

A figura 2 apresenta a relação entre o endereço lógico do byte Y2 em seu endereço físico equivalente. Para localizar o endereço de Y2 na memória física, a MMU recebe o endereço lógico 001 01 e com o auxílio da tabela de páginas identifica que o endereço físico correspondente de Y2 na memória física é 000 01.

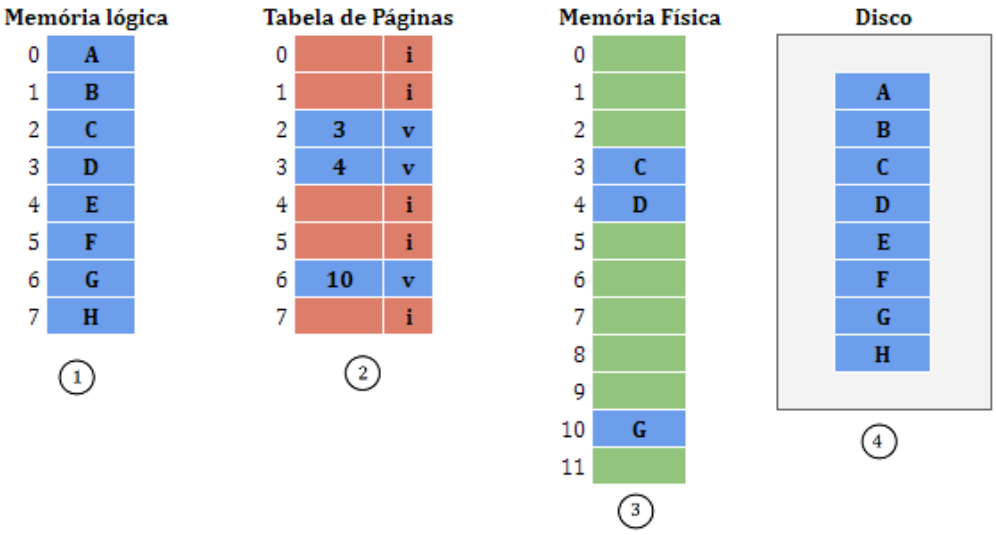
Na paginação, toda a memória lógica do processo deve estar carregada na memória física para o programa ser executado corretamente. Se a memória disponível não for muito grande, possivelmente não será o suficiente para executar programas como editores de texto e de vídeo, pois todos os recursos do programa, mesmo os que não estão sendo utilizados, serão carregados na memória. Para solucionar esse problema, foi criada a paginação por demanda, que associa o mecanismo de paginação ao uso de memória virtual.

2.2.2 PAGINAÇÃO POR DEMANDA

A paginação por demanda é baseada na paginação simples, associada à memória virtual, que usa o disco como memória auxiliar. Semelhante a paginação simples, cada processo tem sua área de memória lógica dividida em blocos de tamanho fixo, chamados de páginas lógicas (OLIVEIRA, CARISSIMI E TOSCANE, 2001), que são carregadas na memória física em quadros (páginas físicas) que possuem o mesmo tamanho das páginas lógicas. Durante o processo de carga, cada página carregada na memória física gera uma entrada na tabela de páginas, que auxilia no mapeamento de uma página lógica em física.

O que ocorre de diferente na paginação por demanda, em relação à paginação, é que apenas as páginas que estão sendo utilizadas efetivamente pelo processo são carregadas para a memória física, assim, as páginas que não estão sendo utilizadas ficam no disco. Uma forma de identificar se uma página está na memória física ou no disco é através do bit de validade (válido/inválido), presente na tabela de páginas (SILBERSCHATZ, GALVIN E GAGNE, 2012). Se o bit for inválido, pode indicar que a página está no disco e ainda não foi carregada para a memória física ou que a página solicitada está fora do endereçamento lógico do processo. Caso o bit seja válido, o conteúdo da página lógica está carregado na memória, de forma que o mapeamento do endereço lógico em físico é realizado normalmente, sem a necessidade de novas ações do SO.

Figura 3. Mecanismo básico de paginação por demanda

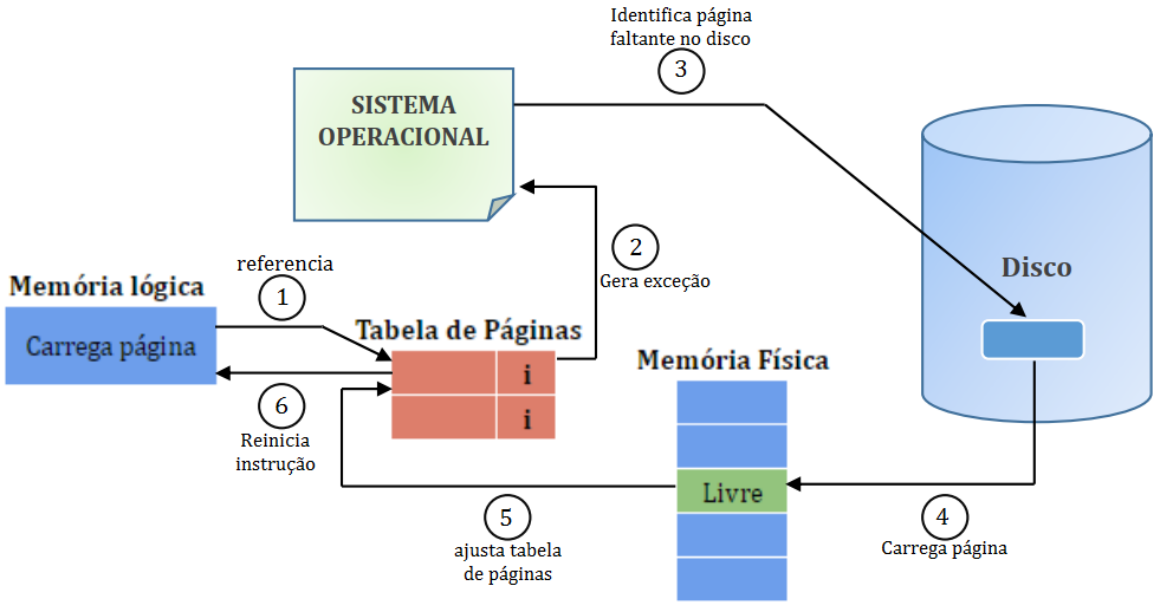


Fonte: Elaborada pelo autor

No exemplo do mecanismo de paginação por demanda da figura 3, a memória física (3) possui doze quadros e a memória lógica (1) possui 8 páginas. No entanto, apenas 3 páginas lógicas estão carregadas na memória física. As páginas lógicas 2, 3 e 6, que estão carregadas na memória física estão destacadas em azul na tabela de páginas (2), possuem respectivamente os endereços físicos 4, 5 e 10 e estão marcadas com o bit válido (v), indicando que as páginas estão carregadas na memória física. Por outro lado, as entradas da tabela de páginas destacadas em vermelho não possuem o endereço físico e estão marcadas com bit inválido (i), o que indica que essas páginas não foram carregadas na memória física.

Para uma página com bit válido na tabela de páginas, o mapeamento do endereço lógico em físico ocorre normalmente, como foi demonstrado na paginação. Caso o bit esteja marcado como inválido, a unidade de gerência de memória (MMU) gera uma interrupção e aciona o sistema operacional (OLIVEIRA, CARISSIMI E TOSCANI 2001). Essa interrupção pode ser causada por uma tentativa de acesso a uma página fora do endereçamento do processo, nesse caso, o SO aborta o processo. Caso a página lógica acessada esteja no endereçamento do processo, é gerada uma interrupção por falta de página (*page-fault*), que significa que aquela página ainda não foi carregada na memória física. A figura 4 apresenta as etapas para tratamento de uma falta de página.

Figura 4. Etapas da paginação por demanda - Adaptada



Fonte: Silberschatz, Galvin e Gagne (2012)

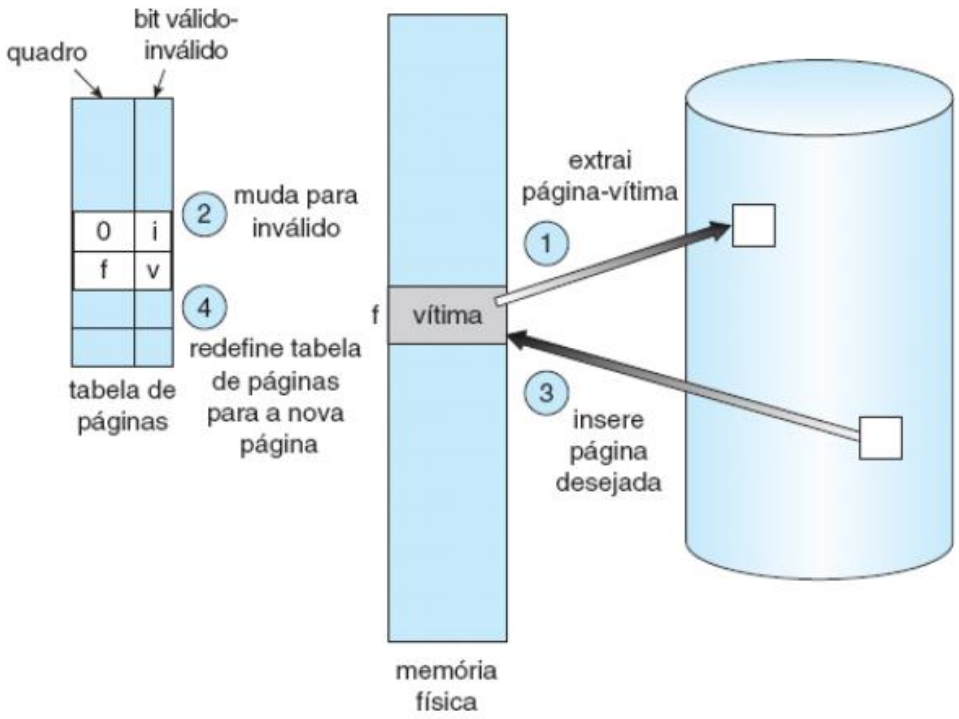
Na primeira etapa, ao referenciar uma página marcada como inválida na tabela de páginas, o SO verifica no descritor do processo se a página pertence ao endereçamento lógico do processo. Caso a página seja do processo, na segunda etapa, uma exceção de *page-fault* é gerada para o sistema operacional. Após ser acionado pela interrupção por falta de página, na terceira etapa, o SO inicia o processo de alocação da página faltante. Na quarta etapa, a página é carregada do disco para o quadro de memória. Com a página já carregada na memória, a tabela de página é atualizada com o número do quadro correspondente e o bit é alterado para válido. Para finalizar, a instrução que gerou a falta de páginas deve ser executada novamente.

Com vários processos executando ao mesmo tempo, pode ocorrer de todas as páginas físicas estarem ocupadas quando ocorrer uma falta de página. Nesse caso, o sistema operacional, a partir do uso de algoritmos de substituição de páginas, escolhe uma página vítima para remover da memória e abrir espaço para uma nova página.

2.2.3 Algoritmos de Substituição de Páginas

Com todas as páginas físicas ocupadas, quando ocorre a falta de página, o sistema operacional tem que escolher uma página para ser removida da memória, conhecida como página vítima. O algoritmo de substituição de páginas define regras para escolher uma página vítima, se a página escolhida para ser removida sofreu alguma modificação enquanto estava na memória, ela deve ser atualizada no disco, caso contrário pode ser substituída sem a necessidade de reescrevê-la no disco (TANENBAUM E BOS 2016).

Figura 5. Substituição de páginas



Fonte: Silberschatz, Galvin e Gagne (2012)

A figura 5 mostra uma situação em que ocorre a substituição de páginas. No passo 1, após identificar que não há quadros livres na memória física, um algoritmo de substituição de páginas é utilizado para selecionar um quadro vítima que terá seu conteúdo gravado de volta no disco. No passo 2, na tabela de páginas a página gravada no disco tem seu bit alterado pata inválido. Com o quadro de memória física liberado, no passo 3, a nova página é carregada do disco para a memória física e, consequentemente, tem sua entrada atualizada para válida na tabela de páginas, passo 4.

É possível identificar que no processo de substituição de páginas da figura 5 ocorreram duas transferências de páginas da memória física para o disco e vice-versa. Esse processo faz com que o tempo de carga da nova página seja efetivamente duplicado, uma forma de reduzir esse tempo é utilizar bits auxiliares associados a tabela de páginas para tornar o mecanismo mais simples e eficiente. Os bits comumente utilizados são: bit de sujeira ou modificação, bit de referência e o bit de tranca.

O bit de sujeira, ou de modificação, identifica se a página sofreu alguma alteração enquanto estava na memória. Esse bit pode ser usado para reduzir a quantidade de leitura e escrita em disco (*overhead)*, pois quando o bit de modificação possui valor 0 não há necessidade de atualizar a página no disco, sendo necessário apenas trazer a nova página para a memória física.

Para identificar se a página foi acessada recentemente, cada entrada na tabela de páginas recebe um bit de referência. O bit de referência de uma página é atualizado a cada intervalo de tempo da CPU, sendo o seu valor alterado para 1 sempre que a página é referenciada, seja para escrita ou leitura. O bit recebe valor 0 quando a página é carregada na tabela de páginas ou quando a página não foi referenciada em um intervalo de tempo da CPU.

O bit de tranca é utilizado pelo sistema operacional para bloquear a remoção de uma página lógica da memória física. Quando uma página lógica é referenciada por um processo para leitura ou escrita, o SO altera o bit de tranca da página para 1, dessa forma, evita que a página seja escolhida como vítima durante sua utilização. Quando o processo é encerrado o SO altera o bit para 0 e libera a página.

Os algoritmos de substituição de páginas têm como objetivo escolher uma página vítima quando não há mais quadros disponíveis na memória. No processo de escolha da página vítima, os algoritmos utilizam os bits auxiliares de forma independente ou combinada, de acordo com o seu funcionamento.

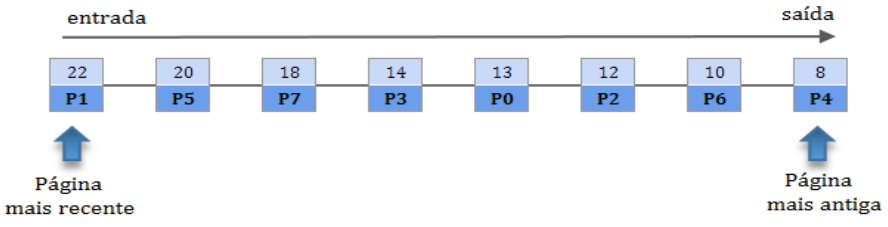
2.2.3.1 ALGORITMO FIFO

O algoritmo primeiro a entrar, primeiro a sair (*First-In-First-Out - FIFO)* é a implementação mais simples dos algoritmos de substituição de páginas. Segundo Machado e Maia (2007), o FIFO seleciona a primeira página que foi carregada para a memória, ou seja, a página que está a mais tempo na memória principal é a escolhida para ser substituída.

Para identificar a página mais antiga na memória o algoritmo FIFO associa a cada página a hora em que a página foi trazida para memória (*Timestamp*), outra forma de identificar as páginas mais antigas é manter uma lista, onde a página mais antiga será encontrada no início da lista (SILBERSCHATZ, GALVIN E GAGNE, 2012).

Figura 6. Escolha de página vítima com algoritmo FIFO

Página	Momento da Carga
P0	13
P1	22
P2	12
P3	14
P4	8
P5	20
P6	10
P7	18



Fonte: Elaborada pelo próprio autor

Na figura 6, a tabela de página possui as páginas P0 a P7 carregadas e seus respectivos momentos de carga (*Timestamp*). Todas as páginas são mantidas em uma lista ordenada pelo momento de carga. As páginas mais recentes, com os maiores momentos de carga, ficam no final, pois acabaram de chegar, já as páginas mais antigas, com menores momentos de carga, ficam no início da lista por serem mais antigas. Por exemplo, a página P1 foi carregada no momento 22 e possui o maior tempo de carga, por isso, está no final da lista. Por outro lado, a página P4 foi carregada no momento 8, não havendo página com o momento de carga menor, ela assume a frente da lista. Em uma próxima falta de página, P4 será escolhida como página vítima e será removida da memória.

Usar somente o fator tempo não é suficiente para julgar se uma página está sendo ou não usada constantemente pelo sistema. Portanto, o algoritmo FIFO raramente é usado sem o auxílio de outros mecanismos para melhorar seu desempenho.

2.2.3.2 ALGORITMO ÓTIMO

O algoritmo ótimo de substituição de páginas garante a menor taxa de erro de páginas possível, pois considera remover a página que não será utilizada pelo período de tempo mais longo (SILBERSCHATZ, GALVIN E GAGNE, 2012). Ou seja, seleciona para ser substituída páginas que não serão mais referenciadas no futuro ou que levarão o maior intervalo de tempo para serem utilizadas novamente.

Na prática, o algoritmo ótimo não pode ser implementado, pois o sistema operacional não consegue prever quando cada página será referenciada novamente. Mesmo não sendo implementado, por ser o melhor possível esse algoritmo serve como base para comparar o desempenho dos demais algoritmos de substituição de páginas que são implementados.

Nenhum outro algoritmo consegue alcançar uma taxa de erros de páginas inferior ao algoritmo ótimo. Por exemplo, em uma sequência de testes de referência que se obtém uma quantidade de 10 erros de páginas com o algoritmo ótimo, por mais esforços que se faça na criação de um novo algoritmo para esse mesmo teste o mínimo de erros que este algoritmo irá apresentar serão 10 erros.

A filosofia do algoritmo ótimo é empregada em outros algoritmos de substituição de páginas, visando identificar páginas que não serão utilizadas em um futuro próximo e gerar a menor quantidade de erros de páginas possível.

2.2.3.3 ALGORITMO LRU

O LRU (*least recently used* - usada menos recentemente) é uma aproximação do algoritmo ótimo, pois sua eficiência é comparada a este algoritmo. De acordo com Tanenbaum e Bos (2016), a tendência é que páginas que estão a muito tempo sem serem usadas, provavelmente, não serão utilizadas por um longo período. Partindo dessa ideia, quando for necessário substituir uma página, o LRU fará uma busca pela página que está a mais tempo sem ser referenciada.

Para sua implementação, é necessário que cada página possua o momento do último acesso, sendo necessário atualizar o *timestamp* a cada referência de memória. Outra forma de implementar o LRU seria manter uma lista encadeada de todas as páginas, ordenadas pelo momento do último acesso.

Ambas as formas de implementação do LRU necessitam de apoio substancial de hardware para ser realizada. Também, há um custo muito alto a cada referência de memória no tempo gasto em acessos e atualizações das listas encadeadas. Esses fatores contribuem para que o LRU, em sua forma pura, seja pouco realizável na prática. No entanto, existem algoritmos alternativos ao LRU, como o de histórico de bit de referência e o algoritmo de segunda chance, que se aproximam do LRU, pois tem como objetivo escolher a página que está a mais tempo sem ser referenciada.

2.2.3.4 HISTÓRICO DE BITS DE REFERÊNCIA

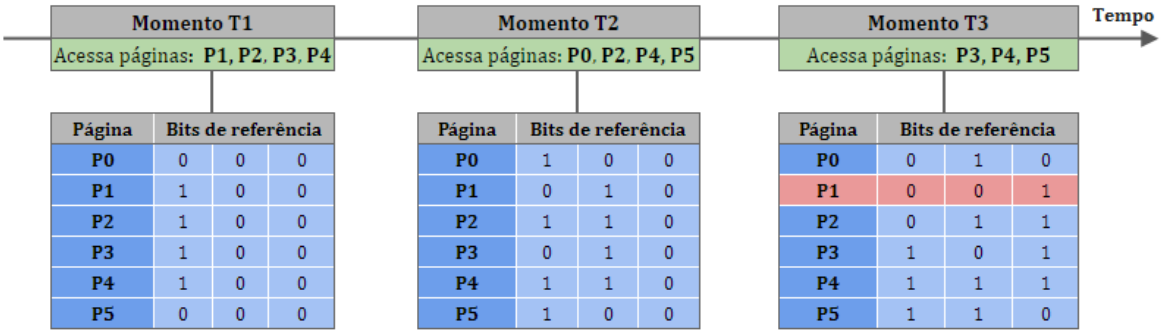
O algoritmo histórico de bits de referência, mantém o histórico de acessos que uma página teve por alguns intervalos de CPU. A partir desse histórico o algoritmo seleciona a página que está a mais tempo sem ser acessada para ser removida, ou seja, baseado nos acessos passados da página o algoritmo entende que a página que está a mais tempo sem ser acessada têm menos probabilidade de ser referenciada.

Considerando que o bit de referência associado a cada página tem seu valor alterado a cada referência de memória, com ele é possível identificar se a página foi acessada na última referência de memória. Porém, apesar de identificar se uma página foi ou não usada no último acesso a memória, com apenas um bit de referência não é possível obter um histórico de uso da página. Assim, nos casos em que mais de uma página esteja com bit desligado (valor 0), se torna impossível identificar qual foi a ordem de uso dessas páginas.

Para resolver essa questão pode ser utilizado um histórico de bit de referência, dessa forma pode-se obter informações adicionais sobre a ordem de uso das páginas, registrando os

bits de referência em intervalos regulares (SILBERSCHATZ, GALVIN E GAGNE, 2012). No histórico bits, o conjunto de bits de referência formam uma espécie de contador. Todos os bits de referência que formam o contador iniciam com valor 0 e a cada interrupção de CPU, a sequência de bit da página é deslocada um bit à direita para que o novo bit de referência seja adicionado ao bit mais à esquerda.

Figura 7. Construção do histórico de bits de referência



Página	Bits de referência		
P0	0	0	0
P1	1	0	0
P2	1	0	0
P3	1	0	0
P4	1	0	0
P5	0	0	0

Página	Bits de referência		
P0	1	0	0
P1	0	1	0
P2	1	1	0
P3	0	1	0
P4	1	1	0
P5	1	0	0

Página	Bits de referência		
P0	0	1	0
P1	0	0	1
P2	0	1	1
P3	1	0	1
P4	1	1	1
P5	1	1	0

Fonte: Elaborada pelo próprio autor

Na figura 7, a tabela de páginas possui as páginas P0 a P5, com histórico de bits formado por três bits de referência cada uma. No momento T1 as páginas P1 a P4 são acessadas, ou seja, tem o bit mais à esquerda alterado para 1. No momento T2 as páginas P0, P2, P4 e P5 receberam acesso, e antes de modificar o bit mais à esquerda o contador das páginas é deslocado um bit à direita, após isso, as páginas acessadas recebem o valor 1 no bit de referência mais à esquerda. Em T3 as páginas P3, P4 e P5 são acessadas, consequentemente, após deslocar o contador um bit para a direita, as páginas acessadas têm o bit de referência alterado para 1.

No momento T3 da figura 7, a página P4 possui o contador com a sequência de bits 111, isso indica que a página foi referenciada nos últimos 3 intervalos de tempo da CPU. Já a página P1 possui o contador com sequência de bit 000, ou seja, essa página não foi acessada nos últimos 3 intervalos de tempo da CPU. Considerando que o algoritmo histórico de bits busca remover a página menos usada recentemente, então a página P1, cujo a sequência de bit é 000 seria a página escolhida pelo algoritmo.

2.2.3.5 ALGORITMO SEGUNDA CHANCE

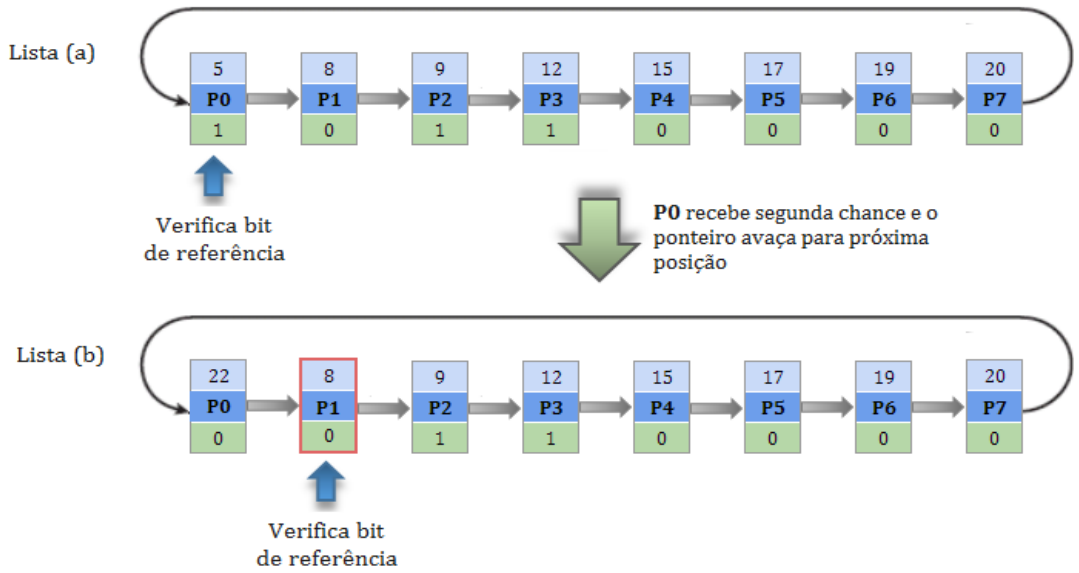
O algoritmo de segunda chance é baseado no FIFO, com a vantagem de utilizar o auxílio do bit de referência para evitar que as páginas constantemente referenciadas sejam removidas da memória (TANENBAUM E BOS 2016). Ao verificar o bit de referência da página

candidata, o algoritmo de segunda chance não considera apenas sua idade, mas também sua importância. Dessa forma, se a página possuir o bit de referência igual a 1, ela recebe uma segunda chance, pois o algoritmo entende que, uma página acessada recentemente, possivelmente será acessada em um futuro próximo.

Ao receber a segunda chance a página tem seu bit de referência zerado e seu tempo de carregamento é atualizado para aquele momento e então a pesquisa continua (SILBERSCHATZ, GALVIN E GAGNE, 2012). Quando encontra uma página com bit de referência igual a zero, o algoritmo de segunda chance se encarrega de remover a página vítima e colocar a nova página na mesma posição na lista.

A implementação do algoritmo de segunda chance pode ser realizada de duas formas: utilizar uma lista encadeada para controlar a ordem que as páginas são carregadas na memória e, quando uma página receber uma nova chance, movê-la para o final da lista; a outra forma e manter as páginas em uma lista circular, de forma que um ponteiro percorre as páginas a partir da mais antiga para mais nova até completar o círculo. Nessa última implementação não é necessário ficar movendo as páginas em torno da lista, o que a torna o algoritmo mais eficiente, conforme a figura 8.

Figura 8. Escolha de página vítima com algoritmo segunda chance



Fonte: Elaborada pelo próprio autor

Na figura 8, a lista circular de páginas possui as páginas P0 a P7. As páginas da lista contêm o momento de carga na parte superior (*timestamp*) e o bit de referência na parte inferior, destacado em verde. Quando ocorre a falta de página, o ponteiro começa a percorrer a lista a

partir da página mais antiga, com menor *timestamp*, para encontrar uma página com bit de referência igual a zero.

No processo de escolha de página vítima da figura 8, o ponteiro inicia a verificação na página P0, por ser a mais antiga da lista (a) conforme identifica seu *timestamp*. Após identificar que P0 possui bit de referência com valor 1, o algoritmo concede uma segunda chance à página, zera o bit de referência e atualiza o momento de carga para o momento mais recente. Dada a segunda chance para P0, o ponteiro avança para página P1, conforme lista (b) da figura 8. Uma vez identificado o bit de referência de P1 com valor 0, o algoritmo de segunda chance deve remover a página P1 da lista e adicionar a nova página na mesma posição com bit de referência igual a 1 e com o momento de carga mais recente.

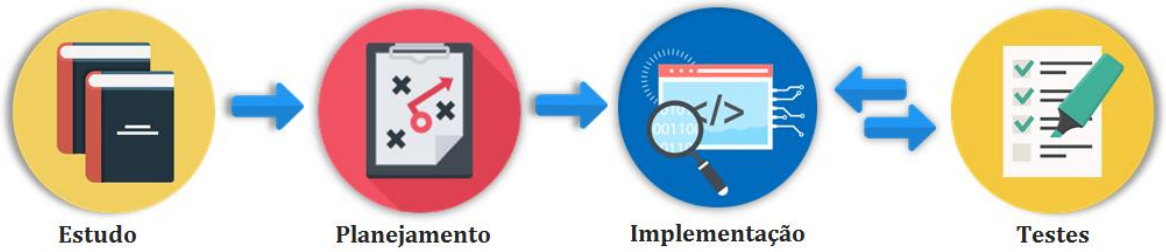
3 METODOLOGIA

Esta seção apresenta o desenho do estudo contendo as etapas para execução deste trabalho, bem como a definição das ferramentas (*softwares*) utilizados para o desenvolvimento do módulo.

3.1 DESENHO DO ESTUDO

A figura 9 apresenta as fases que foram seguidas para o desenvolvimento do módulo de simulação da paginação por demanda com substituição de páginas e indica qual sequência foi seguida até a finalização do projeto.

Figura 9. Sequência das etapas de desenvolvimento do projeto



A fase de estudos engloba toda a parte de construção de conhecimento sobre o tema. Nessa fase, o estudo dos conceitos foi realizado a partir de livros didáticos e artigos científicos, o que serviu como base para construção do referencial teórico e, também, para a aquisição do domínio sobre o tema do projeto a ser desenvolvido.

Com o conhecimento prévio adquirido na fase de estudo, iniciou-se a fase de planejamento. Na fase de planejamento foi elaborada a representação gráfica do mecanismo de paginação por demanda e dos algoritmos de substituição de páginas, esse processo foi feito com base nas representações gráficas apresentadas nos livros didáticos, buscando mostrar de uma maneira simples e compreensível o funcionamento do mecanismo. Dentre os algoritmos representados nessa fase, o algoritmo primeiro a entrar, primeiro a sair (*First-In-First-Out - FIFO*) foi definido como algoritmo que será implementado para simular a substituição de páginas. Após essa etapa, foram definidas as ferramentas utilizadas para execução do projeto na fase de implementação.

A terceira etapa consistiu na implementação da simulação da representação da paginação por demanda com algoritmos de substituição de páginas. Nesta fase foi

implementada a interface para entrada de dados do usuário, assim como a apresentação das simulações e dos resultados. As ferramentas utilizadas nessa fase estão descritas na seção 3.2.

Por fim, na fase de testes, foram realizados os testes de funcionalidade com o objetivo de checar se as funcionalidades estavam adequadas ao objetivo estabelecido inicialmente no projeto. Nesta fase, funcionalidades que apresentaram comportamento não esperado foram remetidas à fase de implementação para correção.

3.2 SOFTWARES

Para implementação do módulo, desde sua interface gráfica até a simulação dos algoritmos de substituição de páginas, serão utilizadas as seguintes ferramentas:

- **Angular**: é uma plataforma e *framework* que fornece a estrutura necessária para facilitar a construção de aplicações *web* baseadas em JavaScript (ZABOT E MATOS, 2020). Dentre os principais elementos do *framework* pode-se destacar os componentes, roteamento, módulos, templates, serviços e ferramentas de teste unitário de aplicações. No trabalho será utilizado para estruturar a aplicação e fornecer a comunicação entre os algoritmos de simulação e a interface gráfica;
- **Notify.js:** é um plugin jQuery para fornecer notificações simples, mas totalmente personalizáveis (NOTIFY.JS, 2021). Foi utilizado para criar notificações sobre eventos realizados pelo sistema;
- **JavaScript:** O JavaScript (JS) é uma linguagem leve, interpretada e baseada em objetos e funções de primeira classe, amplamente conhecida como linguagem de scripts em aplicações da internet (Mozilla Foundation, 2021). Foi utilizada como linguagem para criação da lógica de negócio da aplicação por sua versatilidade em ambientes de aplicações *web*.
- **Bootstrap**: desenvolvido pela equipe do Twitter, o Bootstrap é um *framework opensource* compatível com HTML5 e CSS3 que foi criado para auxiliar no desenvolvimento de interfaces *web* responsivas (ZABOT E MATOS, 2020). Será utilizado na criação da interface gráfica da simulação do mecanismo de substituição de páginas.

4 RESULTADOS E DISCUSSÃO

Esta seção apresenta a aplicação *web* implementada, que realiza a simulação da paginação por demanda com substituição de páginas FIFO. O simulador desenvolvido oferece a interação entre o usuário e o mecanismo, o que proporciona ao aluno simular diversas situações e visualizar o resultado de cada uma delas. Dessa forma, espera-se que a aplicação auxilie os alunos a compreender os conceitos estudados em sala de aula.

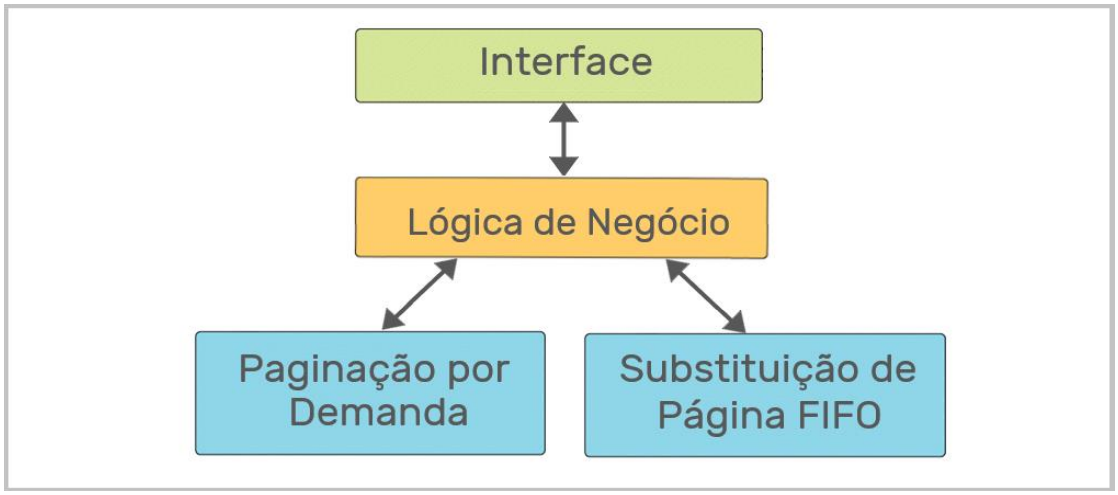
A aplicação é *web*, disponibilizada para acesso na internet sem a necessidade de fazer *download*. O acesso da aplicação pelo navegador dispensa o usuário da necessidade de documentação para instalação e da complexidade do processo de instalação e desinstalação, permitindo que a aplicação seja acessada de qualquer computador conectado à internet.

Nas seções seguintes serão apresentadas a arquitetura do software, com informações sobre a implementação, e as telas do módulo de simulação da paginação com substituição de páginas. As telas foram capturadas na utilização da aplicação em uma simulação para exemplificar o mecanismo de paginação por demanda e o mecanismo de substituição de páginas com algoritmo FIFO. Os exemplos apresentam informações sobre os recursos que permitem ao usuário realizar as simulações.

4.1 ARQUITETURA DO SOFTWARE

Para melhor compreender o funcionamento da aplicação implementada, a figura 10 apresenta a arquitetura do software que consistiu na implementação de três camadas sendo elas a interface, lógica de negócios e a paginação por demanda com substituição de páginas.

Figura 10. Arquitetura da Aplicação



A camada de interface representa o ambiente em que o usuário realiza a interação com o mecanismo. A interface foi implementada utilizando HTML e bootstrap para oferecer a entrada de dados e a visualização do resultado das diferentes simulações criadas pelo usuário. Também foi utilizado javascript para exibir notificações de eventos realizados pelo usuário durante as simulações.

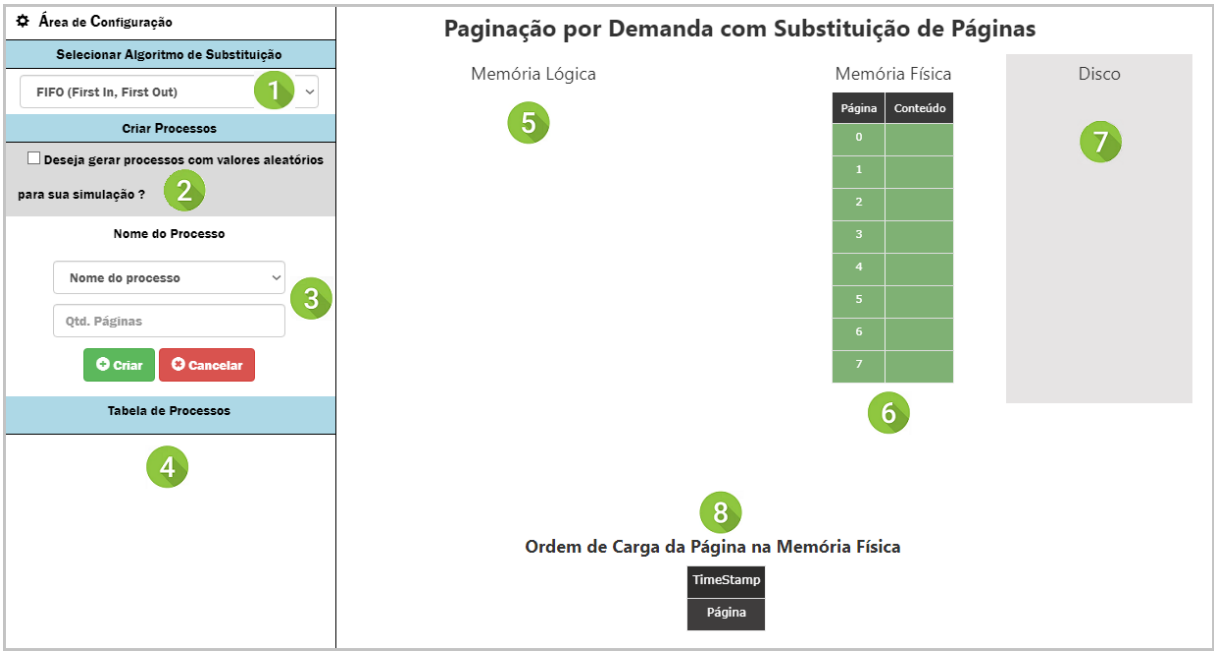
A camada de lógica de negócios é responsável por realizar toda a parte lógica da ferramenta. Esse controle é realizado utilizando o *framework* Angular e o javascript, que faz a intermediação e controle de fluxo entre a interface da aplicação e o algoritmo da paginação por demanda com substituição de páginas.

A paginação por demanda com substituição de páginas executa a simulação que é apresentada na camada interface ao usuário. Os algoritmos implementados recebem os dados e solicitações originadas da interface e, posteriormente, retorna o resultado das simulações para serem apresentadas ao usuário. Na implementação dos algoritmos foram utilizados o angular e o javascript, para criação dos métodos e atributos e na comunicação com a interface.

4.1 Interface da Aplicação

Ao acessar a aplicação *web* do módulo, o usuário terá acesso a tela inicial do simulador de paginação por demanda com substituição de páginas. A tela mostrada na figura 11 é aberta quando o usuário acessa a aplicação, antes de executar qualquer entrada de dados ou simulação.

Figura 11. Tela Inicial



A figura 11 apresenta a área de entrada de dados e a de apresentação das simulações, que estão dispostas da seguinte forma:

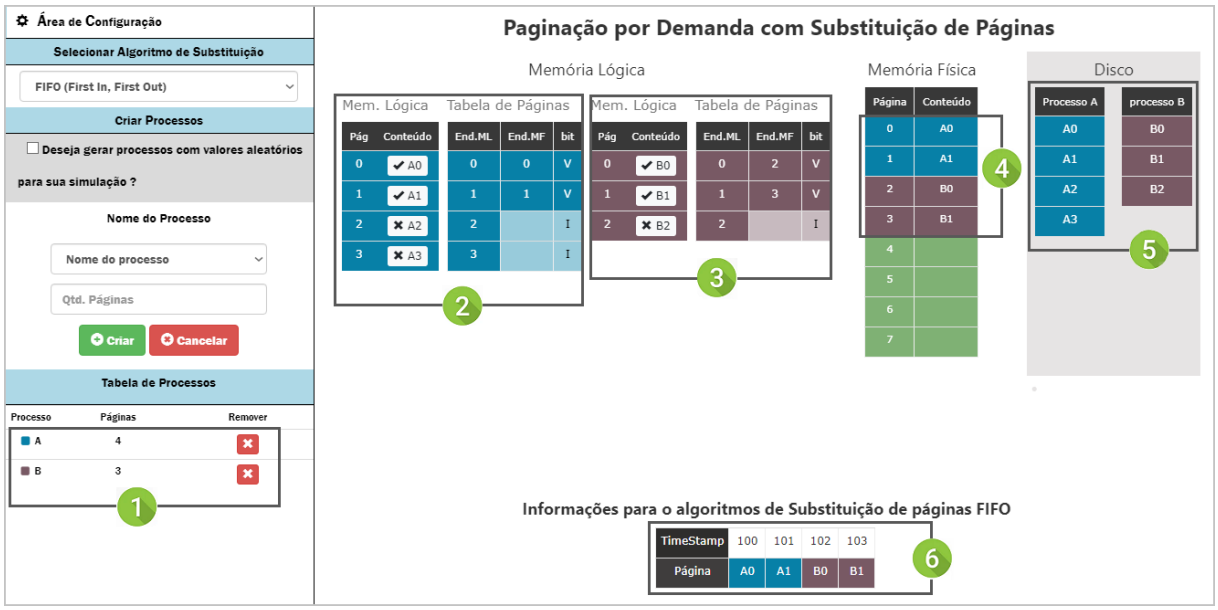
- o número 1 identifica o seletor para escolha do algoritmo de substituição de páginas que será utilizado quando houver a necessidade de substituir as páginas. Atualmente, o módulo possui apenas o algoritmo FIFO implementado, o qual fica selecionado como algoritmo padrão;
- o número 2 identifica a opção de gerar os processos de forma automática com valores aleatórios para que não seja necessário cadastrar manualmente todos os processos usados na simulação. A opção gera aleatoriamente 4 processos contendo de 1 a 4 páginas cada um;
- o número 3 corresponde ao formulário de criação manual dos processos. Para criar o processo o usuário tem à disposição na caixa de combinação os processos A até D. Para cada processo criado o usuário deve inserir a quantidade de páginas de no mínimo 1 e no máximo 4 por processo;
- o número 4 identifica a área em que será apresentada a tabela de processos, após a criação dos processos é apresentada uma tabela com a lista dos processos criados e seus respectivos número de páginas. Ao final de cada entrada da tabela é apresentado um botão que possibilita a exclusão do processo;
- o número 5 apresenta a área de memória lógica dos processos do SO. Nessa área será apresentada a memória lógica e a tabela de páginas de cada um dos processos criados, sendo que cada processo possui sua própria área de memória lógica independente;
- o número 6 apresenta a memória física que contém espaço para 8 páginas. A memória física possui a coluna página que identifica o endereço da página na própria memória e é utilizado pela tabela de páginas para mapear endereços lógicos em físicos, a coluna conteúdo recebe o valor das páginas carregadas;
- o número 7 identifica o disco que é usado como memória virtual . O disco armazena todas as páginas dos processos criados, mantendo as informações até que os processos sejam excluídos.
- por fim, o número 5 identifica a fila FIFO que é uma informação usada no algoritmo de substituição de páginas implementado. Na tabela que ilustra a fila, a linha da página deve mostrar as páginas carregadas na memória física (número 6) e o *timestamp* deve mostrar o valor correspondente ao momento de carga da página, ordenando as páginas de acordo com seu momento de carga na memória. A ordenação é baseada no algoritmo FIFO, em que as páginas mais antigas ficam no início da fila para serem removidas e as páginas novas sendo inseridas ao final da fila. A fila FIFO será utilizada sempre que houver necessidade de substituir uma página.

A interface do módulo de simulação da paginação por demanda com substituição de páginas foi desenvolvida seguindo o padrão dos demais módulos que constituem a plataforma OSLive. Essa padronização ocorre nos elementos usados para entrada e saída dos dados e também nos elementos de apresentação visual empregados no módulo. Tudo isso tem o objetivo de proporcionar a integração e a fácil navegação entre os módulos oferecidos pelo OSLive.

4.2 DEMONSTRAÇÃO DA SIMULAÇÃO

Esta seção apresenta um exemplo do funcionamento da simulação do mecanismo de paginação por demanda. Para isso, na aplicação, foram inseridos dois processos: processo A com 4 páginas e o processo B com 3 páginas, sendo que o resultado da simulação a partir desses dados é apresentado na figura 12.

Figura 12. Paginação por demanda



Na área 1, figura 12, os processos A e B são listados na tabela de processos, ambos apresentam as informações de quantidade de páginas e a opção para remover o processo. Ao criar o processo uma cor é definida para ele, dessa forma, as páginas desse processo podem ser facilmente identificadas durante as simulações.

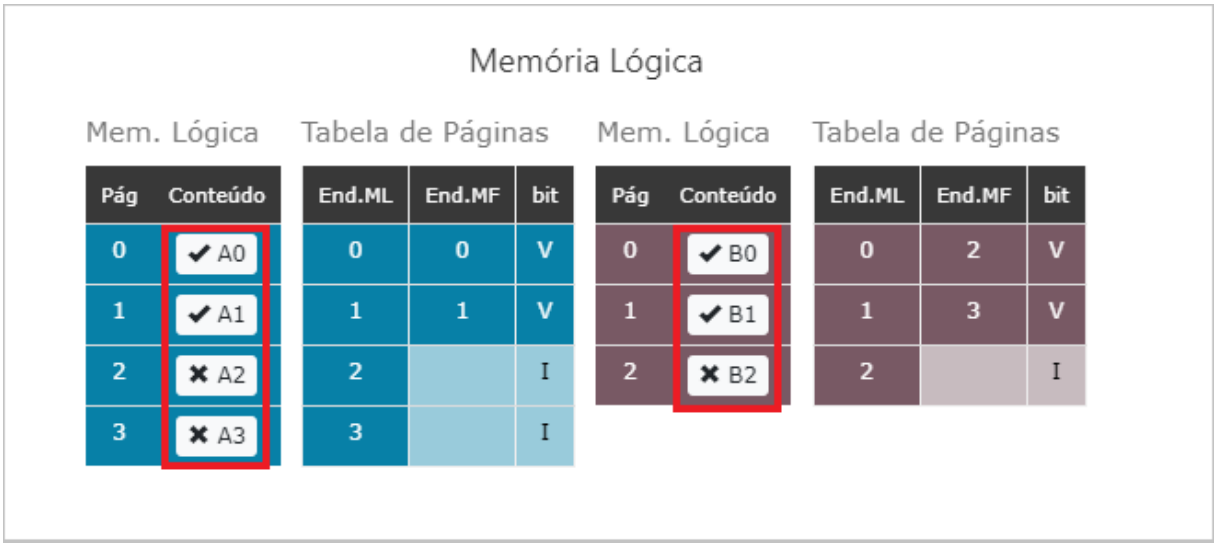
A área 2 apresenta a memória lógica do processo A e a área 3 a do processo B. Tanto o processo A quanto o B tiveram 2 páginas carregadas na memória física (área 4) quando foram criados. A tabela de páginas associada à memória lógica de cada processo fica responsável por relacionar os endereços lógicos com os endereços físicos quando uma página é carregada na memória física, e por sinalizar as páginas que estão somente no disco. Na imagem é possível verificar que as páginas A0 e A1 do processo A e as páginas B0 e B1 do processo B estão com

o bit de verificação marcados como “V”, indicando que foram carregadas na memória física. As demais entradas da tabela de páginas de A e de B estão marcadas como “I”, indicando que as respectivas páginas não foram carregadas na memória física e estão armazenadas apenas no disco.

Na paginação por demanda as páginas são carregadas somente quando são necessárias durante a execução do processo. Conceitualmente, existem duas abordagens que podem ser seguidas pelos SOs ao criar um processo, carregar as páginas iniciais do processo ou não carregar páginas de imediato, sendo que essas vão sendo carregadas de acordo com a demanda. Para o módulo de simulação decidiu-se carregar as 2 primeiras páginas do processo quando ele é criado, que é a forma muito utilizada na implementação da Paginação por Demanda. Seguindo o padrão adotado para a simulação, os processos A e B carregaram apenas as 2 primeiras páginas iniciais na memória física quando foram criados. As páginas que não estão sendo utilizadas pelo processo permanecem com seu conteúdo armazenado apenas no disco (Area 5), o qual armazena o conteúdo de todos os processos quando eles são criados.

Por fim, na área 6, os processos são inseridos em uma fila FIFO que será utilizada pelo algoritmo de substituição de páginas FIFO quando for necessário substituir uma página. A fila é ordenada por ordem de carga, ou seja, a página A0 com timestamp 100 foi a primeira página a ser carregada, por isso está no início da fila, e por ser a página mais antiga em uma eventual falta de página será selecionada para deixar a memória física.

Figura 13. Botões de interação com as páginas



Para gerar mais interação com o usuário, conforme pode ser visto na área destacada na figura 13, a coluna conteúdo da memória lógica dos processos possui um botão identificado com o nome da página. O usuário pode selecionar o botão tanto para carregar quanto para

remover uma determinada página da memória física. Assim, o conceito de paginação por demanda, em que somente páginas efetivamente usadas pelo processo são carregadas na memória, pode ser simulado demonstrando o comportamento do mecanismo quando os processos vão solicitando páginas ou quando vão liberando-as.

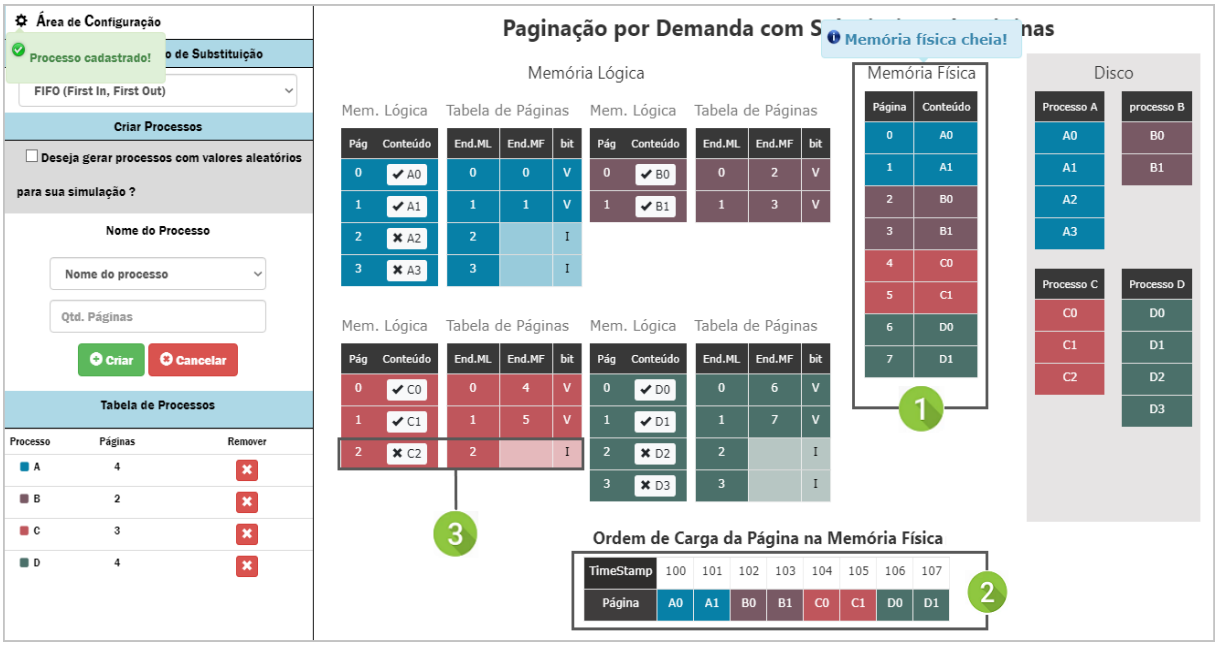
Apresentada a paginação por demanda, a próxima sessão exemplifica a simulação do mecanismo de substituição de página.

4.2.1 Substituição de Páginas FIFO

Se o sistema operacional tentar carregar uma nova página quando a memória estiver cheia, ocorrerá a falta de página. Para solucionar este problema é necessário realizar a substituição de páginas, em que uma página é escolhida como vítima para ser removida da memória, de forma que essa página vítima é selecionada pelo algoritmo de substituição de páginas FIFO. O algoritmo seleciona a página a partir das informações disponibilizadas na fila FIFO, ordenada pela sequência de carga das páginas na memória física.

Para demonstrar uma situação que necessite do algoritmo de substituição de páginas foram criados os seguintes processos: A com 4 páginas; B com 2 páginas, C com 3 páginas e D com 4 páginas. Na simulação, por padrão, quando um processo é criado, as duas primeiras páginas do processo são carregadas na memória física. A figura 14 ilustra como ficou a memória após a criação e a carga dos processos na memória física.

Figura 14. Memória Física Cheia



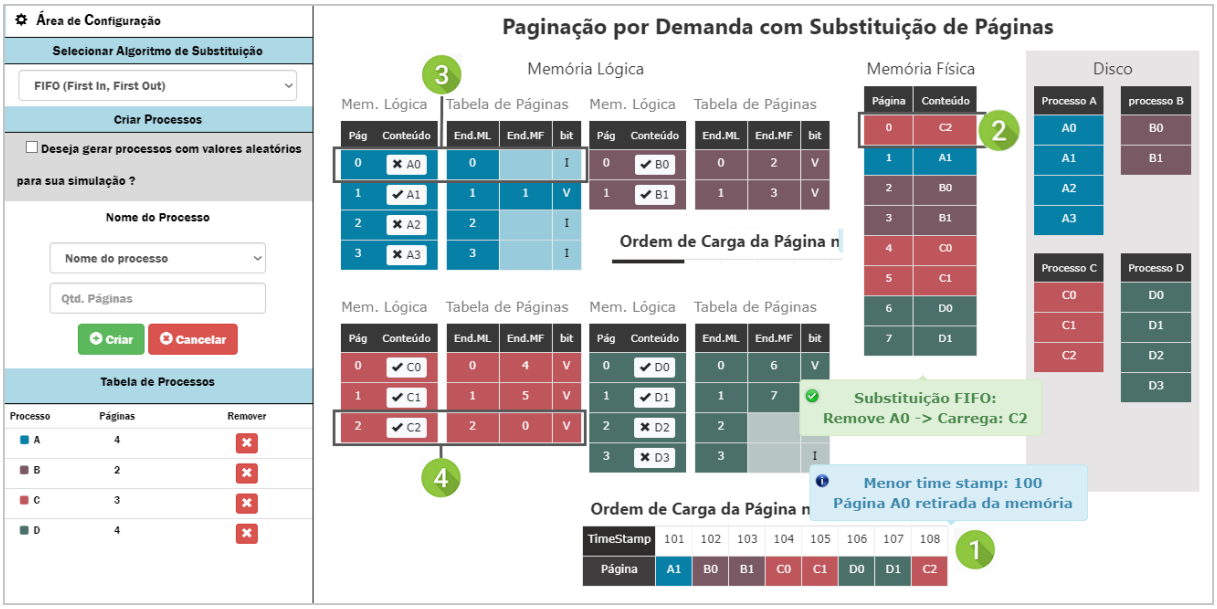
Na figura 14, a área 1 apresenta a memória física com todo o espaço de endereçamento ocupado. Quando a memória física tem seu último endereço ocupado, uma notificação é

mostrada para alertar que a memória física está cheia, nessa situação, caso o usuário opte por inserir uma nova página, ocorrerá a substituição de páginas.

O algoritmo de substituição de páginas FIFO utiliza a fila com a ordem de carga das páginas (área 2) para escolher uma página vítima. Na situação apresentada na figura 14, a página A0 do processo A possui o menor *timestamp* (100), que foi carregada a mais tempo, portanto em uma eventual falta de página ela será escolhida como vítima para ser removida da memória física.

Para simular a substituição de uma página com o algoritmo FIFO a página C2 do processo C (área 3) foi escolhida para ser carregada na memória física. Ao carregar a página, como a memória estava toda ocupada, o mecanismo de substituição de páginas FIFO foi solicitado para escolher uma página vítima e solucionar o problema de falta de página. O resultado da substituição de página pode ser visto na figura 15.

Figura 15. Substituição de página FIFO



Na figura 15, a área 1 apresenta a fila FIFO após a página A0 ser removida da memória e página C2 ser carregada. Conforme a notificação apresentada na área 1 a página A0 foi removida da memória, pois possuía o menor *timestamp*. Por outro lado, a página C2, que causou a falta de páginas, foi inserida no fim da fila FIFO com *timestamp* 108 após ser carregada na memória física.

Em seguida, a área 2 apresenta a página C2 já carregada na memória física, ocupando a página que anteriormente era da página A0 (figura 14). Quando ocorre a substituição da página

FIFO uma notificação, vinculada a memória física, é mostrada indicando que a página A0 foi removida e a página C2 foi carregada na memória física.

Tanto o processo removido quanto o processo carregado na memória física sofrem alterações em suas tabelas de páginas. Na área 3, na tabela de páginas do processo A, a página A0 teve o bit de referência modificado para “I”, enquanto na área 4, a página C2 da tabela de páginas do processo C teve o bit de referência modificado para “V”.

Por meio dos exemplos apresentados, o módulo de paginação por demanda com substituição de páginas torna viável a simulações dos conceitos relacionados a paginação por demanda com substituição de páginas que são vistos na disciplina de sistemas operacionais. Com isso, o aluno poderá utilizar a simulação para praticar os conceitos vistos em sala de aula, sendo essa uma forma de reforçar os conceitos aprendidos e sanar as eventuais dúvidas que surgiram durante o estudo dos conceitos da paginação.

5 CONSIDERAÇÕES FINAIS

O presente trabalho teve como objetivo desenvolver uma aplicação para simular o funcionamento do mecanismo de paginação por demanda com substituição de páginas, algoritmo FIFO, para auxiliar no processo de aprendizagem dos alunos que cursam a disciplina de Sistemas Operacionais. Nesse sentido, o módulo implementado possibilita ao usuário criar diferentes simulações, nas quais são apresentadas a memória lógica, a memória física e a tabela de páginas que mapeia endereços da memória lógica dos processos em endereços da memória física. Além disso, o usuário pode inserir e remover processos a qualquer momento e, em cada uma dessas ações realizadas, a parte gráfica da simulação é atualizada imediatamente, mostrando a situação atual da simulação.

Por meio dos exemplos apresentados neste trabalho, é possível identificar que o módulo simula os principais conceitos do mecanismo, sendo que, o usuário tem liberdade para criar diferentes simulações, de acordo com as entradas realizadas, e repeti-las quantas vezes for necessário para o entender o mecanismo. Assim, espera-se que essa possibilidade de interação do usuário na criação e manipulação dos processos e os recursos visuais disponibilizados durante a simulação sirva de apoio ao professor e ao material didático utilizado pelo aluno.

A aplicação desenvolvida neste trabalho foi utilizada e testada pela especialista da área, professora da disciplina de Sistemas Operacionais dos cursos do departamento de computação do CEULP/ULBRA, que indicou as alterações necessárias para deixar a simulação de acordo com o resultado pretendido. O módulo também foi usado para apresentar os conceitos da paginação por demanda e substituição de páginas aos alunos durante a aula da turma de Sistemas Operacionais do CEULP/ULBRA de 2021/1. Nessa aula, os alunos utilizaram o módulo para realizar uma sequência de ações simulando os principais conceitos presentes na paginação por demanda com substituição de páginas, ao final, os alunos deram um retorno positivo à professora, no qual informaram ter entendido melhor os conceitos ao visualizar as simulações oferecidas pelo módulo.

Apesar de não ter sido realizada uma pesquisa sistematizada e documentada com os alunos, acredita-se que a experiência relatada pela professora demonstra que o uso da simulação será de grande valia, no sentido de que ao produzir suas próprias simulações e acompanhar o resultado o aluno torna mais concretos os conceitos abstratos da paginação.

Na implementação do módulo foi escolhido o algoritmo FIFO para realizar a substituição de páginas, no entanto, há outros algoritmos que realizam a substituição de páginas. Para os trabalhos futuros sugere-se como melhorias: adicionar os demais algoritmos de

substituição, permitir ao usuário salvar as simulações e inserir recursos de animação para deixar a simulação mais intuitiva.

Diante do exposto, espera-se que a inclusão do módulo de paginação por demanda com substituição de páginas na plataforma OSLive, que disponibiliza diferentes módulos das áreas de gerência de processos e gerência de memória, possa complementar o ensino dos conceitos da área de gerência de memória presentes na disciplina de Sistemas Operacionais.

REFERÊNCIAS

JAVASCRIPT. Mozilla Foundation, 2021. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript.Acesso em: 29 Jul. 2021

NOTIFY.JS. **notify.js**, 2021. Disponível em:https://notifyjs.jpillora.com. Acesso em: 30 Jun. 2021.

MEC. Ministério da Educação. Parecer CNE/CES Nº 136/2012, aprovado em 8 de março de 2012. **Diretrizes Curriculares Nacionais para os cursos de graduação em Computação**. Disponível em: https://plataforma.bvirtual.com.br/Leitor/Publicacao/315/pdf/0. Acesso em: 08 Abr. 2020.

MACHADO, Francis Berenger; MAIA, Luiz Paulo. **Arquitetura de Sistemas Operacionais.** 4.ed. Rio de Janeiro: LTC, 2007.

MEDEIROS, A.; MEDEIROS, C. F. D. **Possibilidades e limitações das simulações computacionais no ensino da física**. Revista Brasileira de Ensino de Física, São Paulo, v. 24, 2002. Disponível em: https://www.scielo.br/pdf/rbef/v24n2/a02v24n2.pdf. Acesso em 20 Mai. 2020.

OLIVEIRA, Rômulo Silva de; CARISSIMI, Alexandre da Silva; TOSCANI, Simão Sirineo. **Sistemas Operacionais**. 2. ed. Porto Alegre: Instituto de Informática da UFRGS: Sagra Luzzatto, 2001. 247 p. Disponível em: http://www.romulosilvadeoliveira.eng.br/artigos/Romulo-Carissimi-Simao-Rita2001.pdf Acesso em: 19 Jun. 2020.

OLIVEIRA, Rômulo S.; CARISSIMI, Alexandre S.; TOSCANI, Simão S. **Sistemas Operacionais-Vol. 11: Série Livros Didáticos Informática UFRGS**. Bookman Editora, 2009.Disponível em: https://docplayer.com.br/56547436-Sistemas-operacionais-romulo-silva-de-oliveira-alexandre-da-silva-carissimi-simao-sirineo-toscani.html. Acesso em: 18 Jun. 2020.

SILBERSCHATZ, Abraham; GALVIN, Peter Baer; GAGNE, Greg. **Fundamentos de Sistemas Operacionais**. 8. ed. Rio de Janeiro: Ltc, 2012. 515 p.

TANENBAUM, Andrew S.; BOS, Helbert. **Sistemas Operacionais Modernos**. 3. ed. São Paulo: Pearson, 2016. Tradução de: Jorge Ritter. Disponível em: https://plataforma.bvirtual.com.br/Leitor/Publicacao/36876/pdf/0 [Biblioteca Virtual]. Acesso em: 12 Jun. 2020.

ZABOT, Diego; MATOS, Ecivaldo. **Aplicativos com BootStrap e Angular**: **como desenvolver apps responsivos**. São Paulo: Érica, 2020. Disponível em: https://books.google.com.br/books. Acesso em: 21 Jun. 2019.