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.

Carlos Bruno Freitas Sardinha

DESENVOLVIMENTO DE UM MÓDULO PARA A RESOLUÇÃO DE EXERCÍCIOS SOBRE PAGINAÇÃO POR DEMANDA PARA O AMBIENTE OSLIVE.

Palmas – TO

2022

Carlos Bruno Freitas Sardinha

DESENVOLVIMENTO DE UM MÓDULO PARA A RESOLUÇÃO DE EXERCÍCIOS SOBRE PAGINAÇÃO POR DEMANDA PARA O AMBIENTE OSLIVE.

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

Orientador: Prof. M.e Madianita Bogo Marioti.

Palmas – TO

2022

Carlos Bruno Freitas Sardinha

DESENVOLVIMENTO DE UM MÓDULO PARA A RESOLUÇÃO DE EXERCÍCIOS SOBRE PAGINAÇÃO POR DEMANDA PARA O AMBIENTE OSLIVE.

Projeto de Pesquisa elaborado e apresentado como requisito parcial para aprovação na disciplina de Laboratório de Criação do curso de bacharel em Ciências da Computação pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).

Orientador: Prof. M.e Madianita Bogo Marioti.

Aprovado em: _____/_____/_____

BANCA EXAMINADORA

_____

Prof. M.e Madianita Bogo Mariote

Orientador

Centro Universitário Luterano de Palmas – CEULP

_____

Profº. Esp. Fábio Castro Araújo

Centro Universitário Luterano de Palmas – CEULP

_____

Profº. Esp. Fernanda Pereira Gomes

Centro Universitário Luterano de Palmas – CEULP

Palmas – TO

2022

AGRADECIMENTOS

Primeiramente eu agradeço a Deus, por sempre me abençoar e estar comigo em todos os momentos, por me acompanhar e permitir chegar até aqui. Agradeço aos meus pais e aos meus irmãos pelo incentivo, pelo cuidado, por sempre estarem comigo e me apoiarem nos momentos difíceis. Em destaque a minha mãe, Dulce Freitas da Rocha, pois além de mãe é minha melhor amiga e me alegro que ela possa me ver chegar até aqui. Espero finalmente retribuir os sacrifícios.

Agradeço imensamente aos meus professores, pois foram meus amigos e marcaram muito minha história, meus professores do ensino médio, Bruno Rodrigues Rosa, o motorista da van Neto e o cobrador que me ajudaram quando eu não tinha condições de pagar para ir estudar no IFTO paraíso. Meus professores do IFTO, Fabio Silveira Vidal, Fabio Melo, Elkerlane, Sostenes, Gleys Ially Ramos, Antonio da Luz, Saldanha, Thatiane e tanto outros que peço perdão por não citar todos vocês, mas estes eu tenho um carinho e uma saudosa lembrança. Eu agradeço pela força que compartilharam comigo, a lembrança dos bons diálogos e conselhos me ajudaram em momentos difíceis.

E aos meus queridos professores que eu tive a imensa alegria de conhecer na graduação, minha orientadora Madianita Bogo Mariotti, nossa mãe Mádia, agradeço por ser uma orientadora presente, este trabalho não teria atingido esse nível sem sua participação e empenho, eu agradeço imensamente. Professor Fabiano Fagundes, conhecido Fabigod, seu carinho, rigidez e atenção nos momentos em que tive dúvida e medos, sempre com sábias palavras e uma visão madura das coisas, sem contar é claro na excelência das aulas. Professor Jackson, um homem sábio, calmo e com uma visão única, um prodígio, agradeço os bons conselhos que partilhaste comigo, nos momentos de ansiedade me trouxe calma e serenidade. Aos outros professores agradeço o bom humor, carisma e a alegria de ensinar, pois me marcaram, muito obrigado sempre me lembrarei de todos vocês.

Por fim, a todos que colaboraram direta ou indiretamente com este projeto. Gratidão a todos vocês.

RESUMO

Sardinha, Carlos B.F; Marioti, Madianita B. **Desenvolvimento de um módulo para a resolução de exercícios sobre paginação por demanda para o ambiente OSLive**. 2022. Projeto Tecnológico (Graduação) – Curso de Ciência da Computação, Centro Universitário Luterano de Palmas, Palmas/TO, 2022[2].

O OSLive é um ambiente web que busca auxiliar o processo de ensino e aprendizagem da disciplina de Sistemas Operacionais. Este trabalho apresenta um projeto de inclusão de um módulo de resolução de exercícios sobre paginação por demanda para o ambiente OSLive. O qual visa oferecer exercícios que testam o conhecimento do usuário em relação ao funcionamento da paginação por demanda e algoritmos de substituição de página, como: O algoritmo FCFS, o algoritmo Histórico de Referência e o algoritmo Segunda Chance. A metodologia deste projeto consistiu no estudo dos conceitos referentes à paginação e paginação por demanda e demais conceitos basilares para o devido entendimento do domínio. A partir disso, pôde-se elaborar os exercícios pertinentes ao tema; a estrutura lógica para apresentar o exercícios; a disponibilização de *feedbacks* aos usuários - que podem auxiliar a resolução e entendimento dos algoritmos. Os resultados alcançados ao fim deste trabalho foram o desenvolvimento de um módulo que segue os padrões já estabelecidos no ambiente OSLive e que atenda as funcionalidades definidas. No decorrer deste documento, serão apresentados os exercícios ofertados neste módulo, descrições de seu funcionamento, como o desenvolvimento ocorreu e demais informações pertinentes para o entendimento do módulo.

PALAVRAS-CHAVE: Sistemas Operacionais, Módulo de exercícios sobre paginação por demanda, Módulo de Exercícios.

LISTA DE FIGURAS

Figura 1. Representação do sistema operacional……... 16

Figura 2. Funcionamento da MMU……. 19

Figura 3. Mecanismo básico de paginação…….. 21

Figura 4. Fragmentação Interna em um sistema paginado e segmentado……. 22

Figura 5. Paginação por demanda……. 24

Figura 6. Transferência de uma memória paginada para espaço de disco…… 26

Figura 7. Exemplo de funcionamento do algoritmo FCFS……. 27

Figura 8. Exemplo de funcionamento do algoritmo Histórico de Referência…….. 28

Figura 9.Exemplo de funcionamento do algoritmo Segunda Chance…….. 30

Figura 10. Janelas SOsim……... 32

Figura 11. Tela após simulação do mecanismo da paginação MetroOS……. 33

Figura 12. Sequência do trabalho…….. 35

Figura 13. Estrutura da Aplicação…….. 37

Figura 14. Visão da Estrutura Interna de um módulo OSLive……... 39

Figura 15. Sugestão de Macroestrutura para o OSLive……. 40

Figura 16. Sugestão de Estrutura para os Módulos do OSLive…… 41

Figura 17. Tela Inicial do módulo…… 43

Figura 18. Tela do Exercício Preenchimento De Memória Lógica…… 45

Figura 19. Preenchimento da resposta ao exercício…….. 45

Figura 20. Mensagem de êxito……..……. 46

Figura 21. Mensagem de insucesso……. 47

Figura 22. Tela visualizar resposta……. 48

Figura 23. Tela do Exercício Preencher Memória Física…… 49

Figura 24. Preenchimento da Memória Física…..……..…… 50

Figura 25. Tela do Exercício Determinar Página Vítima..…….. 51

Figura 26. Seleção do Algoritmo de Substituição de Páginas…….. 51

Figura 27. Área do Exercício se Selecionado Algoritmo FCFS……. 52
Figura 28. *According Feedback* Algoritmo FCFS……... 53
Figura 29. Área do Exercício se Selecionado Algoritmo Histórico de Referência. 54
Figura 30. *According Feedback* Algoritmo Histórico de Referência……. 55
Figura 31. Área do Exercício se Selecionado Algoritmo Segunda Chance……... 56
Figura 32. *According Feedback* Algoritmo Segunda Chance…… 57
Figura 33. Classe FCFS…….. 59
Figura 34. Classe Histórico Bit De Referência……. 61
Figura 35. Classe Histórico Bit de Referência……. 62
Figura 36. Classe Segunda Chance…….. 63
Figura 37. Função Segunda Chance……. 64

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CEULP/ULBRA	Centro Universitário Luterano de Palmas
E/S	Entrada e Saída
GP	Gerência de Processos
SO	Sistemas Operacionais
CPU	Unidade Central de Processamento
RAM	Memória de Acesso Aleatório (Randon Access Memory)
MMU	Unidade de Gerenciamento de Memória (Memory Management Unit)
GENTE	Grupo de Estudos em Novas Tecnologias para processos de Ensino e Aprendizagem
FCFS	Primeiro a Chegar Primeiro a ser Servido (First Come First Served)
HTML	Linguagem de Marcação de HiperTexto (HyperText Markup Language)
CSS	Folha de Estilo em Cascatas (Cascading Style Sheet)

SUMÁRIO

1 INTRODUÇÃO **11**

2 REFERENCIAL TEÓRICO **15**

2.1 Sistemas Operacionais **15**

2.1.1 Gerência De Memória 17

2.1.1.1 Paginação 20

2.1.1.2 Paginação por Demanda 23

2.1.1.3 Algoritmo FCFS 26

2.1.1.4 Algoritmo Histórico de Referência 28

2.1.1.5 Algoritmo Segunda Chance 29

2.2 Trabalhos Relacionados **31**

2.2.1 Sosim - Simulador Para O Ensino De Sistemas Operacionais 31

2.2.2 MetroOS 33

3 METODOLOGIA **34**

3.1 Materiais **34**

3.1.1 Ferramentas utilizadas 34

3.1.2 Desenho de estudo 35

4 RESULTADOS E DISCUSSÃO **37**

4.1 Arquitetura Do Software 37

4.2 Estrutura interna da Aplicação 38

4.3 Interface Da Aplicação 42

4.4 Demonstração Do Exercício Preencher Memória Lógica 44

4.5 Demonstração Do Exercício Preencher Memória Física 47

4.6 Demonstração Do Exercício Determinação De Página Vítima 49

4.6.1 Demonstração Do Exercício Com Algoritmo FCFS 51

4.6.2 Demonstração Do Exercício Com Algoritmo Histórico de Bits De Referência 52

4.6.3 Demonstração Do Exercício Com Algoritmo Segunda Chance 55

4.7 Trechos De Código Da Aplicação 57

5 CONSIDERAÇÕES FINAIS **65**

6 REFERÊNCIAS **66**

1 INTRODUÇÃO

A disciplina de Sistemas Operacionais (SOs) é uma das disciplinas obrigatórias ou sugeridas nas matrizes curriculares nacionais dos cursos superiores da área de computação e Informática no Brasil, conforme o Ministério da Educação - MEC (BRASIL, 2012). Assim, o estudo desta disciplina possibilita ao estudante conhecer conceitos, recursos, funções que compõem o funcionamento interno de um SO, como também, a interação do SO com outras aplicações. Esse conhecimento permite desenvolver sistemas mais harmônicos com o funcionamento do SO.

O OSLive é um ambiente web que busca auxiliar o processo de ensino e aprendizagem da disciplina de Sistemas Operacionais. Nessa ferramenta, o usuário tem acesso a diversos módulos que oferecem recursos como simulação, proposição e correção de exercícios, que desafiam e testam a aplicação dos conhecimentos teóricos, entre outros, relacionados aos conceitos relativamente complexos e abstratos abordados na disciplina de SO. O OSLive é uma ferramenta que faz parte do Grupo de Estudos em Novas Tecnologias para Processos de Ensino e Aprendizagem (Gente) do CEULP/ULBRA.

Para Tanenbaum e Bos (2016), um Sistema Operacional (SO) é responsável por gerenciar todos os itens de *hardware* que compõem um computador, como os discos, processadores, memórias e dispositivos de entrada e saída, por exemplo. Desta forma, pode-se dizer que o sistema operacional oferece uma interface amigável na qual é ocultada toda a complexidade contida na intercomunicação entre *hardwares* e seu gerenciamento de recursos, possibilitando a utilização por uma maior gama de usuários.

Dentre os assuntos abordados nessa disciplina está a paginação que, de acordo com Tanenbaum e Bos (2016), 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.

No processo de paginação todas as páginas do programa são carregadas para os quadros da memória física, procedimento que pode ser oneroso no ponto de vista do compartilhamento de memória na multiprogramação, uma vez que apenas alguns recursos seriam utilizados. Assim, para reduzir este ônus há o recurso de paginação por demanda que, baseado em Machado e Maia (2007), é uma estratégia alternativa 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.

Para que essa dinâmica ocorra de forma oculta ao usuário, necessita-se que os algoritmos de substituição de página sejam eficientes, o que implica que devem levar muitos conceitos técnicos e consideração. Por exemplo: Postergação indefinida, Tempo médio de espera, CPU-Bound, Critérios de Prioridade e preempção entre outros. Caso haja ineficiência ou inobservância a esses conceitos aconteceriam travamentos, lentidão entre outros efeitos.

Tendo em vista a complexidade desse mecanismo, que fica oculto aos usuários, o processo de aprendizado desses conceitos, pelos alunos, pode-se tornar um tanto complexo e abstrato. Por esta razão o ambiente OSLive oferta simulação do funcionamento de vários mecanismos e exercícios sobre estes assuntos, a fim de servir de ferramenta de auxílio aos usuários na visualização desses conceitos em funcionamento.

Com base no que foi exposto, fica clara a importância e a complexidade deste assunto. No OSLive, o usuário já possui acesso a diversos módulos que oferecem recursos como simulação, proposição e correção de exercícios, que desafiam e testam a aplicação dos conhecimentos teóricos, entre outros, relacionados aos conceitos relativamente complexos e abstratos abordados na disciplina de SO.

Por essa razão fez-se necessário o desenvolvimento de um módulo de exercícios para que os usuários pudessem estudar de forma ativa o conteúdo de paginação por demanda e algoritmos de substituição de páginas. Haja vista que a resolução de questões permite ao usuário treinar sua capacidade de interpretar e resolver problemas referentes ao tema, tendo um *feedback* sobre a resposta, em caso de erro, de modo a auxiliar no seu processo de estudo. Assim, os exercícios

definidos foram: Preencher a Memória Física, Preencher a tabela de processos com a posição do processo na Memória Física e Determinar a Página vítima nos principais algoritmos de substituição páginas.

No mercado, existem ferramentas que podem ser utilizadas para ensinar conceitos de SO, como: Sistemas operacionais reais didáticos(SORD), que permitem edição e estudo via linha de código e Sistema de Gerenciamento da Segurança Operacional (SGSO), que demonstra de forma visual e/ou animada os conceitos da disciplina (OLIVEIRA et al, 2015). Alguns SGSOs possuem uma abordagem do conteúdo diferente dos livros didáticos mais utilizados atualmente. Outro ponto a se observar é a ausência de um módulo de exercícios na maioria das ferramentas desse segmento.

Pressman e Maxim (2016, p. 108-109) afirmam que qualquer sistema complexo pode ser dividido em módulos (componentes), porém a boa prática de engenharia de *software* demanda mais do que isso. A modularidade deve ser efetiva. Isto é, cada módulo deve se concentrar exclusivamente em um aspecto bem restrito do sistema – deve ser coeso em sua função e/ou apresentar conteúdo bem preciso. Assim, pode-se concluir que um módulo é uma parte do sistema que faz uso de sua arquitetura e recursos, e que é responsável por atividades que satisfazem um assunto bem definido.

Com isso, o desenvolvimento deste módulo foi validado por meio de testes funcionais, como o teste de caixa preta que, segundo Pressman e Maxim (2016, p. 108-109), faz referência a testes realizados na interface do *software*. Um teste caixa preta examina alguns aspectos fundamentais de um sistema, com pouca preocupação em relação à estrutura lógica interna do software. Assim, desenvolvido e validado, o módulo pôde ser integrado à ferramenta.

Para finalizar, vale salientar que a prática docente consoante ao uso das tecnologias da informação e comunicação tende a aproximar os usuários ao conteúdo estudado tornando o ambiente escolar mais interessante. Além disso, o usuário ganha mais uma forma interativa para estudar, na qual pode testar o que aprendeu, como também aprender com *feedbacks* da ferramenta. Ademais, o

professor ganha mais um recurso didático para auxiliar em seu trabalho, seja durante a aula, seja no pós aula.

2 REFERENCIAL TEÓRICO

Nesta seção serão explanados alguns aspectos gerais de um Sistema Operacional e suas quatro áreas: gerência de arquivos, gerência de entrada e saída, gerência de processos e gerência de memória. E, com base no subsistema de gerência de memória, apresentar mais detalhadamente o mecanismo de paginação por demanda, que é o foco deste trabalho.

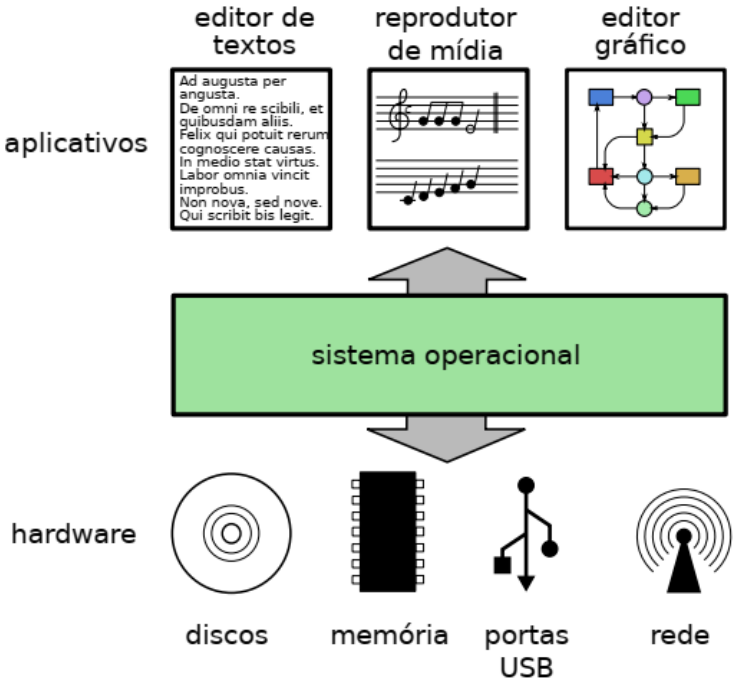
2.1 Sistemas Operacionais

Um Sistema Operacional (SO) é um sistema complexo, pois trata de compatibilizar o acesso aos *hardwares* e permitir que aplicativos utilizem seus recursos mais facilmente. Isso é de grande importância, pois elimina a necessidade de o desenvolvedor preparar sua aplicação para acessar o *hardware* diretamente, ou seja, elimina a necessidade de implementar um subsistema para compatibilizar a aplicação aos diversos tipos de *hardware* que podem compor um computador.

“Acessar os recursos de *hardware* de um sistema de computação pode ser uma tarefa complexa, devido às características específicas de cada dispositivo físico e a complexidade de suas interfaces” (Oliveira, R. A. et al 2015, p. 17). Assim, o SO fornece a interface amigável que possibilita diversos outros sistemas serem desenvolvidos sem a necessidade da equipe de desenvolvimento ter que conhecer os drivers específicos de cada fabricante.

Nesse contexto, Silberschatz et al. (2013, p. 3) afirmam que o Sistema Operacional (SO) fornece a interação entre homem e máquina, atuando como um gerenciador que fica em uma camada entre os aplicativos de usuário e o *hardware* do computador. Para elucidar esse conceito, a Figura 1 exemplifica o sistema operacional como intermediário entre homem e *hardware* do computador.

Figura 1. Representação do sistema operacional



Fonte: Oliveira, R. A. et al (2015)

Como pode ser observado na Figura 1, os softwares instalados tem acesso ao hardware com o intermédio do SO, o qual oculta o acesso ao hardware - questões de drive e gerenciamento de interação entre hardwares -, assim, disponibiliza de forma amigável o acesso aos componentes e recursos de hardware do computador.

Maziero (2013, p.18) afirma que “os programas aplicativos usam o hardware para atingir seus objetivos: ler e armazenar dados, editar e imprimir documentos, navegar na Internet, tocar música etc.. Em um sistema com várias atividades simultâneas, podem surgir conflitos no uso do hardware, quando dois ou mais aplicativos precisam dos mesmos recursos para poder executar”. Com isso, o SO define regras de acesso aos recursos de *hardware* pelos aplicativos, bem como eventuais problemas de disputa de acesso aos recursos.

Com isso, percebe-se a importância e a abrangência de um sistema operacional, que, para o gerenciamento dos recursos computacionais é subdivido em quatro áreas, sendo elas:

- **Gerência de arquivos**: a gerência de arquivos é a camada do SO mais próxima do usuário, pois, em virtude da interface amigável ele pode executar as tarefas básicas sobre arquivos e diretórios;

- **Gerência de entrada e saída**: o SO omite o acesso ao *hardware* e a interrelação entre eles, com o objetivo de entregar uma interface estável e amigável para que o software tenha acesso a entradas vindas do teclado, mouse, pendrive etc., como também realize a saída de dados e informação no monitor, na impressora, entre outros. Assim, o objetivo primeiro do subsistema de entrada e saída é tentar padronizar ao máximo as rotinas de acesso aos periféricos;
- **Gerência de processos**: Processo pode ser entendido basicamente como um programa em execução, que possui uma associação a um espaço de endereçamento de memória. “Um processo é na essência um contêiner que armazena todas as informações necessárias para executar um programa” (Tanenbaum e Bos, 2016, p. 46). A função da gerência de processos é coordenar o uso dos recursos, como CPU, para que vários processos sejam executados simultaneamente. Há uma profunda relação da gerência de processos com a gerência de memória, ou seja, é uma parte importante do SO a harmonia da relação de processo, espaço de endereçamento com a memória física e virtual;
- **Gerência de memória**: é a gerência responsável por assegurar que diversos processos compartilhem o armazenamento da memória RAM, alocando e liberando espaço de memória, de modo que um processo não interfira no espaço de memória do outro.

Como o objetivo deste trabalho está relacionado ao mecanismo de paginação por demanda, que é um conceito contido na área de gerência de memória, a próxima seção aborda esse gerenciamento de forma mais detalhada.

2.1.1 Gerência De Memória

“A memória principal é um componente fundamental em qualquer sistema de computação. Ela constitui o “espaço de trabalho” do sistema, no qual são mantidos os processos, *threads* e bibliotecas compartilhadas, além do próprio núcleo do sistema operacional, com seu código e suas estruturas de dados” (MAZIERO, 2013, p. 175). Haja vista que a memória principal constitui o “espaço de trabalho" de um sistema operacional, a inter-relação do subsistema de memória com os demais subsistemas é imprescindível para o bom funcionamento do sistema computacional. Uma vez que é nesse “espaço de trabalho” que são mantidos os processos, *threads*

e bibliotecas compartilhadas. Além é claro dos próprios códigos e estruturas que permitem o funcionamento de um sistema operacional.

Os SOs atuais tem suporte a multiprogramação, que lhes garante a capacidade de executar mais de um processo ao mesmo tempo. Com a multiprogramação é necessário gerenciar a utilização compartilhada da memória, do processador e dos dispositivos de E/S. Durante o seu ciclo de vida, os processos são mantidos na memória, de onde, por um determinado tempo, cada processo pode utilizar a CPU e/ou um dado dispositivo ou conjunto de dispositivos de E/S, esse procedimento ocorre a uma velocidade que propicia a ilusão de simultaneidade. Nesse contexto, é importante ressaltar que é função do subsistema de gerência de memória assegurar que os diversos processos compartilhem memória de forma segura e eficiente.

Na área de gerência de memória, existem dois tipos de memória: a memória física e a memória lógica. Sendo que a memória física é a memória RAM contida no dispositivo, e a memória lógica, em específico os endereços lógicos, é a memória que os processos conseguem “enxergar”. Segundo Maziero (2013), os endereços físicos são os endereços dos bytes de memória física do computador, definidos pela quantidade de memória disponível na máquina. Já endereços lógicos são os endereços de memória usados pelos processos e pelo sistema operacional e, portanto, usados pelo processador durante a execução. Esses endereços são definidos de acordo com o espaço de endereçamento do processador.

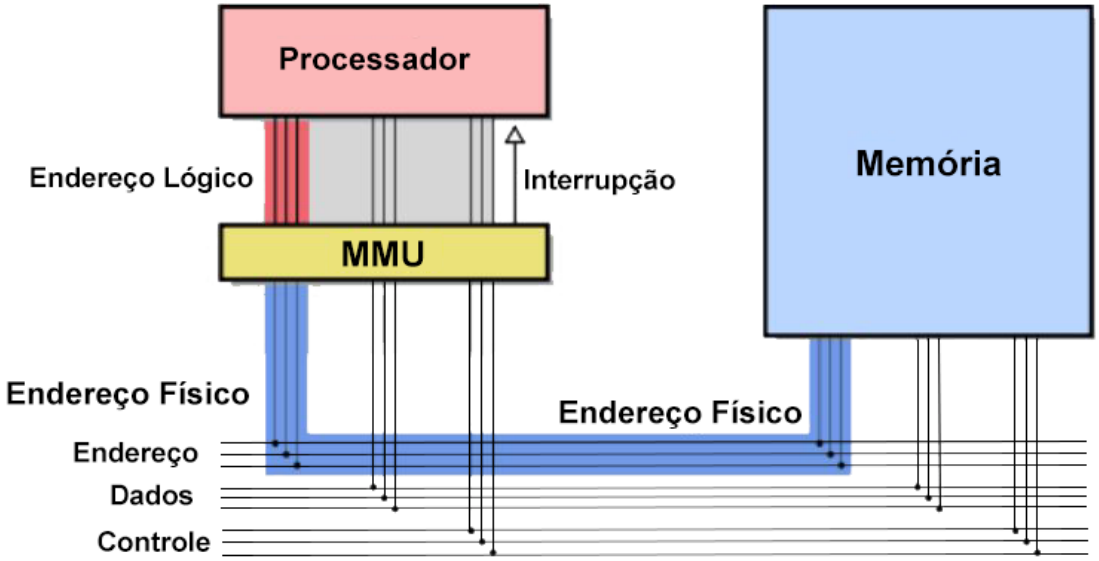
O espaço de endereçamento lógico, criado pelo processador, não é limitado à memória física. Dependendo da configuração da memória RAM, o espaço de endereçamento pode conter diversas áreas válidas e outras inválidas. Segundo Maziero (2013), um endereço lógico gerado pelo processador é válido quando está associado a um endereço físico da memória RAM.

A Unidade de Gerência de Memória (MMU – *Memory Management Unit*) é responsável por fazer o mapeamento de um endereço lógico para físico, ou seja, receber os endereços lógicos do processador, traduzir para endereços físicos associados a eles e retornar a informação. “Na maioria dos processadores atuais, a MMU encontra-se integrada ao chip da própria CPU” (MAZIERO, 2013, p. 164).

Como pode ser observado na Figura 2, a MMU é um dispositivo que vai gerenciar os endereços lógicos gerados pelo processador, para executar os processos, e seus respectivos endereços físicos correspondentes. Caso o

processador tente acessar um endereço inválido, a MMU gera uma interrupção de hardware para notificar essa ocorrência, vale destacar que essas regras de interrupção são definidas pelo SO” (MAZIERO, 2013, p. 164).

Figura 2: Funcionamento da MMU



Fonte: **Maziero (2013)**

Conforme Oliveira R. S et al(2001, p. 9), ”Quando um programa é carregado em uma área de memória maior que o necessário, isso resulta em um desperdício de memória que é chamado de fragmentação interna”. Por exemplo, quando um processo de 80 kbytes é alocado em 5 páginas, sabendo que cada página possui 25 kbytes de memória, na última página haverá um desperdício de memória de 20 kbytes, que não são utilizados pelo processo e não serão disponibilizados para alocação por outros processos.

Outra situação é existir duas partições livres de 25 e 100 Kbytes não contíguas, por exemplo, e ser criado um processo para executar um programa de 110 Kbytes. Neste caso ocorre a denominada fragmentação externa, que em virtude do fluxo de processos alocando e desalocando a memória, pode ocorrer de haver no total espaço disponível, porém o sistema não consegue alocá-lo por ser não contíguo. “Se o programa não pode ser executado devido à forma como a memória é gerenciada, o problema é chamado de fragmentação externa, isto é, memória perdida fora da área ocupada por um processo” Oliveira R. S et al(2001, p. 9).

Existem vários mecanismos de alocação de memória, ou seja, que determinam como se dará a associação entre memória lógica e física. Um deles é o mecanismo de paginação que, por ser base do mecanismo de paginação por demanda que é o foco do trabalho, será descrito com mais detalhes na próxima seção.

2.1.1.1 Paginação

”Na paginação, a memória física é dividida em quadros e a memória lógica é dividida em páginas, do mesmo tamanho e com tamanho fixo.” (SILBERSCHATZ et al, 2013, p. 415). Assim, quando um processo é inicializado, seu tamanho expresso em páginas é examinado, para que haja a alocação de quaisquer quadros de memória disponíveis a todos os quadros do processo. Ressalta-se que há um mapeamento que relaciona um quadro, unidade da memória RAM, a uma página, que é a unidade de memória lógica, sendo esse procedimento efetuado pela MMU.

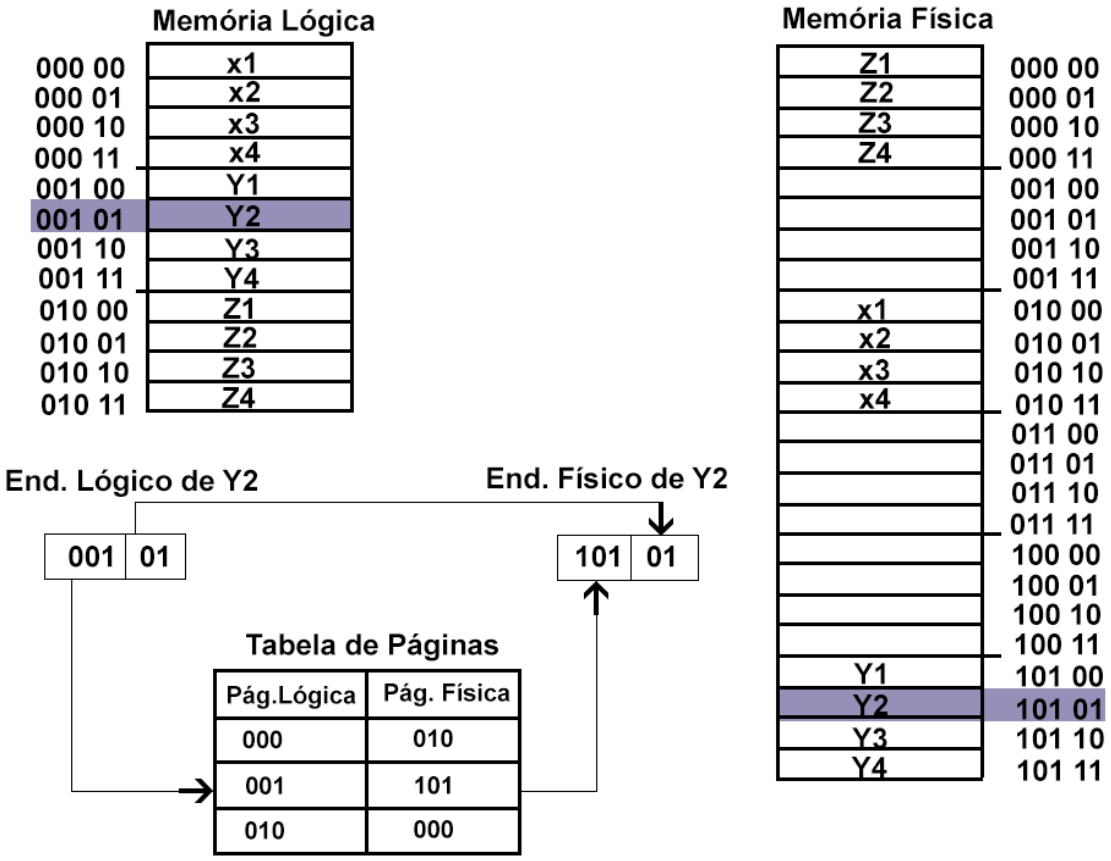
Quando um processo é carregado na memória, busca-se que o espaço que ele irá ocupar seja sequencial, ou seja, que quando seja alocado na memória, ele ocupe um espaço contínuo. Por exemplo, um processo de 20 mb, quando alocado na memória, ocupará do endereço “20” ao “39“ de modo que não ocorra dentro dessa faixa espaços alocados por outros processos, quando isso ocorre, denomina-se alocação contígua.

Todavia, essa possibilidade não ocorrerá sempre que se precise alocar um processo na memória, assim pode ocorrer a alocação não contígua, imaginando o mesmo processo de 20 mb, pode ser alocado com parte do processo ocupando na memória física do endereço 20 ao 29 e outra parte do endereço 40 ao 49, sendo que na memória lógica a alocação será contígua, sendo esta vinculação mapeada pela MMU e registrada na tabela de páginas.

Silberschatz et al (2013, p. 416) afirmam que, na paginação cada endereço gerado pela CPU é dividido em duas partes: um número de página (p) e um deslocamento de página (d). O número de páginas é usado como índice em uma tabela de páginas. Essa tabela de páginas armazena o endereço base das páginas de um determinado processo, juntamente com o seu deslocamento. A partir dessa junção é definido o endereço físico a ser utilizado pela página durante sua execução, sendo este endereço verificado pela MMU.

A seguir, a Figura 3 apresenta um exemplo do mecanismo de paginação, mostrando a memória lógica de um processo de 12 bytes, a tabela de páginas e a memória física.

Figura 3: Mecanismo básico de paginação



Fonte: **Oliveira, R. S et al (2001)**

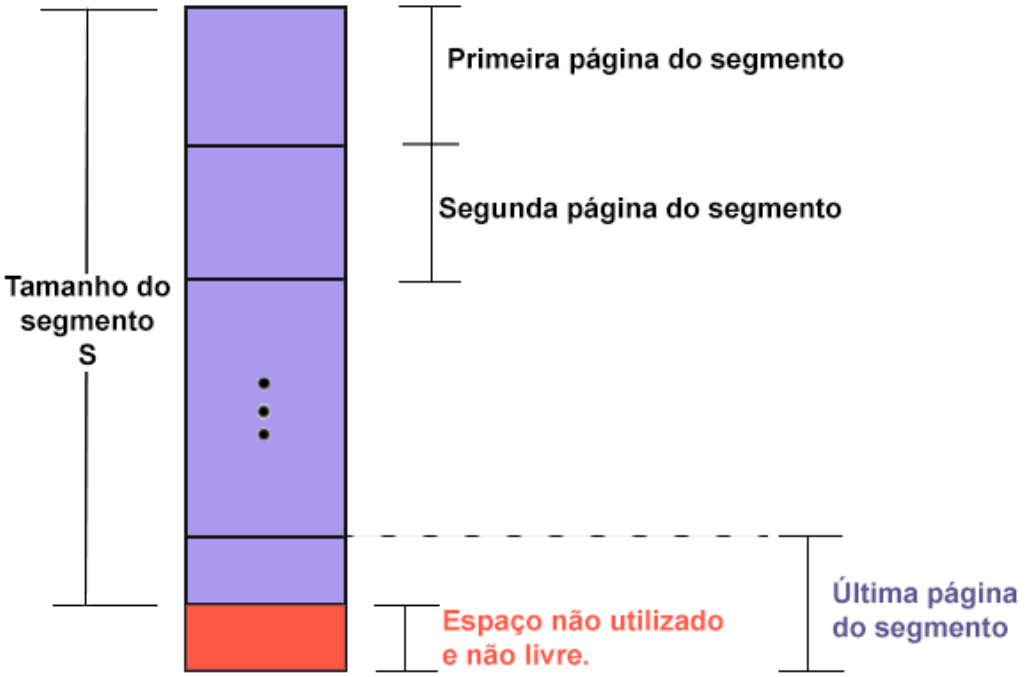
No exemplo apresentado na Figura 3, a memória física é composta por 24 bytes, dividida em quadros com 4 bytes cada, sendo que as páginas lógicas terão tamanho equivalente. Neste caso, como o processo tem 12 bytes, há 6 páginas. Por isso, o endereço da memória física gerado pela CPU é composto por endereçamento lógico e físico, no exemplo da Figura acima, os 3 primeiros bits indicam o número do quadro físico e os dois últimos bits indicam o deslocamento dentro da página física (quadro).

Conforme visto na Figura 3, na tabela de páginas é feita a relação entre os endereços das páginas e quadros, ou seja, entre o endereço lógico e o endereço físico. O endereço base, apresentado na tabela, é combinado com o deslocamento de página para definir o endereço do dado.

Também, pode-se observar, na Figura 3, como o endereço lógico do byte Y2 é vinculado ao endereço físico correspondente. É importante destacar que o endereço lógico é composto pelo endereço da página lógica e deslocamento (001 01), e que o endereço físico é o número do quadro correspondente mais deslocamento (101 01).

Deve-se atentar que o mecanismo de paginação visa eliminar a fragmentação externa, sendo passível a ocorrência de fragmentação interna. Nesse mecanismo, o processo é expresso em páginas que possuem tamanho fixo, e pode acontecer da última página não ser totalmente preenchida, conforme observado na Figura 4, ocorrendo a fragmentação interna. A não ocorrência de fragmentação externa se dá pelo fato de cada quadro estar associado inteiramente a uma página e, como a alocação da memória é não contígua, independe da localização do quadro na memória.

Figura 4: Fragmentação Interna em um sistema paginado e segmentado.



Fonte: Deitel e Choffnes (1999)

Na paginação, o programa a ser executado deve ser todo carregado para a memória física, mesmo que o programa possua partes que raramente, ou nunca, sejam executadas. Assim, há situações em que os programas não precisam estar totalmente carregados, o que é possível usando a memória virtual, que permite a

execução de programas que estejam parcialmente carregados usando o HD como memória auxiliar.

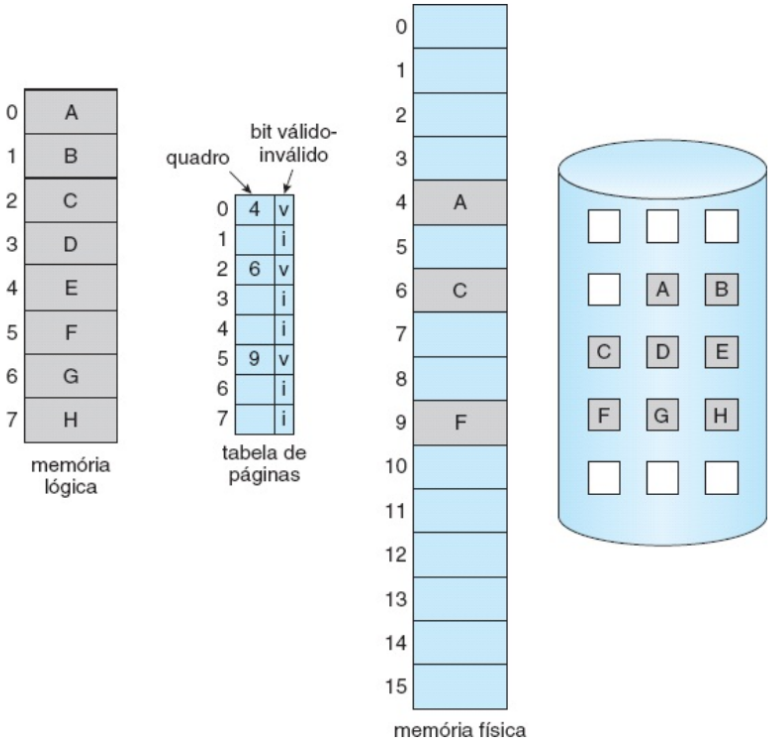
Nesses casos, o espaço de endereçamento lógico poderá ser maior que o espaço de endereçamento físico, pois o processo não precisará estar completamente carregado na RAM. Um mecanismo que implementa memória virtual é a paginação por demanda, o qual será explicado mais detalhadamente na próxima seção.

2.1.1.2 Paginação por Demanda

Conforme Oliveira, R. S et al (2001), o mecanismo de paginação por demanda tem como base o mecanismo de paginação, mas difere da paginação por carregar para a memória física apenas as páginas lógicas necessárias para o funcionamento do *software*, deixando que as demais páginas fiquem em disco - denominada memória secundária ou ainda memória de retaguarda - sendo carregadas na memória principal apenas quando necessárias. Resumidamente, podem haver páginas que, por não terem sido utilizadas na execução, nunca terão alocado recursos de memória.

Esse procedimento é efetuado por um paginador (*pager*) que, segundo Silberschatz et al (2013, p. 453), “quando um processo está para ser inserido na memória, o paginador avalia as páginas que serão usadas”. Assim, o paginador ao invés de inserir o processo completo na memória, insere apenas as páginas mais usadas na memória principal, de forma que as demais podem ocupar a memória secundária, o que diminui a ocupação da memória principal.

Figura 5: Paginação por demanda



Fonte: Silberschatz et al (2013, p. 456)

A Figura 5 apresenta um exemplo do mecanismo de paginação por demanda, em que percebe-se o mapeamento das páginas e quadros, semelhante ao mecanismo de paginação, com o diferencial de indicar se o quadro está na memória física - válido - ou se está na apenas na memória virtual, inválido. De acordo com Oliveira, R. S et al (2001), “um bit válido/inválido na tabela de páginas é usado para indicar quais páginas lógicas foram carregadas” na memória física. Assim, uma página marcada na tabela de páginas como inválida pode não ter sido carregada ou pode estar fora do espaço de endereçamento lógico do processo.

Com essa implementação, apenas as partes necessárias ou solicitadas do processo ocupam a memória principal, enquanto as demais aguardam serem solicitadas. Segundo Oliveira, R. S et al (2001), quando um processo tenta acessar uma página marcada como inválida ocorre uma interrupção por falta de página (*page fault*), de forma que o SO executa as seguintes ações:

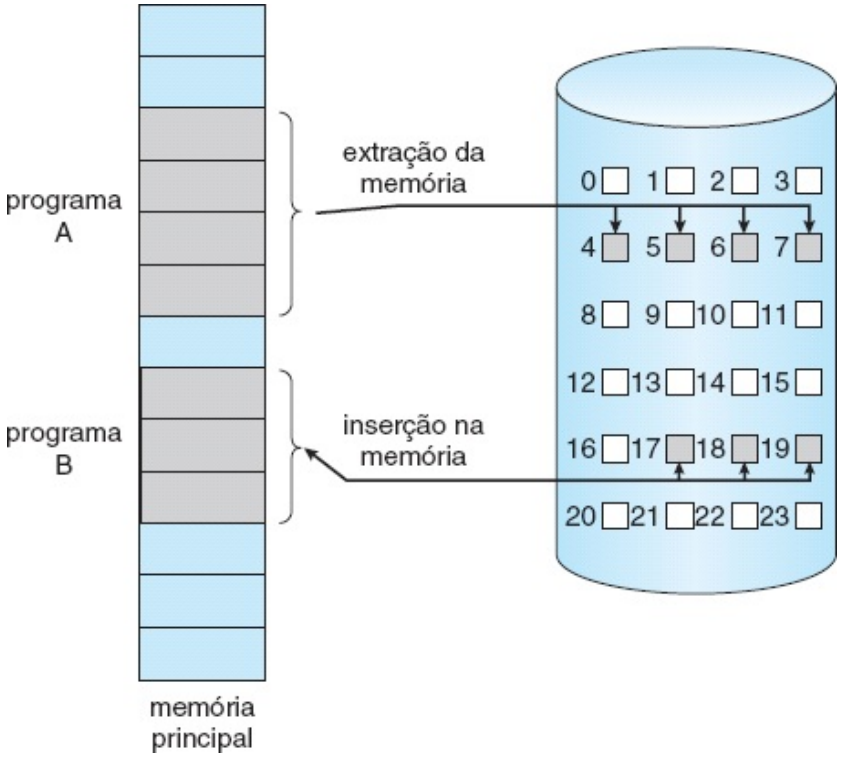
- o processo que gerou a interrupção de falta de página é suspenso, seu descritor de processo é removido da fila do processador e inserido em uma fila especial, a "fila dos processos esperando página lógica";
- uma página física livre deve ser alocada;
- a página lógica acessada deve ser localizada no disco;

> - uma operação de leitura do disco deve ser solicitada, indicando o endereço da página lógica no disco e o endereço da página física alocada;
> - a gerência do processador pode então selecionar outro processo para executar. Quando a operação de leitura do disco for concluída, a gerência de memória deve concluir o atendimento à falta de página realizando as seguintes ações;
> - a tabela de páginas do processo é corrigida para indicar que a página lógica causadora da interrupção é agora válida e está na página física que fora alocada antes;
> - o descritor do processo é retirado da "fila dos processos esperando página lógica" e colocado na fila do processador.

No entanto, pode ocorrer que a memória física esteja cheia, neste caso, será necessário realizar uma substituição de páginas para liberar quadros, ou seja, copiar os dados que estão na memória física, na quantidade de páginas necessárias, para o disco, liberando espaço. Isso permitirá que as páginas solicitadas para a execução sejam carregadas e associadas aos quadros da memória livres. As páginas que são copiadas para o disco dá-se o nome de página vítima, e a escolha das páginas vítimas é determinada por um dado algoritmo de escalonamento, como FIFO, RR ou outro.

Para resumir, o mecanismo de paginação por demanda permite essa dinâmica de inserção e extração de quadros e páginas da memória quando necessário, conforme demonstrado na Figura 6, propiciando um ambiente favorável à multiprogramação.

Figura 6: Transferência de uma memória paginada para espaço de disco.



Fonte: **Silberschatz et al (2013, p. 454)**

Vale acrescentar que além do bit válido/inválido contido na tabela de páginas, essa pode conter outros bits que, embora não sejam necessários, favorecem a execução da paginação por demanda, por exemplo: bit de sujeira, que indica se o conteúdo de uma página foi alterado durante o processo de execução - se alterada deve ser salva-, um bit que indique bloqueio, indicando que essa página não pode ser escolhida como vítima caso outro processo precise de espaço; entre outros que favoreçam o funcionamento do mecanismo.

2.1.1.3 Algoritmo FCFS

No algoritmo FCFS (First Come First Served) ou O Primeiro a Chegar é o Primeiro a ser Servido, escolhe-se a página que está há mais tempo na memória como vítima. As páginas são ordenadas em ordem crescente de Timestamp, ou seja, a página que está há mais tempo na memória fica na frente da fila. Assim, a página vítima será a primeira da fila.

Figura 7 - Exemplo de funcionamento do algoritmo FCFS

Na Figura 7, é apresentado três momentos da memória física, sendo utilizado o algoritmo FCFS. No primeiro instante a memória está cheia, e a página C2 é solicitada, então o algoritmo determina a página que está a maior tempo na memória, ou seja, possui o menor *TimeStamp*, sendo identificada dessa maneira a página A0, a partir desse momento ocorre a substituição, a página A0 perde o espaço na memória e a página C2 passa a ocupar este espaço, com um novo TimeStamp.

No segundo instante, a página D2 solicita memória, o algoritmo identifica a página A1 como a com menor TimeStamp, e portanto efetua a substituição, a página A1 perde espaço na memória física, e D2 é alocada neste espaço com um novo TimeStamp. No instante 3, é apresentada a alocação da memória resultante dessas substituições.

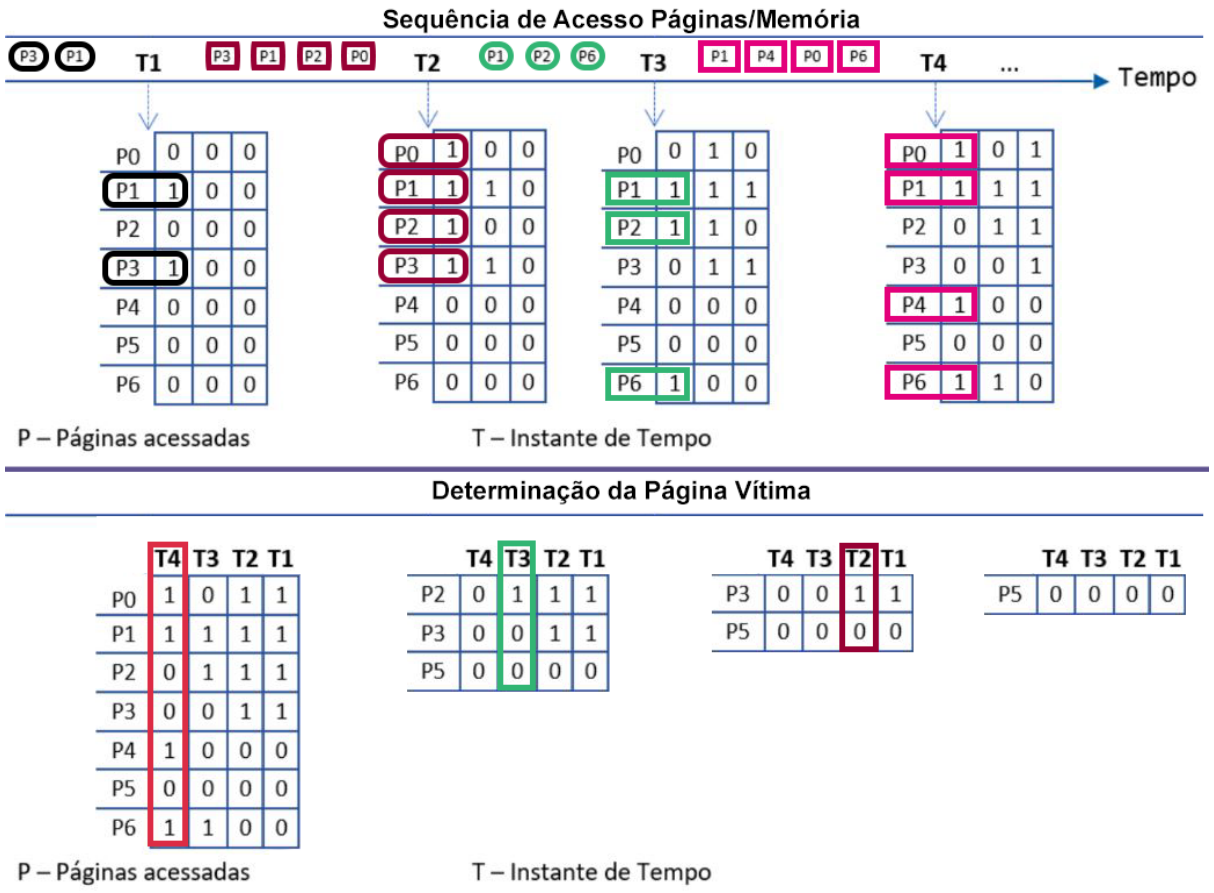
Embora seja de simples implementação, este algoritmo não atenta a prioridade das páginas, como por exemplo as páginas do sistema operacional ou demais processos críticos. Dessa maneira é possível que ocorram travamentos ou falhas caso ele fosse implementado sem acréscimo a observação de outros conceitos, como o antes citado.

2.1.1.4 Algoritmo Histórico de Referência

No algoritmo de Histórico de Referência, guarda-se um histórico dos bits de referência que possibilitará a identificação do processo que está a mais tempo sem ser acessado e que liberará a memória RAM. Quando o bit de referência está igual a 1 indica que a página foi acessada entre os tempos Ti-1 e Ti - por exemplo, na imagem P0 está com bit 1 em T4 porque foi acessado entre T3 (i-1) e T4 (i).

Para escolha da página vítima, verifica-se o último Tempo/Momento (T4 na imagem), a página com bit de referência 0 (não foi acessada depois do tempo i-1, no caso T3) e é escolhida como vítima. No caso de empate, repete-se o processo no Tempo i-1 (no caso T3) e persiste até que se encontre apenas uma página com bit 0. Se todo o histórico for percorrido e ainda houver empate, a página que tiver o *TimeStamp* menor (aquela que está a mais tempo na memória) será a página vítima escolhida.

Figura 8 - Exemplo de funcionamento do algoritmo Histórico de Referência.



Na Figura 8, durante a sequência de acessos às páginas é mantido um histórico, que indica as solicitações das páginas durante um espaço de tempo, neste caso foram representados 4 momentos. A partir disso, caso a memória esteja cheia, far-se-á uma verificação de qual página foi menos solicitada. Assim verificando do momento mais recente T4 até o momento mais anterior armazenado, neste caso T1.

No exemplo da Figura 8, durante a Determinação da página vítima há uma verificação de quais páginas não foram acessadas no momento T4, sendo identificadas 3 páginas P2, P3 e P5. Neste caso se regride em mais um momento, ou seja o momento T3 sendo identificada quais páginas não foram acessadas neste momento, duas páginas foram selecionadas P3 e P5. Neste caso regride mais um momento no tempo, ou seja T2 é analisado e com isso é identificado que apenas a página P5 não foi acessada, caso o empate tivesse persistido, seria analisado o momento T1, e se ainda assim houvesse o empate seria determinado como página vítima aquela página que possuísse o menor *TimeStamp*.

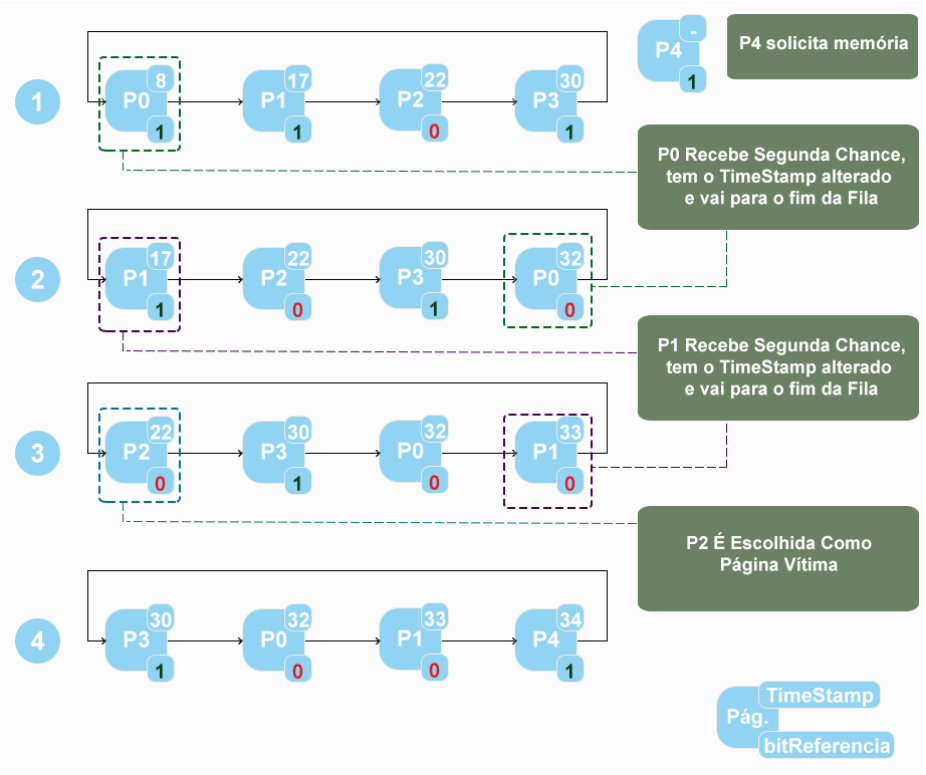
2.1.1.5 Algoritmo Segunda Chance

No algoritmo de Segunda Chance, as páginas são percorridas de forma circular em uma fila ordenada por *Timestamp* (ordem crescente, com mais antigo na frente). São associados bits de referência a cada uma das páginas, o bit de referência 1 indica que foram acessadas recentemente.

No processo de escolha da página vítima, percorre-se a fila e verifica-se o bit de referência associado à página, de forma que as páginas que têm o bit de referência igual a 1 recebem uma segunda chance. A primeira página que tiver o bit de referência 0 (não foi acessada recentemente) é escolhida como vítima.

As páginas que receberam segunda chance têm seu bit de referência alterado para 0, sua marca de tempo (*TimeStamp*) é ajustada para o momento atual e vão para o final da fila (simulando uma nova página na memória).

Figura 9 - Exemplo de funcionamento do algoritmo Segunda Chance



Na Figura 9, são apresentados quatro momentos da memória física, cada momento é representado pelo número contido dentro do círculo no lado esquerdo. Já no canto inferior direito, há uma legenda indicando o significado da representação das páginas, em que no canto direito superior de cada página, tem-se o TimeStamp, e no canto direito inferior é indicado o bit de referência. Então em quatro momentos da memória física é demonstrado o funcionamento do algoritmo Segunda Chance.

No primeiro momento a memória está cheia e o processo P4 solicita memória. Assim a página que está em primeiro lugar na fila P1 e a que possui o menor TimeStamp, pois está a mais tempo na memória. Então tem seu bit de referência verificado, como é igual a 1, a página recebe segunda chance. Dessa maneira, vai para o fim da fila, tendo seu bit de referência zerado, e recebe um novo TimeStamp, assim conclui-se o momento um e dá se início ao momento 2.

No momento 2, como a página P2 foi para o fim da fila, tem-se a página P1 no topo da fila, de igual maneira, por estar mais tempo na memória, possuir o menor TimeStamp. Por isso, a página tem seu bit de referência verificado, como é igual a 1, recebe a segunda chance, sendo movida para o fim da fila e tendo seu bit de referência zerado e recebendo um novo TimeStamp. Dessa forma conclui-se o momento 2 e dá-se início ao momento 3.

No momento 3, a página P2 está no topo da fila, já que possui o menor TimeStamp, e por esta razão tem seu bit de referência verificado. Mas neste caso ocorre uma situação diferente, posto que o bit de referência é igual a 0, em virtude disso, essa página é escolhida como a página vítima. O que reverbera na página P2 perder seu espaço de memória física, e a página P4 ser alocada nele, sendo adicionada no final da fila com um novo *TimeStamp* e com seu bit de referência sendo igual a 1. Resultando na visualização que temos no momento 4.

2.2 Trabalhos Relacionados

Atualmente, há duas categorias de ferramentas que são utilizadas para o ensino de sistemas operacionais: "SORD - Sistemas Operacionais Reais Didáticos e SGSO - Simuladores Genéricos de Sistemas Operacionais." (OLIVEIRA et al, 2015). Nesta seção serão apresentados aspectos gerais e considerações sobre duas ferramentas semelhantes ao OSLive, os SGSOs: SOsim e MetroOS. Essas ferramentas tem por objetivo prover o auxílio no aprendizado dos mecanismos básicos de um sistema operacional.

2.2.1 SOsim - Simulador Para O Ensino De Sistemas Operacionais

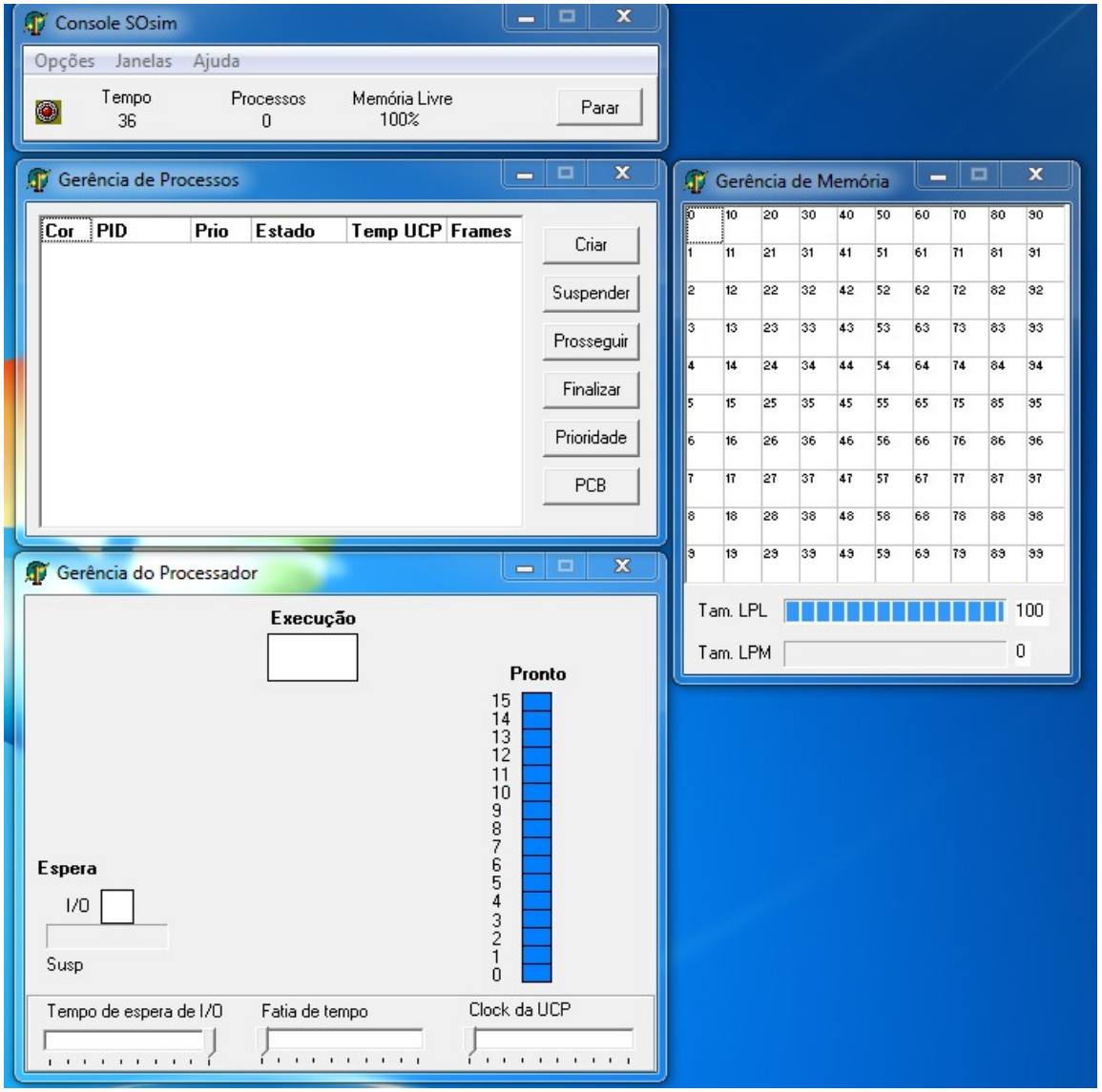
O SOsim, criado pelo professor Me. Luiz Paulo Maia - sob a orientação do prof. Ageu Pacheco -, é uma ferramenta voltada para o auxílio no processo de aprendizagem dos usuários na matéria de SO. Essa ferramenta apresenta conceitos de sistemas operacionais como: gerência de processos, gerência do processador e gerência de memória. Por meio de uma interface gráfica composta por janelas separadas por assunto é possível simular sobre os respectivos temas.

O site do SOsim diz que,

> O sistema permite ao professor apresentar os conceitos e mecanismos de um sistema operacional multiprogramável e/ou multitarefa, como *Unix*, *OpenVMS* e *Windows*, de forma simples e animada. O simulador também permite visualizar os conceitos de multiprogramação, processos e suas mudanças de estado, gerência do processador (escalonamento) e a gerência memória. A partir das configurações do app, é possível selecionar por exemplo diferentes políticas de gerência de memória, e alterar o funcionamento do simulador. Desta forma, o aluno tem a oportunidade de visualizar os conceitos teóricos apresentados em aula de forma simples e animada.

A visão das Janelas relacionadas às funcionalidade do Sosim é apresentada na Figura 10.

Figura 10 - Janelas SOsim



Fonte: Benacchio, J. H. Q

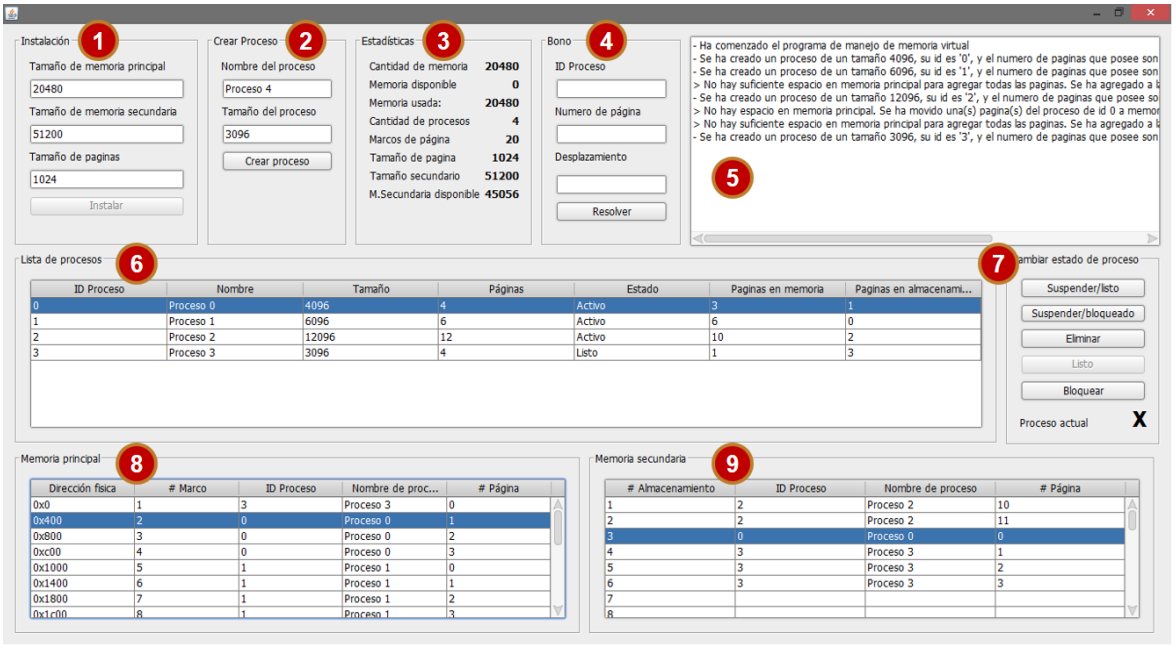
Em suma, é uma ferramenta interessante, pois se trata de um simulador com o qual o usuário pode testar e visualizar conceitos, ou seja, pode simular situações reais e observar as consequências. Porém, não é compatível com todas as plataformas, pois foi desenvolvido especificamente para plataformas *windows*. Outra ressalva é a ausência de um módulo de exercícios que estimule os usuários a interpretar e resolver questões sobre os referidos assuntos. Por fim, nota-se também a ausência de uma versão online, a qual viabiliza sua utilização em todas as plataformas. Em síntese, é uma ferramenta que propicia um auxílio ao aprendizado,

mas que poderia ser mais trabalhada e expandida a fim de atender melhor às necessidades dos estudantes.

2.2.2 MetroOS

A ferramenta MetroOS trabalha com a simulação do mecanismo de paginação. Nessa ferramenta há um detalhamento das informações pertinentes ao funcionamento da paginação, como visto na Figura 11, onde a numeração significa respectivamente: Instalar memória física; Criar processo; Estatísticas da memória; Apresentar informações extra do processo indicado; Área de notificações; Lista de processos; Ações para realizar com os processos; Representação gráfica da memória física primária; Representação gráfica da memória física secundária.

Figura 11. Tela após simulação do mecanismo da paginação MetroOS



Fonte: Santos, A.M. (2017, p. 23)

Embora muito detalhada e interessante, essa ferramenta mantém as ausências da ferramenta anterior que são: não é compatível com todas as plataformas, pois foi desenvolvido especificamente para plataformas *windows*; e não oferece um módulo de exercícios que estimule os usuários a interpretar e resolver questões sobre os referidos assuntos. Somando-se a isso, a MetroOS trabalha apenas com paginação simples, sem o mecanismo de memória virtual. Nessa ferramenta, segundo Santos, A.M. (2017, p. 27), ainda nota-se que, não são apresentados endereços lógicos e nem a tabela de páginas, consequentemente, não

consegue seguir com fidelidade o material didático. Vale salientar, portanto, que a interface é adequada ao contexto, pois é bem instrutiva quanto às informações sobre os processos.

3 METODOLOGIA

Nesta seção serão apresentados os materiais, as ferramentas e os métodos utilizados para compor o fluxo deste trabalho, que intenta elaborar um módulo de resolução de exercícios sobre paginação por demanda, para que possa ser agregado a ambiente OSLive, que já possui um módulo de simulação de paginação por demanda.

3.1 Materiais

As ferramentas que foram utilizadas, tanto para o desenvolvimento da interface quanto do funcionamento do módulo de questões, estão citadas e descritas abaixo. Além do desenho do estudo que ilustra as etapas desenvolvidas no projeto.

3.1.1 Ferramentas utilizadas

O Front-end é a parte gráfica do desenvolvimento, ou seja, a interação direta com o usuário. Nele são exibidas as informações e os meios que vão gerar a interação entre a plataforma e o usuário.

Para criar o Front-end deste trabalho das alternativas disponíveis no mercado, foi escolhido o *framework* Angular, que é uma plataforma de desenvolvimento construída em TypeScript, um *framework* baseado em componentes feito para aplicações Web. (ANGULAR, 2022). Para este desenvolvimento foi utilizada a linguagem de programação TypeScript, junto ao Angular, para fazer a configuração das funcionalidades da plataforma.

Também foram utilizados o *framework* Bootstrap que é um *framework open source* criado em 2010 pela equipe da rede social *Twitter*, desenvolvida em para ser um guia de estilos para interfaces de aplicações web (BOOTSTRAP, 2021). Para dar uma melhor interação do usuário com a plataforma, e garantir que o projeto siga os padrões mais comuns do mercado, o CSS que será usado para estilizar alguns detalhes a aparência do site e que, junto ao Bootstrap, oferecerão uma caráter mais

amigável a plataforma facilitando a navegação do usuário, além do HTML5 que será usado para criar o básico das páginas da plataforma.

3.1.2 Desenho de estudo

Os métodos utilizados para o desenvolvimento deste projeto, a criação do módulo de exercícios de paginação por demanda, estão representados no fluxograma de atividades da Figura 12, conforme foram executados.

Figura 12: Sequência do trabalho

Na primeira etapa, foi realizada uma reunião com a especialista do domínio onde foram explicados os objetivos a serem alcançados pelo módulo de exercícios sobre paginação por demanda.

Na segunda etapa foi realizada uma pesquisa sobre os conceitos de SO referentes ao tema bem como um estudo de questões a fim de obter os conhecimentos necessários para o desenvolvimento do projeto, bem como as tecnologias que seriam utilizadas no Front-End. Na terceira etapa, foram definidos os requisitos mínimos e padrões de tela a serem atendidos pelo módulo, bem como a elaboração dos exercícios mínimos a serem ofertados pelo módulo.

Na quarta etapa, ocorreu a implementação do módulo, que seguiu os passos descritos no item (4) da Figura 12. Como apresentado na Figura acima, essa etapa contemplou o desenvolvimento, os testes e a validação dos requisitos determinados, na terceira etapa, pela especialista do domínio.

Desse modo, na etapa de desenvolvimento um conjunto de funcionalidades foram selecionadas previamente para serem desenvolvidas. Gerando assim um protótipo que seria submetido às seguintes sub etapas de testes pelo desenvolvedor, para que pudesse ser apresentado à especialista do domínio para validação e/ou indicação de correções ou adequações.

Na etapa de testes, foram testados: Entradas de dados; Mensagens explicativas; Cores para representar componentes; Fluxo de dados; Estrutura do projeto e Mensagens a serem apresentadas em caso de erro. Pela especialista do domínio, foram testados, a usabilidade da aplicação; Verificação das informações de entrada e saída; Representação correta do funcionamento dos algoritmos e corretude dos resultados.

Já na etapa de validação, ocorreram reuniões, ora com o desenvolvedor apresentando o que fôra desenvolvido, ora com a especialista testando individualmente o sistema e fazendo sua avaliação e levantamento de correções e adequações. Assim, caso a funcionalidade fosse aprovada, outra era escolhida para ser desenvolvida, até que se concluísse a lista de requisitos determinados.

Na quinta etapa, foi realizado o processo de escrita do projeto, e a definição das então as ferramentas que foram usadas para o desenvolvimento da plataforma, além da elaboração de esboços para a interface da plataforma.

4 RESULTADOS E DISCUSSÃO

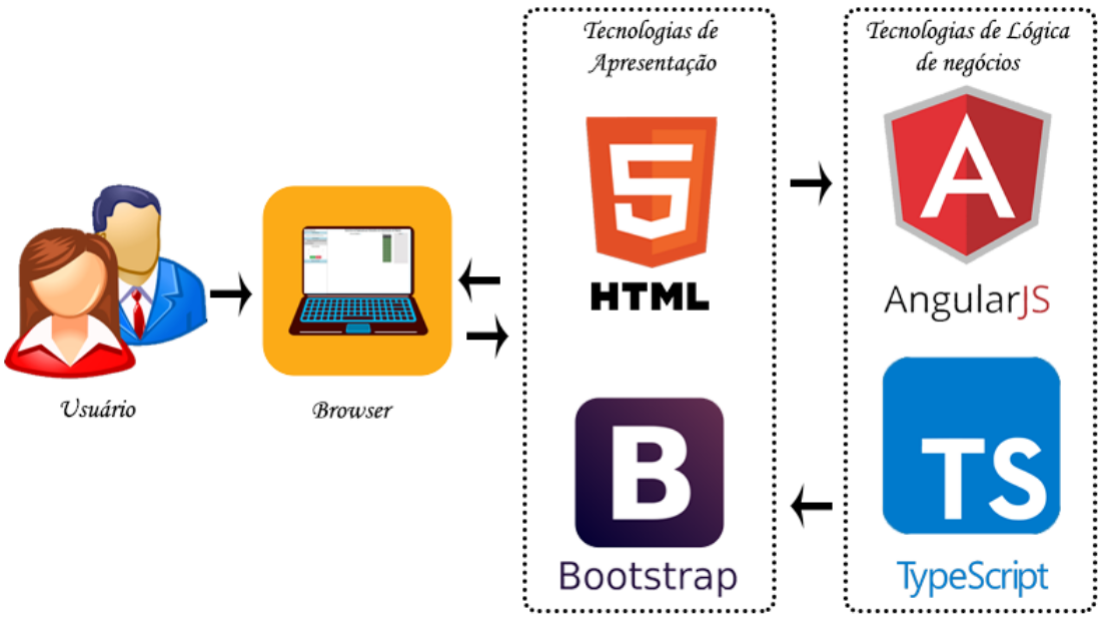
O módulo de exercícios de paginação por demanda com substituição de páginas oferece exercícios como: Preencher memória física, Preencher da memória física e determinar a página vítima. Espera-se que este módulo possa servir como objeto de auxílio aos usuários no processo de aprendizado desta área do conhecimento.

Serão abordados nas sessões seguintes: a arquitetura do software; a estrutura das tecnologias utilizadas em cada camada da aplicação; a estrutura interna da aplicação; a interface de interação com o usuário; Exemplos dos exercícios propostos; e, apresentação de trechos relevantes da aplicação.

4.1 Arquitetura Do Software

A Figura 13 apresenta a arquitetura da aplicação. A imagem ilustra as ferramentas/tecnologias utilizadas para a implementação e o relacionamento entre elas.

Figura 13. Estrutura da Aplicação



Conforme ilustrado na Figura 13, a arquitetura possui duas camadas, sendo a primeira a de Tecnologia de Apresentação, na qual o usuário pode visualizar os resultados obtidos a partir da camada de Lógica de Negócios. Assim, fornece ao

usuário a interface para a visualização das informações e funcionalidades contidas na aplicação.

Considerando que o OSLive é disponibilizado de forma *online*, sendo possível seu acesso via *browser*, tem-se uma maior disponibilidade e transparência para os usuários. Com isso, não se faz necessário a posse de recursos computacionais mais robustos, tornando assim o sistema de mais fácil acesso àqueles interessados.

Outro ponto a ser observado é a composição desta camada de Tecnologias de Apresentação utiliza o Bootstrap para oferecer uma uma identidade visual padronizada com as interfaces já conhecidas na Web. Além deste recurso, nesta camada, também foi utilizado o HTML na construção do layout, formulários e elementos.

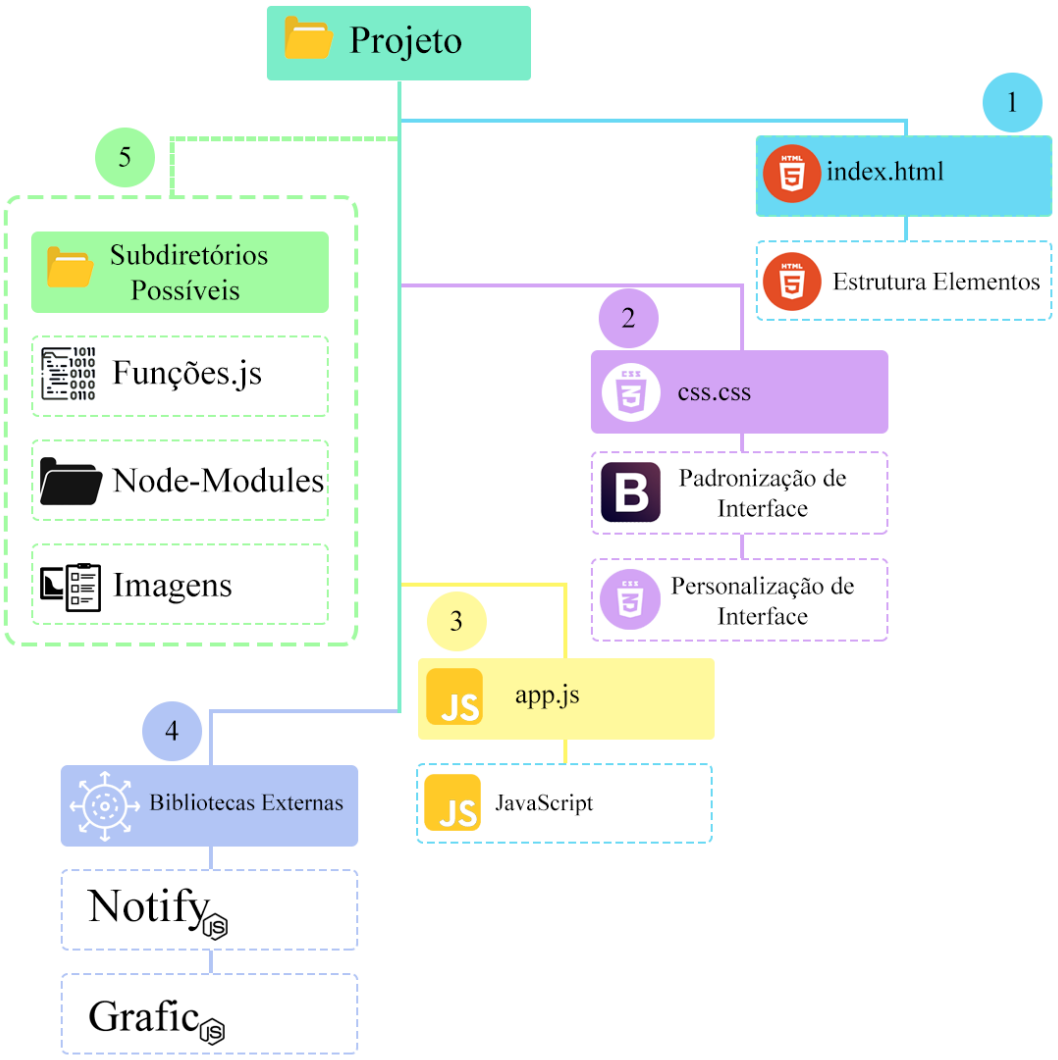
Já a camada de Lógica de Negócios é a camada que contém as classes, métodos, funções e demais recursos utilizados na implementação do módulo. Nessa camada, utilizou-se o *framework* Angular, juntamente com a linguagem de programação TypeScript, para a implementação dos métodos que controlam o fluxo de informações e tornam possível a emulação do funcionamento dos mecanismos de paginação por demanda e seus respectivos algoritmos de substituição de página, para a execução e correção dos exercícios.

4.2 Estrutura interna da Aplicação

Quanto à estrutura Interna da aplicação, foi efetuado um estudo dos projetos contidos no OSLive, visando identificar o padrão de organização já existente. Com base nos estudos realizados, verificou-se que os projetos contidos são independentes, autônomos e possuem uma formulação não voltada à intercomunicação entre eles, ou seja, aproveitamento de classes, funções e estruturas.

Foi identificado, também, que a maioria dos projetos foram feitos em javaScript, HTML, CSS e Bootstrap. E que, basicamente, seguiam uma organização interna das pastas e arquivos totalmente independente dos demais módulos já existentes, em outras palavras não havia um aproveitamento de códigos, e recursos já existentes em outros projetos. Para visualizar a estrutura contida nos projetos observe-se a Figura 14.

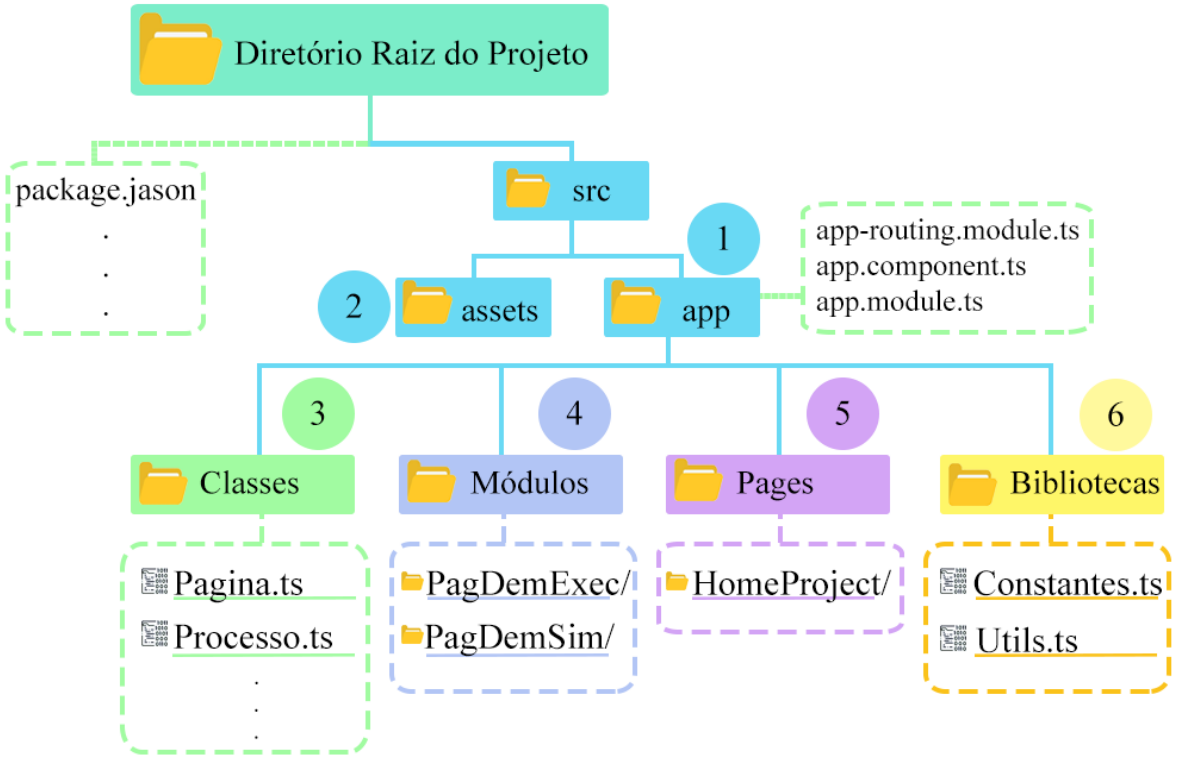
Figura 14. Visão da Estrutura Interna de um módulo OSLive



Como mostra a Figura 14, o item 1 apresenta o arquivo em HTML que possui a estrutura da aplicação, já o item 2 apresenta o arquivo de formatação, que possui as configurações específicas utilizadas e a presença de vinculação as formatações fornecidas pelo próprio BOOTSTRAP, e o item 3 a lógica de negócio, desenvolvida na linguagem JavaScript. Os item 4 apresenta a inclusão de bibliotecas externas ao código, como Notify.js (notificações ao usuário) e Grafic.js (plotação de diversos gráficos), e o item 5 contido em alguns projetos e que tem por intuito organização, ou seja, alguns projetos há uma pasta para imagens, ou outra para funções, ou outra para algum recurso que o desenvolvedor achou interessante organizar em uma pasta separada.

Com base na Figura 14 tem-se que os projetos possuíam apenas um diretório com todos os arquivos, com uma estrutura simples e intuitiva, que era pensada apenas para aquele projeto. Todavia, para o crescimento do ambiente OSLive, requer uma mudança de estrutura, para que os módulos trabalhem em um ambiente integrado e coeso. A fim de que seja possível o aproveitamento de código, e que todos os módulos atendam a um funcionamento semelhante de entendimento e utilização de estruturas e classes como: páginas, processos e memória. Assim há um crescimento no nível da aplicação, uma vez que haverá aproveitamento de código, estruturas que seguem um mesmo conceito de estrutura e redução do tamanho da aplicação, posto que não haverá redundância de recursos já desenvolvidos. A Figura 15 apresenta a sugestão de padrão de estrutura interna para o ambiente OSLive.

Figura 15. Sugestão de Macroestrutura para o OSLive



Na Figura 15 demonstra-se a estrutura interna sugerida para que o OSLive seja um projeto único e coeso, de forma que os demais módulos a serem desenvolvidos sejam interligados e usem uma estrutura padrão. Na Figura é demonstrada a organização de um projeto Angular, bem como a estrutura interna

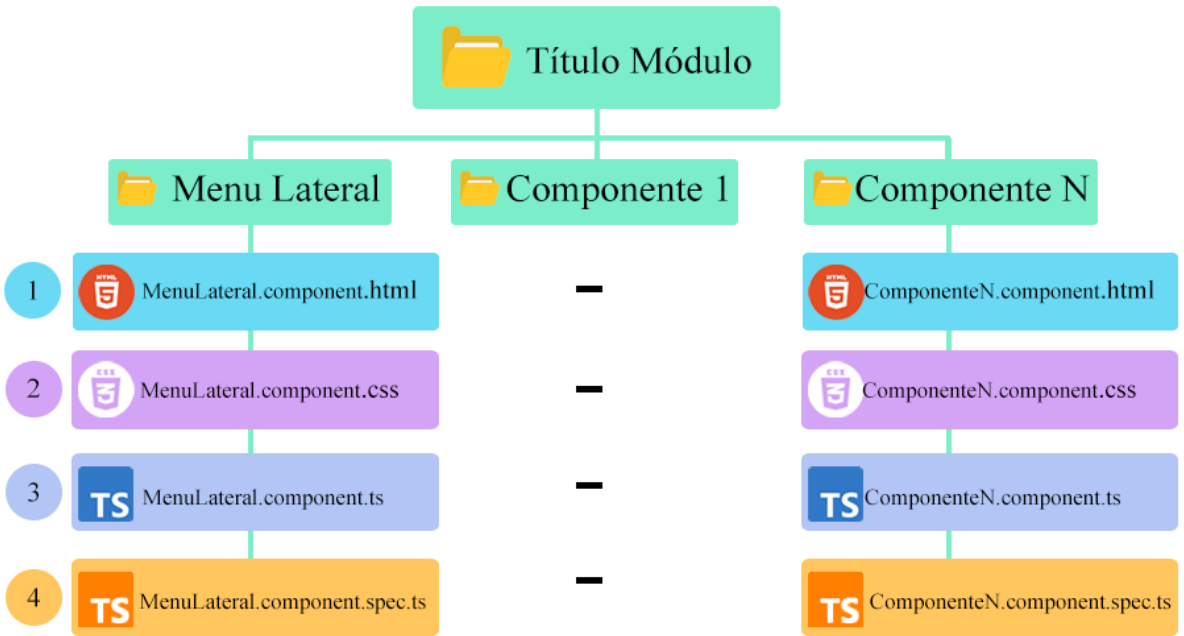
sugerida, as áreas em forma de retângulo tracejado indicam os arquivos contidos dentro das pastas, que estão numerados de 1 a 6.

O item (1), pasta app, é onde fica o projeto base, no qual por meio do arquivo **`app-routing.module.ts`** há a determinação da página inicial, bem como das rotas(links) para os módulos. Vale ressaltar que a pasta Pages (5) é indicada como local para inserir o projeto de *home page* do projeto, como também páginas não relacionadas aos módulos, como: quem somos nós, colaboradores, dentre outras.

O item (2), pasta assets, é o local onde pode-se armazenar arquivos como imagens, entre outros, que sejam necessários nos projetos. Nesta arquitetura, os projetos podem aproveitar classes (3), funções (6), parametrizar constantes (6) e até elementos já prontos. A pasta módulo (4) é o local em que os módulos deverão ser armazenados.

Como pode ser visto na Figura 15, a macroestrutura é preparada para servir os módulos, de modo que possa ocorrer aproveitamento de códigos já desenvolvidos em outros trabalhos e seja possível realizar um refinamento no que já existe, quando necessário. Visando demonstrar a estrutura proposta aos trabalhos a serem desenvolvidos, a Figura 16 apresenta o padrão de arquitetura utilizado na implementação do módulo para a resolução de exercícios sobre paginação por demanda e algoritmos de substituição de páginas desenvolvido neste trabalho.

Figura 16. Proposta de Estrutura para os Módulos do OSLive



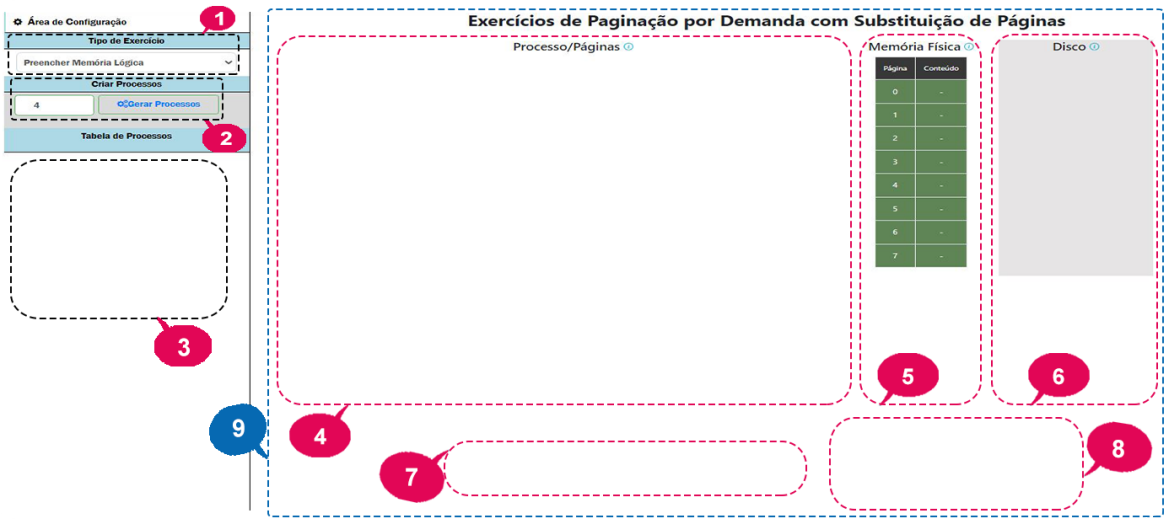
A Figura 16 apresenta o padrão que foi seguido neste módulo e que é sugerido a ser utilizado na arquitetura e implementação de futuros módulos no OSLive. Tendo em vista os elementos padrões do ambiente, há um módulo componente para o menu lateral, outro para a área do exercício e outro para ser a *home* daquele módulo, onde o desenvolvedor poderá fazer todas as mudanças necessárias para sua interação sem interferir em outros projetos e ainda assim ter acesso aos recursos apresentados na Figura 15.

A ideia de componentização de N elementos se dá pela concepção de que cada componente deve executar apenas uma finalidade. Já que dessa maneira apenas o que é necessário é carregado ao usuário no momento de utilização do sistema. Posto que no momento em que um único componente executa mais de uma tarefa distinta, acaba por acarretar no carregamento e alocação de variáveis, códigos, elementos e arquivos desnecessariamente a memória RAM do usuário.

4.3 Interface Da Aplicação

A apresentação visual da interface do módulo de exercícios de paginação por demanda com substituição de páginas segue as escolhas de *design* já utilizadas nos outros módulos do OSLive, principalmente, as do módulo de Simulação de paginação por demanda. Os elementos de interação e os objetos resultantes das interações seguem o mesmo padrão das demais interfaces do ambiente OSLive, buscando padronizar e proporcionar uma interação mais intuitiva e amigável com o usuário. A Figura 17 mostra a tela inicial do módulo, antes de qualquer interação com o usuário.

Figura 17. Tela Inicial do módulo



Na Figura 17 é apresentada a tela inicial do módulo na qual foram enumeradas os componentes da interface, abaixo segue a descrição de cada uma delas:

1. Menu de seleção do exercício a ser realizado: no momento os exercícios ofertados são: "Preencher Memória Física", "Preencher Memória Lógica" e "Determinar Página Vítima". No caso específico de se escolher o exercício "Determinar Página Vítima", surge mais uma aba, entre o item (1) e (2) da Figura acima, a qual permite a escolha do algoritmo de substituição de páginas que deve ser utilizado. Atualmente os algoritmos disponibilizados são: O algoritmo FCFS, o algoritmo Histórico de Referência e o algoritmo Segunda Chance;
2. Neste local, há uma caixa de texto, no qual vem pré informado o valor 4 que pode ser editado pelo usuário, sendo possível a escolha de quantos processos deseja que o exercício contenha, sendo um valor compreendido entre 1 a 4. Com base nesse valor contido neste campo os processos são criados em tamanho aleatório a partir do clique no botão gerar processo contido neste item;
3. Nessa área é exibida uma tabela que contém a lista de processos criados, apresentando sua cor, nome e quantidade de páginas;
4. Essa área é reservada para a exibição do(s) processo(s) na memória lógica -Processos/Páginas-, de acordo com a quantidade definida pelo usuário.

Exibindo assim a(s) página(a) do(s) processo(s) a tabela de páginas de cada processo, indicando se o mesmo está alocado na memória principal;

5. É a área de representação da memória física, neste caso, de tamanho correspondente a 8 bytes de espaço disponível. Nessa representação a primeira coluna corresponde ao endereço do espaço da memória principal, e a segunda coluna, caso preenchida, corresponde a página que está alocada naquele endereço, do contrário ela fica em branco que indica que o espaço está disponível;
6. É a área de representação do disco utilizado como memória de retaguarda. Que exibe o conceito de que as páginas são alocadas primeiro na memória de retaguarda e carregadas a memória principal apenas sob demanda;
7. Área reservada para os botões, “Corrigir Resposta”, "Visualisar Resposta” e “Limpar”, que respetivamente corrigem a resposta apresentada, exibem o gabarito do exercício e retornam a home, mas especificamente ao exercício de preencher memória lógica em seu estado inicial;
8. Área reservada para mensagens, que apontem êxito ou inexatidão da resposta;
9. Demarcação da área de apresentação do exercício.

A proposta dessa seção foi apresentar a interface do módulo de exercícios de paginação por demanda e algoritmos de substituição de páginas e os elementos que a compõem, nas próximas seções serão apresentadas demonstrações dos exercícios e descrição de seu funcionamento, assim, será possível observar o comportamento da interface em decorrência das situações.

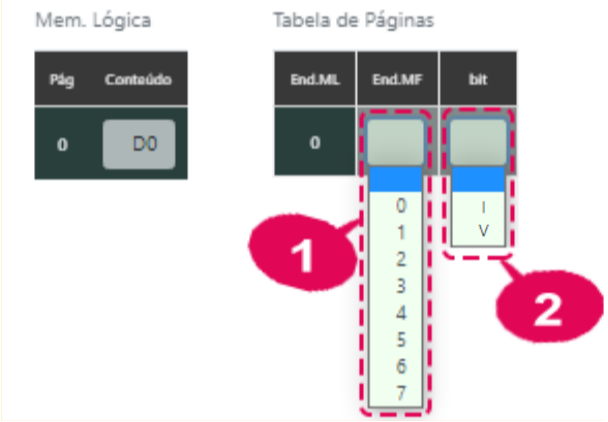
4.4 Demonstração Do Exercício Preencher Memória Lógica

Nesta seção é apresentada a demonstração do exercício de Preencher memória lógica. Nesse exercício o usuário deve preencher os campos (1) e (2) conforme mostra a Figura 19, com o intuito de representar os estados e endereços das páginas em um determinado momento de execução, com base nas informações das páginas alocadas ou não na memória física representada no item (3) da Figura 18.

Figura 18. Tela do Exercício Preencher Memória Lógica

Na situação apresentada na Figura 18 foram criados 4 processos (1) em que a quantidade de páginas foi gerada aleatoriamente, no exemplo tem-se: A, com 3 páginas; B, com 4 páginas; C, com 3 páginas; D, com uma página. Após a criação dos processos, suas páginas foram alocadas na memória física (3) de forma aleatória, podendo assim o usuário começar a preencher as informações ausentes na memória lógica.

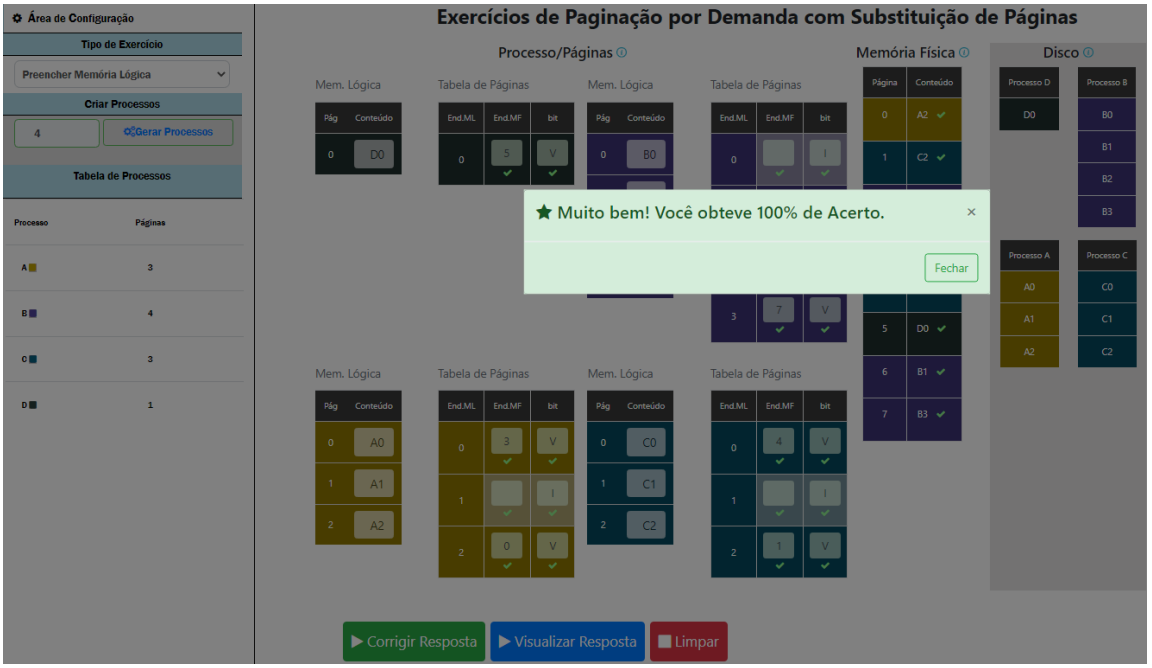
Figura 19. Preenchimento da resposta ao exercício



A partir desse momento, com o exercício gerado, o usuário deve indicar corretamente: o endereço que a página está ocupando na memória física, item (1), deixando em branco o campo referente à páginas não alocadas; e o status do bit de validade que indica a alocação na memória - preenchendo-o com I caso não alocado ou V caso alocado, item (2), sendo considerado erro deixar em branco esse campo.

Quando terminado o preenchimento da resposta, o usuário deve selecionar a opção “Corrigir Resposta”, item (5) da Figura 18, para que o exercício seja corrigido. Quando essa opção selecionada apresenta-se uma mensagem de parabenização quando o usuário preenche tudo corretamente, como apresentado na Figura 20.

Figura 20. Mensagem de êxito.



Conforme apresentado na Figura 20, além da mensagem de êxito, todos os campos em que o usuário preencheu com a resposta correta, são indicados com o símbolo “**V**” na cor verde. Mas caso o usuário não atinja 100% de êxito, os campos incorretos são marcados com o símbolo **“X”** na cor vermelha. Isso pode ser observado na Figura 21.

Figura 21. Mensagem de insucesso.

Quando o usuário preenche alguma informação de forma incorreta, é apresentada a mensagem na Figura 21 e as respectivas indicações dos campos com as respostas erradas, por meio do símbolo "**X**" na cor vermelha. Outro ponto a ser observado é que, mesmo quando se clique no botão de "Corrigir Resposta" ainda fica habilitado o usuário editar sua resposta, sendo possível que ele tente corrigir a si mesmo a fim de obter êxito.

Quando o usuário clicar no botão "Visualizar Resposta", contido no item (5) da Figura 18, ele tem a visualização do gabarito. Conforme apresentado na Figura 22.

Figura 22. Tela visualizar resposta.

Quando o usuário clicar na opção "Limpar", item (5) da Figura 18, o mesmo é encaminhado à tela inicial do módulo. Na qual não há processos gerados e o exercício Preencher Memória Lógica está previamente selecionado.

4.5 Demonstração Do Exercício Preencher Memória Física

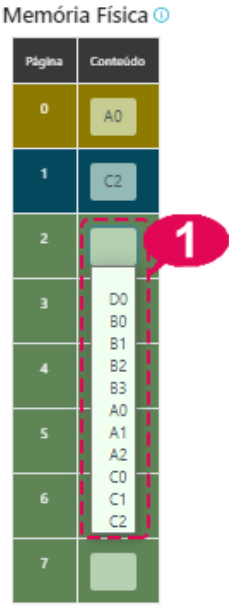
Nesta seção é apresentada a demonstração do exercício de preencher memória física item (3) da Figura 23. Nesse exercício o usuário deve preencher os endereço da memória física (3) com o nome das páginas alocadas em um determinado momento de execução, com base no registro das páginas dos processos contidos na memória lógica (2) - “Processo/Páginas” - representada na Figura 23.

Figura 23. Tela do Exercício Preencher Memória Física

Na Figura 23, tendo em vista a indicação da quantidade de processos, foram criados 4 processos (1) em que a quantidade de páginas foi gerada aleatoriamente. No exemplo apresentado na Figura 23 temos: A, com 3 páginas; B com 4 páginas; C com 3 páginas; e D com uma página. Após a criação dos processos suas páginas foram alocadas na memória física (3) de forma aleatória.

Concluída essa etapa, a tela não apresenta inicialmente a alocação da memória física, onde fica disponível ao usuário clicar no campo conteúdo correspondente a cada endereço de espaço da memória física (3), e selecionar qual página está alocada naquele endereço, item (1) da Figura 24. Há também a possibilidade do campo ficar em branco, caso não haja processos o suficiente para preencher aquele espaço de memória.

Figura 24. Preenchimento da Memória Física



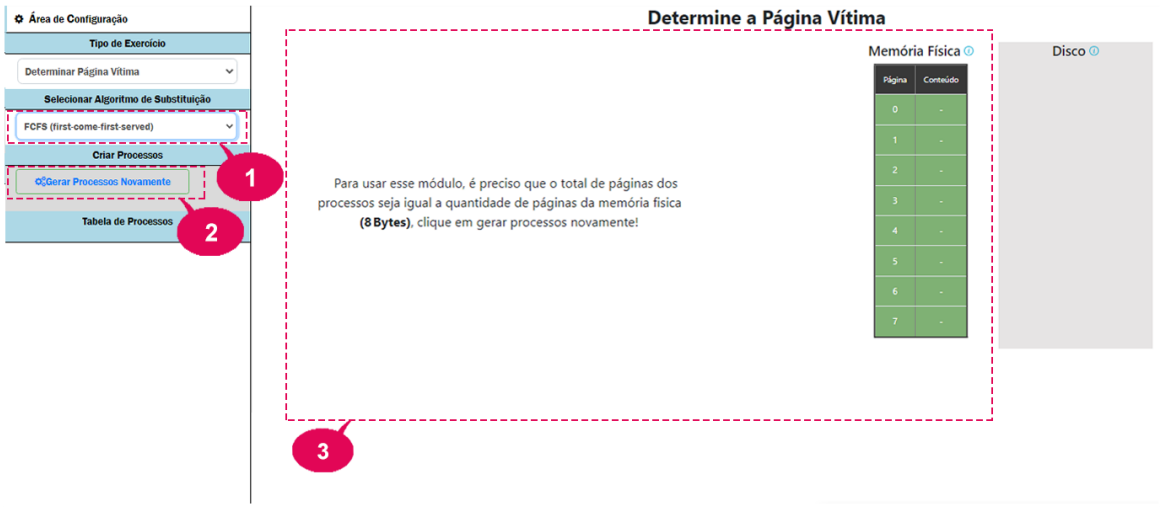
Quando o usuário concluir o preenchimento, poderá clicar no botão “Corrigir Resposta”, o qual irá marcar com “**V**” os espaços preenchidos corretamente, e com “**X**” os preenchidos de forma equívoca. Como no exercício anterior, não fica bloqueada a edição, podendo o usuário ir nos campos que errou e mudar a resposta indicada.

Caso o usuário preencha corretamente todos os campos, obterá uma mensagem de parabenização, a qual contém o índice de acerto. De igual maneira, caso a resposta não esteja perfeitamente correta, uma mensagem indicará que há divergência e indicará o índice de acerto obtido.

4.6 Demonstração Do Exercício Determinação De Página Vítima

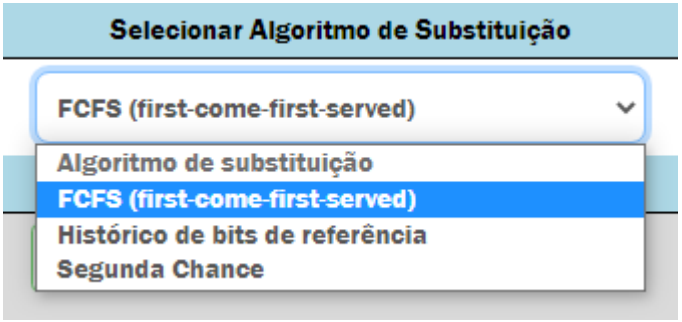
Nesta seção é apresentada a demonstração do exercício determinar página vítima, com base nas informações que são apresentadas no item (3) da Figura 25. Nesse exercício o usuário deve determinar a página vítima com base nos algoritmos de substituição de página disponíveis.

Figura 25 - Tela do Exercício Determinar Página Vítima



Na Figura 25, tem-se a visão inicial do módulo, quando ainda não estão alocados os processos ou ao se chegar a este exercício com menos páginas do que a quantidade de espaços físicos disponíveis. Desta maneira, não apresentando o exercício, já que não haverá página vítima se houver espaços livres para alocação de páginas. Dando prosseguimento, no item (1) o usuário pode escolher qual algoritmo irá ser utilizado no exercício, como pode ser visto abaixo na Figura 26.

Figura 26- Seleção do Algoritmo de Substituição de Páginas

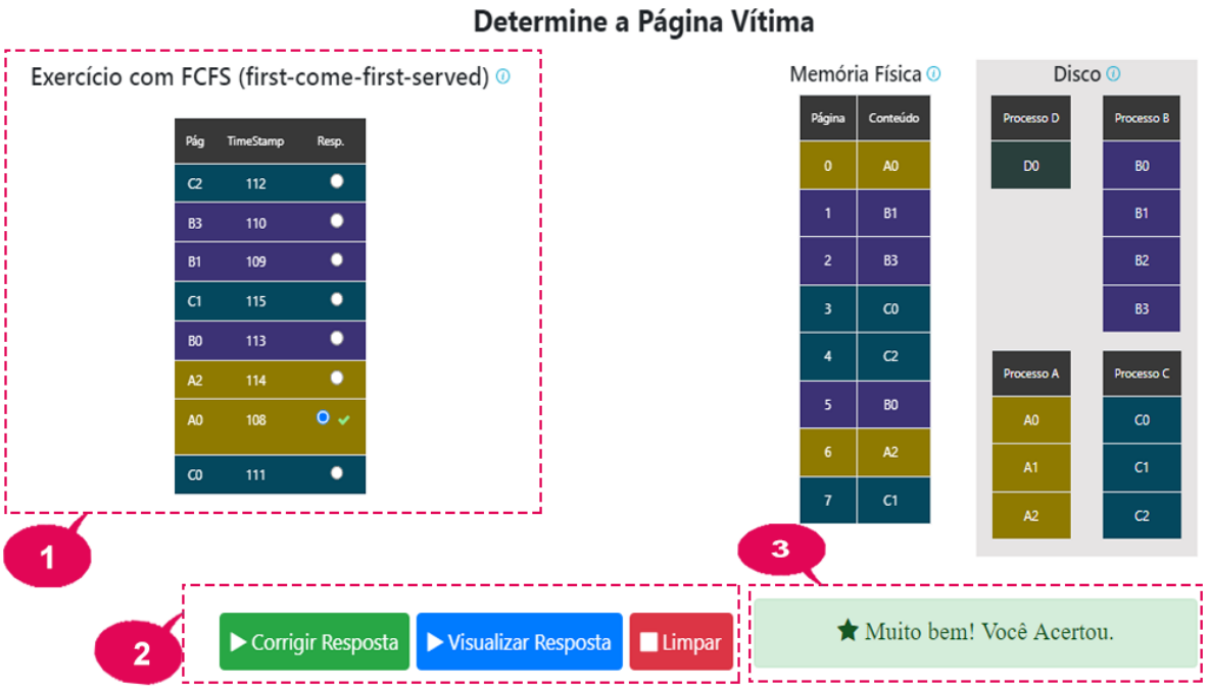


Com base na Figura 26, tem-se a visão dos algoritmos disponíveis, o que por sua vez selecionado ainda ficará aguardando a geração dos processos. Quando gerado os processos, uma das seguintes telas será exibida para que o exercício seja realizado, conforme pode ser observado nos itens 4.6.1 Demonstração Do Exercício Com Algoritmo FCFS, 4.6.2 Demonstração Do Exercício Com Algoritmo Histórico de Bits De Referência e 4.6.3 Demonstração Do Exercício Com Algoritmo Segunda Chance.

4.6.1 Demonstração Do Exercício Com Algoritmo FCFS

Nesta seção é apresentada a demonstração do exercício determinar página vítima, quando escolhido este algoritmo e tendo sido gerado os processos. Com base nas informações que são apresentadas no item (1) da Figura 27. O usuário terá que identificar qual página será escolhida como vítima.

Figura 27- Área do Exercício se Selecionado Algoritmo FCFS



Na Figura 27, é mostrado um exemplo de utilização do módulo, onde se obteve êxito na resposta. Veja que neste exercícios as informações necessárias para sua resolução encontram-se no item (1), onde usuário marca sua resposta. No item (2), ao clicar em “Corrigir Resposta”, caso esteja correta a marcação é indicado no item (3) uma mensagem indicando o acerto. Contudo, caso a opção selecionada esteja incorreta, o usuário recebe uma segunda chance, posto que surge na tela a Figura 28, que apresenta o funcionamento do algoritmo, e pede que o usuário tente novamente.

Figura 28 - *According Feedback* Algoritmo FCFS

Caso o usuário mantenha sua escolha, ou mude sua resposta para uma alternativa para uma incorreta, há a apresentação de uma mensagem indicando o erro, que é apresentada no item (3) da Figura 28. Do contrário, caso escolha a alternativa correta é parabenizado pelo êxito.

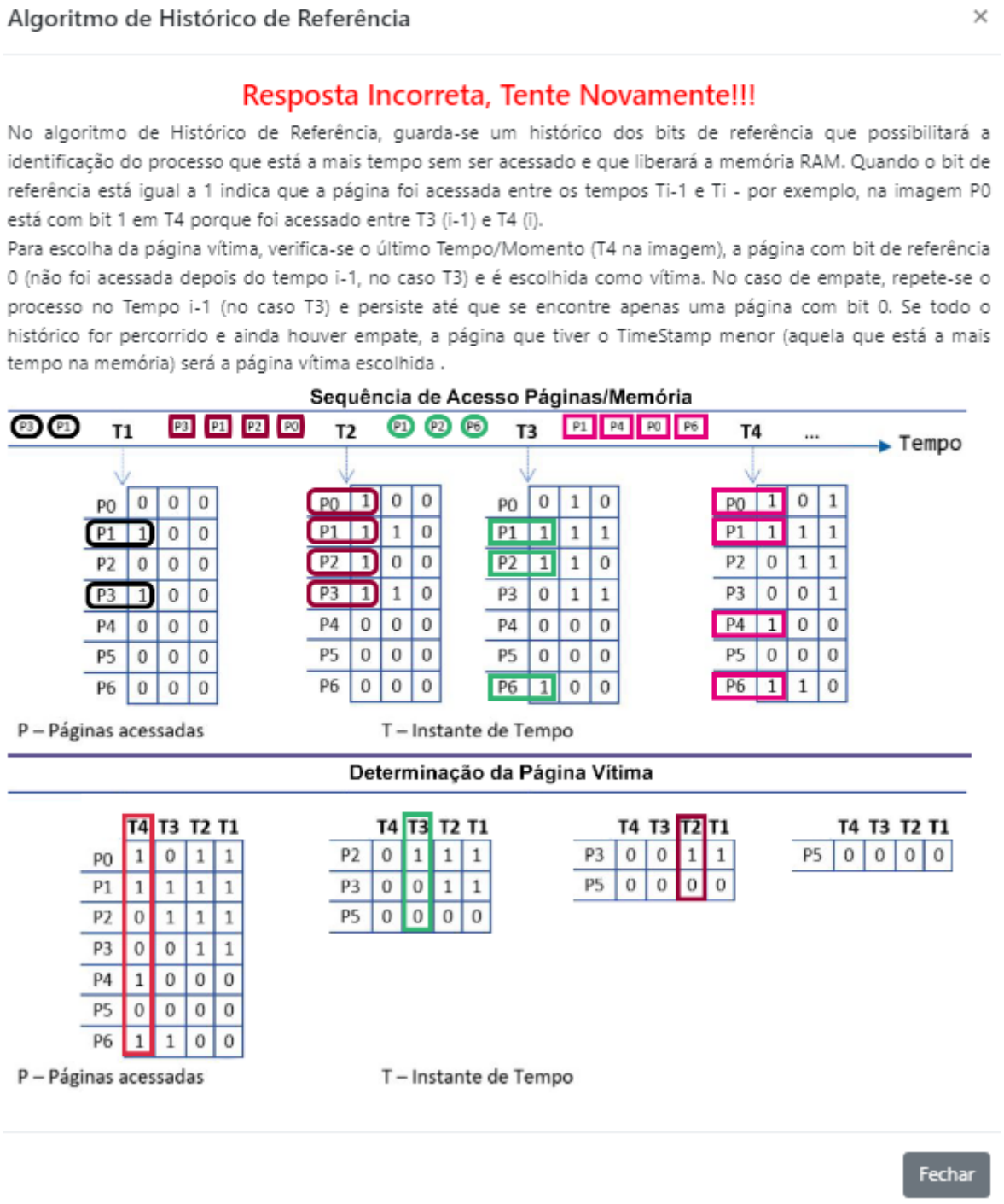
4.6.2 Demonstração Do Exercício Com Algoritmo Histórico de Bits De Referência

Nesta seção é apresentada a demonstração do exercício determinar página vítima, quando escolhido o algoritmo histórico de referência após terem sido gerados os processos. Com base nas informações que são apresentadas no item (1) da Figura 29. O usuário terá que identificar qual página será escolhida como vítima.

Figura 29- Área do Exercício se Selecionado Algoritmo Histórico de Referência

Conforme explicado no item 2.1.1.4 deste trabalho, observados todos os critérios de desempate, o campo *TimeStamp* é o campo utilizado para efetuar o desempate final quando necessário. O mesmo descrito no capítulo anterior se aplica a este quando o usuário escolhe uma opção incorreta, ele recebe uma segunda chance sendo apresentado o *feedback* abaixo, Figura 29, que além de apresentar o conceito e funcionamento deste algoritmo, indica ao usuário que a resposta indicada está inadequada.

Figura 30 - *According Feedback* Algoritmo Histórico de Referência

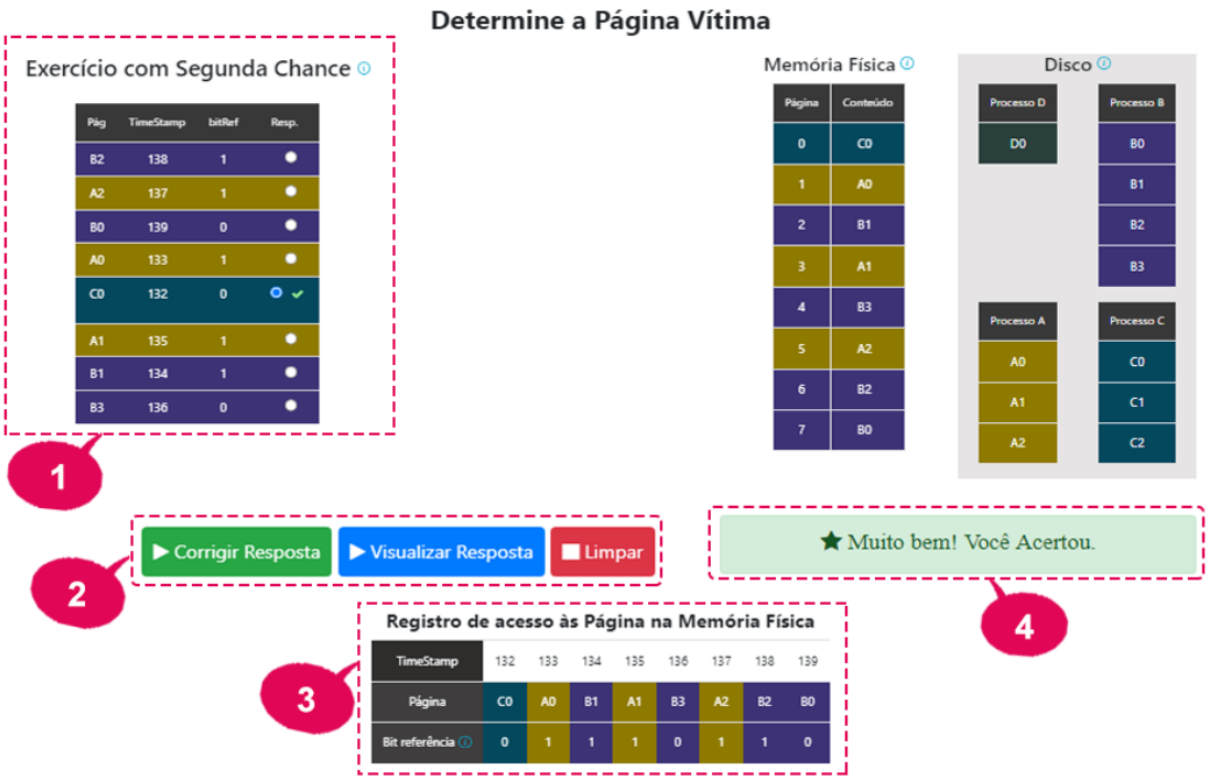


Caso o usuário mantenha sua escolha, ou mude sua resposta para uma alternativa para uma incorreta, há a apresentação de uma mensagem indicando o erro, que é apresentada no item (3) da Figura 29. Do contrário, caso escolha a alternativa correta é parabenizado pelo êxito.

4.6.3 Demonstração Do Exercício Com Algoritmo Segunda Chance

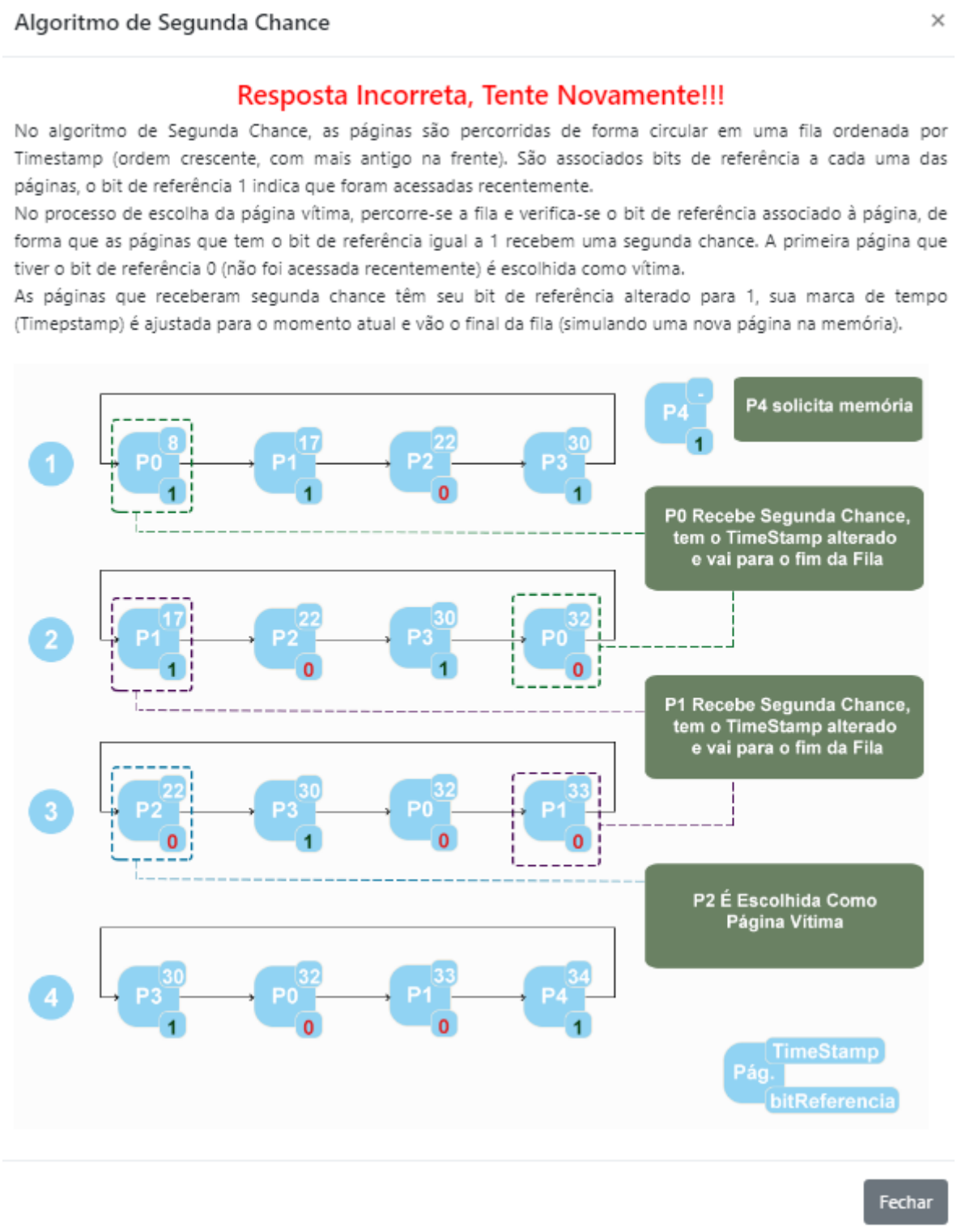
Nesta seção é apresentada a demonstração do exercício determinar página vítima, quando escolhido o algoritmo segunda chance, após terem sido gerados os processos. Com base nas informações que são apresentadas no item (1) e item (3) da Figura 31. O usuário terá que identificar qual página será escolhida como vítima.

Figura 31 - Área do Exercício se Selecionado Algoritmo Segunda Chance



Conforme explicado no item 2.1.1.6 o usuário deve responder este exercício, o item (3) serve como um auxílio para o usuário identificar mais facilmente a ordem dos processos e identificar qual página possui o menor *TimeStamp* com bit de referência igual a 0. O mesmo descrito no capítulo anterior se aplica a este quando o usuário escolhe uma opção incorreta, ele recebe uma segunda chance sendo apresentado o *feedback* abaixo, Figura 32, que além de apresentar o conceito e funcionamento deste algoritmo, indica ao usuário que a resposta indicada está inadequada.

Figura 32 - *According Feedback* Algoritmo Segunda Chance



Caso o usuário mantenha sua escolha, ou mude sua resposta para uma alternativa para uma incorreta, há a apresentação de uma mensagem indicando o erro, que é apresentada no item (4) da Figura 31. Salvo escolha a alternativa correta, sendo então parabenizado pelo êxito.

4.7 Trechos De Código Da Aplicação

Nesta seção são apresentados os trechos de código mais relevantes, usados para implementar o funcionamento dos algoritmos disponibilizados. No decorrer desta seção serão apresentados os códigos que implementam os conceitos de: Página; Processo; Memória Física; Algoritmos de Substituição de Páginas: FCFS, Histórico bit de referência e Segunda Chance. Além disso, são apresentados trechos dos código dos Exercícios de Preencher Memória Física e Lógica e Determinar Página Vítima.

Para isso é importante entender como as classes basilares no que tange sua composição e também como ocorrem suas interações. Neste caso, as classes básicas são: **`Pagina`**, **`Processo`** e **`MemoriaFisica`**, que determinam os componentes lógicos e físicos necessários para implementar o mecanismo de paginação por demanda.

A classe **`Pagina`** é composta por: Nome; Cor; Índice na Memória Lógica do Processo; Índice na Memória Física e *TimeStamp* do momento de Alocação. É importante entender que o nome da página é igual ao nome do processo, dessa maneira quando se deseja visualizar o nome da página (nome do processo mais a posição que a mesma ocupa na memória lógica) deve-se utilizar a função “toString()”.

A classe **`Processo`** é composta por: nome do processo, um *Array* de páginas, a cor do processo e o bit usado como flag para indicar alocação do processo. Nessa classe, quando se cria um processo, basta informar o nome do processo, sua cor e a quantidade de páginas, sendo que o *Array* de páginas será preenchido automaticamente com todas as informações básicas iniciais.

Já a classe **`MemoriaFisica`** é composta por: Endereço da Memória Física; Cor, seja a cor padrão não alocada ou a cor da página que a ocupa; Nome, que pode ser vazio - representando que não página alocada naquele endereço - ou o nome da página que ocupa aquele endereço; e a horaCarga que corresponde ao *TimeStamp* da página que a ocupa.

Nos algoritmos de substituição de página, a instância da memória física é enviada à instância da classe do algoritmo de substituição de páginas em questão que ao efetuar as alterações - alterações como vincular endereços de memória física

a páginas e vice-versa -, trabalha diretamente no endereço da variável do código principal. Com base nesses conceitos, o classe **FCFS** basicamente é uma lista de páginas que segue os funcionamento conceitual do algoritmo de substituição FCFS, como descrito no subcapítulo 2.1.1.3. A Figura 33 apresenta o trecho de adição de páginas à memória física.

Figura 33 - Classe FCFS

```
6   export class FCFS {
7         public lista: Array<Pagina> = [];
8
9         constructor (){
10        }
11  >     primeiraPosicaoDisponivel(memoriaFisica: Array<MemoriaFisica>): number{ ···
16        }
17  >     listaVazia():boolean{ ···
19        }
20
21        addPaginaEmMemoriaFisica(memoriaFisica: Array<MemoriaFisica>, paginaX: Pagina, timestamp:number):number{
22              var posicaoParaInsercao:number = this.primeiraPosicaoDisponivel(memoriaFisica);
23              if(posicaoParaInsercao==-1){
24                    posicaoParaInsercao = this.lista[0].indiceMemoriaFisica;
25                    this.lista[0].timeStamp = 0;
26                    this.lista[0].indiceMemoriaFisica = -1;
27                    this.lista.shift();
28              }
29
30              memoriaFisica[posicaoParaInsercao].nome = paginaX.toString();
31              memoriaFisica[posicaoParaInsercao].cor = paginaX.cor;
32              memoriaFisica[posicaoParaInsercao].horaCarga = timestamp;
33
34              paginaX.indiceMemoriaFisica = posicaoParaInsercao;
35              paginaX.timeStamp = timestamp;
36
37              this.lista.push(paginaX);
38
39              return posicaoParaInsercao;
40        }
41
42  >     removerProcesso(memoriaFisica: Array<MemoriaFisica>, paginaX: Pagina) : number{ ···
61        }
62  }
```

Na Figura 33,das linhas 21 a 40, como pode ser observado na função “addPaginaEmMemoriaFisica()”, primeiro é executadaa função para verificar se a memória física está cheia, função “primeiraPosicaoDisponivel()”, que retorna “-1”, caso a memória esteja cheia, ou a primeira posição disponível para alocação.

Com isso, caso seja indicado que a memória física está cheia, as regras do algoritmo FCFS determinam que a página com menor *TimeStamp* seja a página vítima. Então, como a lista do algoritmo é ordenada por *TimeStamp*, a página que está na posição “0” zero é a escolhida como página vítima.

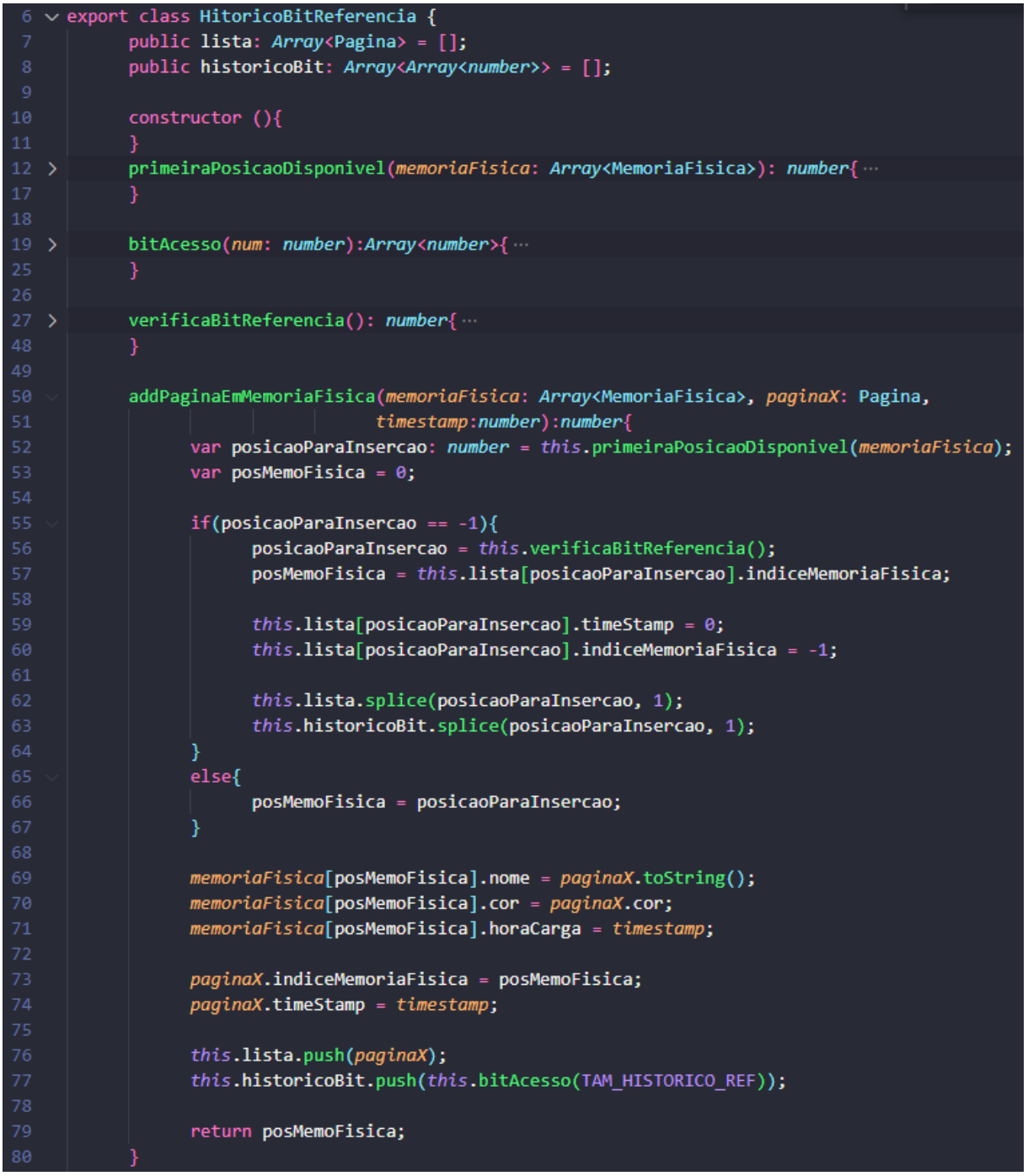
Por estar trabalhando com *Array* de objetos da classe Pagina, antes de efetuar a remoção do primeiro item da lista, é necessário inserir os valores que indicam que a página não está alocada. Uma vez que nestas condições as

alterações efetuadas nas linhas 25, 26 da Figura 33 persistem nessas 3 três variáveis - na instância da MemoriaFisica, na instância do Processo detentor da Pagina e na lista do algoritmo FCFS-. Atentando-se que as alterações da linha 27 da Figura 33, remove o apontamento da lista do algoritmo FCFS à página vítima e isto impacta apenas nesta instância.

Caso a função retorne um valor diferente de “-1”, indica a posição que está disponível para alocação. Em ambos os casos, após a posição para inserção ser determinada, há a adição da página lógica na página física, e sua respectiva atribuição de valores - valores esses que vinculam a página lógica a página vítima e vice versa -, por fim sua adição a fila de páginas do algoritmo FCFS.

Na classe que determina o funcionamento do algoritmo de substituição de página histórico de bit de referência, há duas listas, uma que armazena a lista de páginas ordenadas por *TimeStamp* e outra que armazena o histórico de bits de referência, dessa maneira a função de adicionar páginas a memória física segue a estrutura apresentada na Figura 34.

Figura 34 - Classe Histórico Bit De Referência



```
6  export class HitoricoBitReferencia {
7        public lista: Array<Pagina> = [];
8        public historicoBit: Array<Array<number>> = [];
9
10       constructor (){
11       }
12       primeiraPosicaoDisponivel(memoriaFisica: Array<MemoriaFisica>): number{ ···
17       }
18
19       bitAcesso(num: number):Array<number>{ ···
25       }
26
27       verificaBitReferencia(): number{ ···
48       }
49
50       addPaginaEmMemoriaFisica(memoriaFisica: Array<MemoriaFisica>, paginaX: Pagina,
51                                timestamp:number):number{
52            var posicaoParaInsercao: number = this.primeiraPosicaoDisponivel(memoriaFisica);
53            var posMemoFisica = 0;
54
55            if(posicaoParaInsercao == -1){
56                 posicaoParaInsercao = this.verificaBitReferencia();
57                 posMemoFisica = this.lista[posicaoParaInsercao].indiceMemoriaFisica;
58
59                 this.lista[posicaoParaInsercao].timeStamp = 0;
60                 this.lista[posicaoParaInsercao].indiceMemoriaFisica = -1;
61
62                 this.lista.splice(posicaoParaInsercao, 1);
63                 this.historicoBit.splice(posicaoParaInsercao, 1);
64            }
65            else{
66                 posMemoFisica = posicaoParaInsercao;
67            }
68
69            memoriaFisica[posMemoFisica].nome = paginaX.toString();
70            memoriaFisica[posMemoFisica].cor = paginaX.cor;
71            memoriaFisica[posMemoFisica].horaCarga = timestamp;
72
73            paginaX.indiceMemoriaFisica = posMemoFisica;
74            paginaX.timeStamp = timestamp;
75
76            this.lista.push(paginaX);
77            this.historicoBit.push(this.bitAcesso(TAM_HISTORICO_REF));
78
79            return posMemoFisica;
80       }
```

Como pode ser observado na Figura 34, das linhas 63 a 94, a função “addPaginaEmMemoriaFisica()” chama a função “primeiraPosicaoDisponivel()” para verificar se a memória física está cheia, função que retorna “-1”, caso a memória física esteja cheia, ou a primeira posição disponível para alocação.

Quando é apontado que a memória física está cheia, a função “addPaginaEmMemoriaFisica()” chama a função “verificaBitReferencia()” para a

verificação do histórico de bits de referência, e como foi indicado o subcapítulo 2.1.1.4, sendo acrescido como último critério de desempate que a página com menor *TimeStamp* será a página vítima. Para exemplificar esse funcionamento, a Figura 35 apresenta o trecho de código que contém a função responsável por utilizar estes conceitos e retornar qual é a posição da página vítima.

Figura 35 - Função Verifica Bit de Referência

```
27      verificaBitReferencia(): number{
28            var posicaoComzero: Array<number> = Utils.listaNum(TAM);
29
30            for(var j = 0;j < TAM_HISTORICO_REF;j++){
31                  var apagar: Array<number> = [];
32
33                  for(var i = 0; i< posicaoComzero.length; i++){
34                        if(this.historicoBit[posicaoComzero[i]][j] == 1)
35                              apagar.push(posicaoComzero[i]);
36                  }
37                  if(posicaoComzero.length==apagar.length) apagar = [];
38                  else {
39                        for( let i of apagar){
40                              var x = posicaoComzero.indexOf(i);
41                              posicaoComzero.splice(x, 1);
42                        }
43                        if((posicaoComzero.length-apagar.length) == 1)
44                              j = TAM_HISTORICO_REF;
45                  }
46            }
47            return posicaoComzero[0];
48      }
```

Na Figura 35, a função “verificaBitReferencia()” apresenta a verificação do histórico dos bits de referência e determina qual página será a página vítima. Este código efetua a verificação de todas as páginas que contém o número 1 que indica que a página foi acessada, e remove da lista de páginas a serem excluídas, atentando-se claro, a possibilidade de todas terem sido acessadas naquele instante, como visto no condicional da linha 37 da Figura 35, caso isso ocorra, não remove nenhuma página, e se regride no momento analisado. Caso persista empate por todas as páginas terem sido acessadas, ou por não terem sido, o *TimeStamp* será o critério utilizado para o desempate.

No algoritmo de segunda chance, semelhante ao algoritmo anterior há duas filas, e baseado nos conceitos apresentados no subcapítulo 2.1.1.5, existe a lista de páginas, de mesmo tamanho da memória física, e a segunda lista que contém o bit

de referência, ou seja, cada página possui apenas 1 bit de referência. Desse modo, como nos exemplos anteriores, se verifica a ocorrência de espaço disponível na memória física, e se apontado que a memória está cheia, então utiliza-se o método que aplica a fila o algoritmo segunda chance.

Quando a posição é definida ocorre o procedimento de resetar os valores da página vítima (inserir os valores default que indicam página não alocada) e inserir na fila a nova página e suas respectivas atualizações de valores por estar alocada e possuir *TimeStamp*. Vale destacar que conforme expresso na Figura 36, na linha 73, há uma aleatoriedade - entre 0 ou 1 - nos valores do bit de referência no momento de criação do exercício.

Figura 36 - Classe Segunda Chance

```
49      addPaginaEmMemoriaFisica(memoriaFisica: Array<MemoriaFisica>, paginaX: Pagina, _timestamp:number):number{
50            var posicaoParaInsercao:number = this.primeiraPosicaoDisponivel(memoriaFisica);
51            var posMemoFisica = 0;
52
53            if(posicaoParaInsercao == -1){
54                  posicaoParaInsercao = this.segundaChance(_timestamp);
55                  posMemoFisica = this.lista[posicaoParaInsercao].indiceMemoriaFisica;
56
57                  this.lista[posicaoParaInsercao].timeStamp = 0;
58                  this.lista[posicaoParaInsercao].indiceMemoriaFisica = -1;
59
60                  this.lista.splice(posicaoParaInsercao, 1);
61                  this.bitReferencia.splice(posicaoParaInsercao, 1);
62            }
63            else{ posMemoFisica = posicaoParaInsercao; }
64
65            memoriaFisica[posMemoFisica].nome = paginaX.toString();
66            memoriaFisica[posMemoFisica].cor = paginaX.cor;
67            memoriaFisica[posMemoFisica].horaCarga = _timestamp;
68
69            paginaX.indiceMemoriaFisica = posMemoFisica;
70            paginaX.timeStamp = _timestamp;
71
72            this.lista.push(paginaX);
73            this.bitReferencia.push((Math.floor(Math.random() * 15))%2);
74
75            return posMemoFisica;
76      }
```

Conforme apresentado na Figura 36, na linha 53 e 54, quando a memória física estiver cheia, é utilizada a função “segundachance()” a qual retorna a posição da página vítima, baseada nos conceitos do algoritmo segunda chance. A Figura 37 apresenta a referida função.

Figura 37 - Função Segunda Chance

```
18      segundaChance(_time: number,): number{
19          for(var i = 0; i< this.bitReferencia.length; i++){
20              if(this.bitReferencia[i] == 0 ){
21                  return i;
22              }
23              else {
24                  this.bitReferencia[i] = 0;
25                  this.lista[i].timeStamp = _time;
26                  _time += 1;
27
28                  var temp = this.lista[i];
29                  this.lista.splice(i,1);
30                  this.lista.push(temp);
31
32                  var temp2 = this.bitReferencia[i];
33                  this.bitReferencia.splice(i,1);
34                  this.bitReferencia.push(temp2);
35                  i--;
36              }
37          }
38          return 0;
39      }
```

Na Figura 37, há a demonstração do trecho de código que efetua o conceito do algoritmo de segunda chance, no qual é preciso que se percorra uma única vez a lista de bit de referência, em busca daquele que o bit seja igual a 0. Enquanto essa condição não for atendida, os bits verificados no primeiro “if” linha 20 da Figura 37, são convertidos em 0.

Percorrendo-se uma vez a lista inteira, e efetuando as movimentações demonstradas nas linhas 28 a 34 da Figura 37, e não sendo encontrado nenhuma página com bit igual a 0, aquele elemento que estiver na posição 0 da lista de páginas será a página vítima. Já que a lista é ordenada por *TimeStamp*, e o primeiro elemento - aquele que está na posição 0 da lista - é o que possui o menor *TimeStamp*, portanto o que está a mais tempo na CPU.

Por fim, o código que permite o bom funcionamento do exercício de preenchimento ou das páginas lógicas e bit de estado, ou então do preenchimento da memória física. É um algoritmo simples, posto que após criados os processos e alocados na memória as variáveis com a resposta já são geradas, ou seja, uma

Array de Processo e uma instância da classe MemoriaFisica que a título de exemplo serão nomeadas respectivamente “listaDeProcessos” e “memoriaF”.

Assim, após geradas e preenchidas, dependendo de qual exercício escolheu, o usuário irá preencher uma terceira variável da mesma classe, podendo então ser, “respostaMemoriaLogica” ou então “respostaMemoriaFisica”. Assim, quando se efetua a correção, ocorre uma comparação entre a variável que contém as respostas do usuário e a variável que foi gerada inicialmente, ou seja, o gabarito do exercício selecionado, “listaDeProcessos” ou “memoriaF”.

5 CONSIDERAÇÕES FINAIS

Este trabalho teve como objetivo desenvolver um módulo de paginação por demanda para o ambiente OSLive, ambiente que é utilizado como ferramenta de auxílio no aprendizado da disciplina de Sistemas Operacionais. Neste módulo são ofertados os exercícios: Preencher Memória Lógica, Preencher Memória Física e Determinar Página Vítima - tendo os algoritmos FCFS, Histórico bit de Referência e Segunda Chance. Esses exercícios servem de recurso para os alunos testarem seu entendimento sobre o funcionamento dos mecanismos de paginação por demanda explicados em sala de aula.

Além de testarem seus conhecimentos, os alunos podem aprender durante a utilização do módulo, pois quando eles erram é apresentado um *feedback* contendo uma explicação sobre o mecanismo do exercício proposto. Também é ofertada a possibilidade de segunda tentativa, a qual permite responder o exercício proposto, a partir do ponto em que estava, munido de uma explicação sobre o mecanismo em questão.

Além disso, a partir deste projeto foi determinado um modelo de estrutura para acoplar os demais projetos já existentes - macroestrutura -, como também uma sugestão de um modelo de microestrutura para os demais projetos que serão desenvolvidos. Essa microestrutura construída serve de exemplo para os próximos módulos, auxiliando a manter o padrão de interface como também exemplificando recursos de transmissão de valores e interação entre componentes.

Ademais, outros recursos foram incluídos, como uma pasta denominada **`Bibliotecas`** que contém arquivos como: **`Constantes.ts`** que contém as constantes que servem de parametrização dos valores dos recursos utilizados na aplicação, tais como: tamanho da memória física, tamanho do histórico de referência e valores representativos para vazio, e booleanos. Outro exemplo é o arquivo **`Utils.ts`** o qual contém funções que podem ser reaproveitadas por todos os módulos, uma das funções contidas neste arquivo e que teve uma atenção foi a forma de geração de cores, assegurando tons pastéis, pré definidos aleatoriamente sem repetição e que se relacionam bem entre si quando apresentadas na tela.

O desenvolvimento deste trabalho teve a supervisão e orientação da professora da disciplina de sistemas operacionais dos cursos de computação do CEULP/ULBRA, sob esta orientação o projeto pôde ser consolidado de forma a

atender o público interessado, e assim atingir os objetivos esperados. Para trabalhos futuros, sugere-se: Definir e implementar mecanismos para que seja passível compatibilização com modelos SCORM; mecanismos de armazenamento e análise de respostas de nível de acerto de um dado usuário ou uma turma.

Por fim, espera-se que o presente trabalho contribua para uma melhoria no módulo de paginação por demanda, e que sirva de base para um novo padrão de desenvolvimento, em que seja possível o aproveitamento de componentes para uma evolução como plataforma. Além disso, que este módulo possa auxiliar no ensino e aprendizagem dos conceitos estudados na disciplina de Sistemas Operacionais.

6 REFERÊNCIAS

Angular.io, **WHAT IS ANGULAR**, 2022. Disponível em: https://angular.io/guide/what-is-angular. Acesso em: 15, Out. 2022.

BENACCHIO, J. H. Q, LABORATÓRIO DE SISTEMAS OPERACIONAIS. Disponível em: <http://wiki.foz.ifpr.edu.br/wiki/images/9/98/SOsim.pdf>.Acesso em: 23 Out. 2021.

BOOTSTRAP, **About bootstrap**, 2021. Disponível em: https://getbootstrap.com/docs/4.1/about/overview. Acessado em: 16 Nov. 2021.

BRASIL. Ministério da Educação. **Parecer CNE/CES Nº 136/2012**, aprovado em 9 de março de 2012. Diretrizes Curriculares Nacionais para os cursos de graduação em Computação. Disponível em: <http://portal.mec.gov.br/index.php?option=com_docman&view=download&alias=11205-pces136-11-pdf&category_slug=julho-2012-pdf&Itemid=30192 >. Acesso em: 19 Set. 2021.

DEITEL, H. M.; DEITEL, P. J.; CHOFFNES, D. R. Choffnes. **Sistemas Operacionais.** 3° Edição. ed. São Paulo - Brasil: Pearson, 1999. 760 p. Disponível em: <https://goo.gl/7Me4my>. Acesso em: 12 Out. 2021.

DJANGO Rest Framework. Disponível em: https://www.django-rest-framework.org/. Acesso em: 16 Nov. 2021.

ELMASRI, Ramez. NAVATHE Shamkant.B. **Sistemas de banco de dados**. 2005. Disponível em: <http://tonysoftwares.com.br/attachments/article/5297/Sistema_de_banco_de_dados_Navathe.pdf> Acesso em: 16 Nov. 2021.

MAIA, L. P.; **SOsim: Simulador para o ensino de sistemas operacionais**. Tese (Mestrado em Computação) – Universidade Federal do Rio de Janeiro – UFRJ. 2001.

MAZIERO, C. A. **Sistemas Operacionais : Conceitos e Mecanismos IX - Virtualização**. [S. l.: s. n.], 2013.

OLIVEIRA, R. A., Carlos, A., & Bahia, S. (2015). **SWSO - Simulador Web de Sistemas Operacionais**. Disponível em: https://labrasoft.ifba.edu.br/publicacoes/tcc/swso-simulador-web-de-sistemas-operacionais.html. Acesso em 12 de Out. 2021.

OLIVEIRA, R. S. de; CARISSIMI, A. da S.; TOSCANI, S. S. **Sistemas operacionais**. v. VIII, p. 1–33, 2001. Disponível em: http://www.romulosilvadeoliveira.eng.br/artigos/Romulo-Carissimi-Simao-Rita2001.pdf Acesso em: 17 Set. 2021.

OLIVEIRA, R. A.; CARLOS, A.; BAHIA, S. **SWSO - Simulador Web de Sistemas Operacionais**. [s. d.]. 2015. Disponível em : https://ads.ifba.edu.br/dl1290 Acesso em 24 Nov. 2021

PRESSMAN, Roger; MAXIM, Bruce. **Engenharia de Software** - 8ª Edição. 2016. Disponível em: . Acesso em: 19 set . 2021.

Santos, A. M.; **Implementação de software para simulação do mecanismo de paginação**. Tese (Graduação em Ciência da Computação) – Centro Universitário Luterano de Palmas ULBRA – CEULP ULBRA . 2017.

SILBERSCHATZ, Abraham; GALVIN, Peter Baer; GAGNE, Greg. **Fundamentos de Sistemas Operacionais**. 8. ed. Rio de Janeiro: Ltc, 2013. 515 p.

TANENBAUM, Andrew S.; BOS, Helbert. **Sistemas Operacionais Modernos.** 3. ed. São Paulo: Pearson, 2016. Tradução de: Jorge Ritter. Disponível em: https://www.academia.edu/44215157/4_a_EDI%C3%87%C3%83O_SISTEMAS_OPERACIONAIS_MODERNOS. Acesso em: 17 Set. 2021.