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
ASSOCIAÇÃO EDUCACIONAL LUTERANA DO BRASIL

Fernando Silva Noleto

DESENVOLVIMENTO DE UM AMBIENTE WEB PARA CRIAÇÃO E CORREÇÃO AUTOMÁTICA DE EXERCÍCIOS DE ALGORITMOS DE ESCALONAMENTO DE PROCESSOS

Palmas – TO

2017

Fernando Silva Noleto

DESENVOLVIMENTO DE UM AMBIENTE WEB PARA CRIAÇÃO E CORREÇÃO AUTOMÁTICA DE EXERCÍCIOS DE ALGORITMOS DE ESCALONAMENTO DE PROCESSOS

Trabalho de Conclusão de Curso (TCC) II elaborado e apresentado como requisito parcial para obtenção do título de bacharel em Sistemas de Informação pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).

Orientador: Prof. M.e Fabiano Fagundes

Palmas – TO

2017

Fernando Silva Noleto

DESENVOLVIMENTO DE UM AMBIENTE WEB PARA CRIAÇÃO E CORREÇÃO AUTOMÁTICA DE EXERCÍCIOS DE ALGORITMOS DE ESCALONAMENTO DE PROCESSOS

Trabalho de Conclusão de Curso (TCC) II elaborado e apresentado como requisito parcial para obtenção do título de bacharel em Sistemas de Informação pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).

Orientador: Prof. M.e Fabiano Fagundes

Aprovado em: _____/_____/_____

BANCA EXAMINADORA

_____

Prof. M.e Fabiano Fagundes

Orientador

Centro Universitário Luterano de Palmas – CEULP

_____

Prof. M.e Madianita Bogo Marioti

Centro Universitário Luterano de Palmas - CEULP

_____

Prof. M.e Fernando Luiz Oliveira

Centro Universitário Luterano de Palmas - CEULP

Palmas – TO

2017

Dedico este trabalho primeiramente a Deus, a minha mãe Leide Pereira Silva, aos meus amigos e professores que pelo exemplo de vida, carinho e atenção, me incentivaram durante essa minha jornada acadêmica.

AGRADECIMENTOS

Agradeço primeiramente a Deus pela oportunidade de concluir um curso superior de Sistemas de Informação, sendo mais uma vitória na minha vida.

Agradeço também a minha mãe, Leide Pereira Silva, por sempre me apoiar e incentivar nos meus estudos.

Agradeço aos meus professores e professoras dos cursos de Sistemas de Informação e Ciência da Computação do CEULP/ULBRA: Cristina D'Onellas Filipakis, Jackson Gomes, Fábio Castro Araújo, Parcilene Fernandes de Brito (minha de orientadora de Estágio), Fernando Luiz de Oliveira e a Madianita Bogo Marioti (minha primeira orientadora). Agradeço as vocês professores, pelos os conhecimentos e lições que me passaram durante essa minha jornada acadêmica. Meu muito obrigado!

Agradeço ao meu orientador de TCC 1 e TCC 2, Fabiano Fagundes, que sempre me apoiou e incentivou neste trabalho de TCC, acreditando no meu potencial e na minha capacidade, em muitas vezes quando eu não acreditava em mim mesmo. Meu muito obrigado Fabiano, por toda sua paciência, atenção, críticas e lições que me passou durante esse tempo.

Agradeço também aos meus grandes amigos e amiga: Leonardo Daniel, Luã Lemos, Matheus Rodrigues, Tayse Virgulino, Francisco Maciel e o Ricardo Araújo. Agradeço pela fidelidade e a alegria. Meu muito obrigado.

RESUMO

NOLETO, Fernando Silva. **Desenvolvimento de um ambiente web para criação e correção automática de exercícios de algoritmos de escalonamento de processos**. 2017. 76 f. Trabalho de Conclusão de Curso (Graduação) – Curso de Sistemas de Informação, Centro Universitário Luterano de Palmas, Palmas/TO, 2017.

Este trabalho apresenta o desenvolvimento de um ambiente web que contempla o conteúdo de algoritmos de escalonamentos de processos, presente na disciplina de Sistemas Operacionais. Nesse ambiente, os algoritmos de escalonamento de processos são apresentados nas formas de exercícios e de suas resoluções, possibilitando a correção das respostas dos usuários através das indicações de seus acertos e erros. O usuário também poderá criar novos exercícios que serão acrescentados ao ambiente sendo sua resolução realizada de forma automática pelo sistema. A metodologia do trabalho deu-se com o estudo dos algoritmos de escalonamentos, troca de informações com professora da disciplina, verificação de ferramentas semelhantes já existentes e o estudo das tecnologias de desenvolvimento que permitem criar um ambiente web, como: JavaScript, AngularJs, WebStorage e Bootstrap. Ao final, chegou-se a um ambiente web responsivo que permite ao usuário utilizá-lo tanto em computadores como em dispositivos móveis para o estudo e aprendizado dos algoritmos de escalonamento de processos.

Palavras-Chave: Sistemas Operacionais; Exercícios; Correção.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Sistemas Operacionais (SOs) é uma das disciplinas obrigatórias nas matrizes curriculares nacionais nos cursos superiores da área de computação e Informática no Brasil, segundo o Ministério da Educação - MEC (BRASIL, 2012). A princípio o SO consiste em um conjunto de programas implementados e gerenciados para atuar entre a camada de aplicação do usuário (*software*) com o *hardware* da máquina, tornando este utilizável (DEITEL; DEITEL, 2005).

A disciplina de SOs apresenta um conjunto de conteúdos complexos e abstratos e tem uma subdivisão de seu conteúdo em quatro principais gerenciamentos. Nos quais consistem em gerenciamento de processos; de memória; de dispositivos de entrada/saída e de arquivos. Muitos dos conceitos, exemplos e exercícios vistos nesses gerenciamentos de SOs, apresentam um razoável grau de complexidade e abstração, o que torna, para alguns alunos, difícil a sua compreensão. Como é o caso do conteúdo algoritmos de escalonamento de processos e, objeto de estudo deste trabalho, presente no gerenciamento de processos.

Assim, esse trabalho apresenta os exercícios dos algoritmos de escalonamento de processos através de um sistema na *World Wide Web* (*web)*. Este sistema é composto pelas páginas de exercícios, resposta do usuário e a apresentação da correção final do exercício em questão. Dessa forma, o sistema oferece um auxílio para o aprendizado dos alunos quanto as resoluções dos exercícios dos algoritmos de escalonamento de processos.

Dessa maneira, serão apresentados, para entendimento do trabalho aqui descrito, os conceitos iniciais dos tipos de gerenciamentos existentes em SOs. Como é o caso do gerenciamento de processos, este que contém os quatro algoritmos de escalonamentos de processos: *First In First Out* (FIFO), *Shortest Job First* (SJF), Prioridade e Round-Robin. É com esses algoritmos de escalonamento de processos que serão exemplificadas as resoluções dos exercícios, bem como se dará a implementação das páginas de exercícios e suas resoluções. A forma como o ambiente poderá auxiliar o aprendizado dos algoritmos de escalonamento de processos são através de exercícios e suas resoluções, com indicações dos acertos e erros do usuário.

O presente trabalho foi estruturado da seguinte forma. A seção 2 apresenta o referencial teórico sobre o estudo de Sistemas Operacionais nos seguintes tópicos: Sistema Operacional (2.1), Gerenciamento de Processos (2.2) e Algoritmos de Escalonamento de Processos (2.3). No capítulo 3 é apresentada a metodologia e os materiais utilizados para o desenvolvimento do projeto. O capítulo 4 são apresentados os resultados obtidos pelo desenvolvimento, inclui dados escritos e visuais. Por fim as considerações finais são apresentadas no capítulo 5 e as referências posteriormente, no capítulo 6.

2 REFERENCIAL TEÓRICO

Nesta seção são abordados os conceitos referentes a Sistema Operacional, Gerência de Processos, ciclo de vida do processo, estados possíveis dos processos, escalonamentos de processos e os algoritmos de escalonamentos de processos.

2.1 Sistema Operacional

Sistema Operacional (SO) consiste "em uma camada de *software* colocada entre o *hardware* e os programas que executam tarefas para os usuários" (OLIVEIRA; CARISSIMI; TOSCANI, 2001, p. 1). Um Sistema Operacional gerencia todas as tarefas executadas pelos usuários, o que pode ser denominada de aplicação do usuário, e também controla todos os recursos do sistema, como é o caso do *hardware*. Uma das finalidades disso é garantir que o *hardware* execute os programas requisitados pela camada de aplicação do usuário.

Em um Sistema Operacional é importante a ideia de que *software* são os programas executáveis (OLIVEIRA; CARISSIMI; TOSCANI, 2001). É o caso dos editores de texto, navegadores e dentre outros presentes em um computador. Em relação ao *hardware* é o meio físico do computador, tais como os periféricos de entrada/saída, discos, memória e dentre outros. Dessa maneira, as aplicações do *software* precisam do *hardware* para serem executadas no computador.

Como Maziero (2014, p. 2) afirma, Sistema Operacional "é uma estrutura de *software* ampla, muitas vezes complexa, que incorpora aspectos de baixo nível (como drivers de dispositivos e gerência de memória física) e de alto nível (como programas utilitários e a própria interface gráfica)". Um computador depende do Sistema Operacional, pois sem este o computador não consegue controlar todos programas da máquina, como periféricos de entrada/saída, editores de texto, navegadores e dentre outros tantos de *software* ou *hardware.*

A função do Sistema Operacional é importante para um computador. Através do Sistema Operacional é que existe o controle dos *softwares* e dos *hardwares*, conforme as aplicações requisitadas pelos usuários ou pelo próprio Sistema Operacional. Isso significa que o Sistema Operacional evita que o computador paralise suas atividades, por causa de um conflito de demandas entre um *software* e um *hardware*. É o que afirma Jandl (2004, p. 5):

> "um sistema operacional é um programa, ou conjunto de programas, especialmente desenvolvido para oferecer, da forma mais simples e transparente possível, os recursos de um sistema computacional aos seus usuários, controlando e organizando o uso destes recursos de maneira que se obtenha um sistema eficiente e seguro".

O controle entre o *software* e *hardware* é feito através de recursos que o computador gera ao longo do tempo, onde o Sistema Operacional coordena os recursos durante o funcionamento do computador. Os recursos nesse caso seriam quaisquer componentes do *hardware* disputados pelos programas ou *software* (OLIVEIRA; CARISSIMI; TOSCANI, 2001, p. 2). Em Sistema Operacional, os recursos são gerenciados pelos quatros principais tipos de gerenciamento existentes: gerência de memória, de dispositivos de entrada e saída, de arquivos e de processos.

O gerenciamento de memória tem a tarefa de "monitorar quais partes da memória estão em uso e quais estão disponíveis; alocar e liberar memória para os processos" (CARDOZO; MAGALHÃES, 2002, p. 64). Durante o funcionamento do computador, vários processos solicitam acesso e alocação a memória, assim o Sistema Operacional gerencia os processos quanto ao acesso na memória, de maneira que não ocorra qualquer conflito de acesso entre os processos.

Outro gerenciamento é o de dispositivos de entrada/saída (E/S). Segundo Cardozo e Magalhães (2002, p. 89) sua tarefa "é controlar todos os dispositivos de entrada/saída (E/S) do computador, emitindo comandos para os dispositivos, atendendo interrupções e manipulando erros". Dessa maneira, o Sistema Operacional gerencia os recursos que possibilitam formas simples das aplicações dos usuários aos acessos de dispositivos do *hardware*, no momento que são requisitados na aplicação, como no caso de leitura e escrita em um disco rígido no computador, palavras digitadas pelos usuários no teclado e dentre outras formas de usos das partes físicas no computador;

O gerenciamento em arquivos, conforme Maziero (2014, p. 163) "é basicamente um conjunto de dados armazenados em um dispositivo físico não-volátil, com um nome ou outra referência que permite sua localização posterior". Assim, o Sistema Operacional gerencia os recursos para as atividades de armazenados de dados no disco físico não-volátil do computador, onde os dados poderão ser consultados;

O gerenciamento de processos é quando o Sistema Operacional administra os processos nas filas para acessarem a *Central Processing Unit* (CPU), sendo os processos compartilhados com o processador (OLIVEIRA; CARISSIMI; TOSCANI, 2001). Em Sistema Operacional, a CPU é uma unidade central de processamento, sendo parte física do computador, um *hardware.* Na gerência de processos, a CPU é controlada para executar os processos nos momentos adequados no processador, através de uma unidade de tempo. Dessa maneira, a parte de gerência de processos apresenta o objeto de estudo deste trabalho, onde é apresentado o

conteúdo de algoritmos de escalonamento de processos, que faz justamente o controle do tempo dos processos quando tentam acessar a CPU do computador.

2.2 Gerência de Processos

O processo é "basicamente um programa em execução" (TANENBAUM; WOODHULL, 1999, p. 26), onde o Sistema Operacional pode criar, remover, finalizar os processos quando for necessário para o funcionamento do computador. Os processos são criados a partir dos programas ou aplicativos que o usuário venha a executar no computador, que podem ser considerados como atividades dos usuários. Desta forma, o Sistema Operacional controla toda camada existente entre aplicação do usuário com a camada de *hardware.*

Em Sistema Operacional, os processos precisam dos recursos do computador para poder executar os programas dos usuários, sendo que os recursos são alocados aos processos no momento que estes são criados ou quando estão em execução (SILBERSCHATZ; GALVIN; GAGNE, 2013). Dessa maneira, no momento que os processos são criados a partir dos programas dos usuários, os processos precisam dos recursos além de serem alocado na CPU. Em relação ao acesso a CPU, os processos precisam fazer solicitações de acesso, pois é possível que existam outros processos na espera para serem executados.

A possibilidade de existir vários processos solicitando acesso a CPU faz com que o Sistema Operacional necessite de um controle de acesso, pois, como Jandl (2004) afirma, quando os processos são executados na CPU, demandam tempo e quantidade de espaço. Desta maneira, o Sistema Operacional precisa controlar os acessos dos processos a CPU, de forma que todos os processos executem, mas de forma controlada. Para isto, existe o conceito de multiprogramação, uma característica do Sistema Operacional quanto à gerência de processos no acesso dos processos a CPU do computador.

A multiprogramação é a "capacidade de armazenar o código de muitos processos na memória simultaneamente" (SHAY, 1996, p. 28). Em multiprogramação, o Sistema Operacional gerencia os acessos dos processos a CPU, de forma que os recursos existentes nos processos se tornam mais eficientes ao serem executados no computador, mesmo que estejam ocupando todo o tempo e espaço da CPU. Assim, a gerência de processos consiste no controle de vários processos quanto a alocação na CPU do computador, de modo que o SO utilize da característica da multiprogramação como fator de controle de acessos dos processos.

A gerência da multiprogramação é feita pelas concorrências entre os processos quanto ao acesso à memória da CPU (CARDOZO; MAGALHÃES, 2002). Por isso, existem as sequências de ações dos processos, onde um executa, outro espera e um outro processo finaliza

na CPU. Desta forma, através da multiprogramação o Sistema Operacional gerencia vários processos que solicitam a CPU do sistema; não existindo a multiprogramação, possivelmente a consequência para a CPU seria a inatividade de execução (OLIVEIRA; CARISSIMI; TOSCANI, 2001).

2.2.1 Ciclo de vida do processo

O ciclo de vida do processo constitui no momento da criação do processo até a sua destruição, momentos que são definidos pelo Sistema Operacional (OLIVEIRA; CARISSIMI; TOSCANI, 2001). Na maioria das vezes, os processos são criados a partir das demandas das aplicações dos usuários ou pelo próprio Sistema Operacional, sendo que o ciclo termina com a destruição do processo. Em escalonamento de processos, é importante o conceito do ciclo de vida para a definição do ciclo da CPU/processador e o ciclo de E/S dos processos.

A criação de um processo muitas vezes acontece quando o usuário, através das aplicações dos programas do computador, cria demandas ao Sistema Operacional, que responde através das criações de vários processos para executar um ou mais programas do usuário (TANENBAUM, 2016). Além disso, existem casos em que o próprio Sistema Operacional cria os seus próprios processos, sem que o usuário do sistema necessite executar algum programa no computador, ou quando os processos criam os seus próprios processos (TANENBAUM, 2016).

Após a criação do processo e ter acessado a CPU do computador, normalmente os processos são destruídos pelo Sistema Operacional, caso não tenham nenhuma outra atividade para ser executada. As condições para um processo ser destruído, segundo Tanenbaum (2016) seriam:

- saída normal: quando o processo executa todo o trabalho na máquina;
- erro fatal: processos encerram por uma atividade incorreta do usuário;
- saída por erro: causado pelo próprio processo, decorrente do programa em questão;
- morto por outro processo: um processo encerra outro processo, sendo comum em hierarquia de processos.

Durante o ciclo de vida, normalmente os processos fazem solicitações de acessos a CPU e passam a ocupá-la, mas também podem passar pela operação de E/S (Entrada e Saída), quando não estão acessando a CPU. Esse momento em que os processos estão acessando a CPU ou que realizam uma operação de E/S, são denominados respectivamente em ciclo de processador e

ciclo de E/S. “Os momentos em que um processo não está esperando por E/S, ou seja, em que ele deseja ocupar o processador, são chamados de “ciclos de processador” (OLIVEIRA *et al*, 2001, p. 15). Quando o processo está esperando por uma operação de E/S, ele está em um “ciclo de E/S””, sendo o ciclo de E/S um caso dos processos em Gerenciamento de Dispositivos de Entrada/Saída.

2.2.2 Estados de um processo

Os conceitos de multiprogramação e do ciclo de vida do processo definem como o processo se comporta no Sistema Operacional, do momento em que são criados até sua destruição. Durante o funcionamento do SO, muitos processos solicitam a CPU. Enquanto alguns processos esperam, outros estão em execução na memória da CPU. Essas condições dos processos em Sistemas Operacionais são definidas como estados possíveis dos processos (OLIVEIRA; CARISSIMI; TOSCANI, 2001). Em Sistema Operacional multiprogramado, os processos possuem estados possíveis para serem identificados, a fim do processador executar na CPU, como ilustra na Figura 1, onde é possível observar os processos que estão em execução no processador e outros que esperam para ser executados.

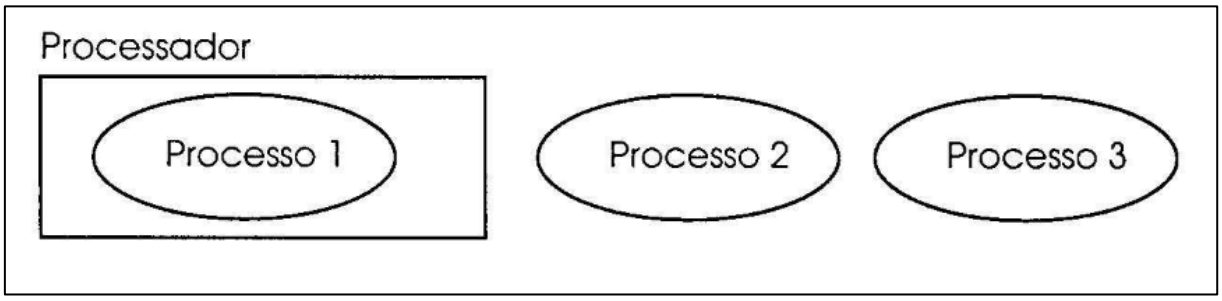


Figura 1 - Fila de processos

Fonte: Oliveira, Carissimi, Toscani, 2001 - pág 17

A Figura 1 mostra que o Processo 1 está alocado no processador, ou seja, está executando na CPU, enquanto que os Processos 2 e 3 esperam na fila de aptos para serem executados. Este é um exemplo básico de como a gerência de processo controla os acessos dos processos a CPU, mostrando especificamente os processos em espera e o executando.

Desta maneira, cada processo recebe o seu estado possível, na qual pode ser: criado, apto, bloqueado, executando e destruído. Ainda, o processo do Sistema Operacional responsável por coordenar os estados possíveis dos processos é o escalonador de processos, que faz o papel de troca os estados dos processos em escalonamento de processos. A Figura 2, demonstra o diagrama de estados de um processo.

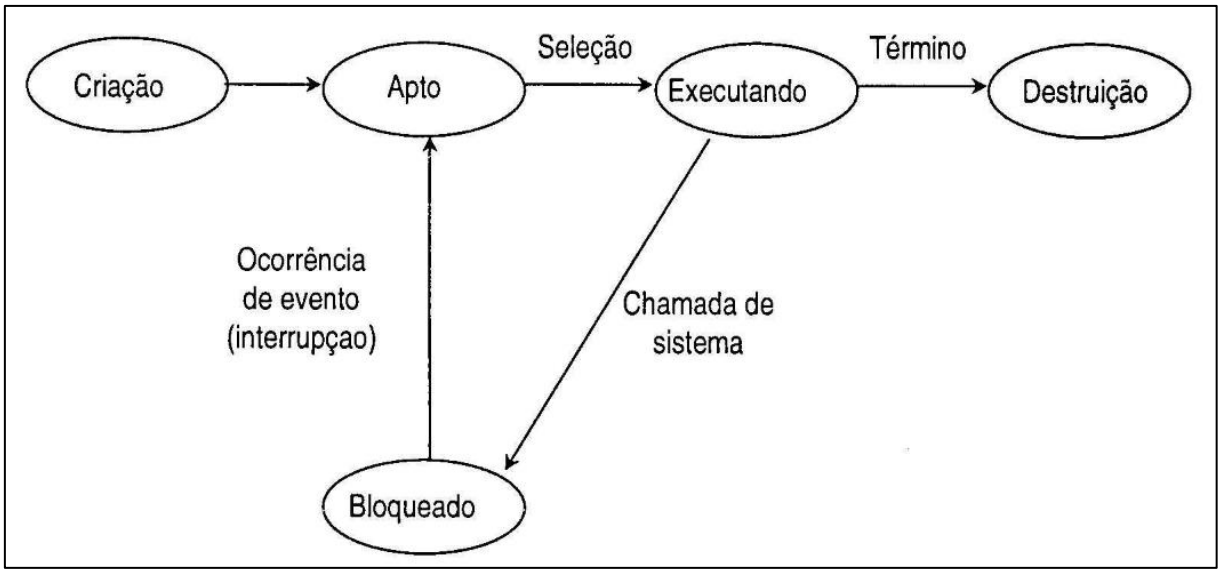


Figura 2 - Diagrama de Estados dos processos

Fonte: Oliveira, Carissimi, Toscani, 2001 - pág 17

Conforme a Figura 2, os estados possíveis para os processos seriam:

- criação: um novo processo é criado;
- apto: consiste no processo na fila de espera para ser executado pela CPU, sendo comum a definição de fila de aptos;
- executando: o processo está sendo executado na CPU no momento;
- bloqueado: o processo não está executando, ao ponto de esperar que aconteça alguma ocorrência ou uma nova espera na execução de E/S;
- destruição: o processo termina sua execução/atividade.

2.2.3 Escalonamento de Processos

Escalonamento de processos ou *Scheduling* da CPU é a base do Sistema Operacional multiprogramado que lida com a alternância de acessos dos processos a CPU do computador (SILBERSCHATZ; GALVIN; GAGNE, 2013). O uso de escalonamento de processos favorece uma performance do processador ao lidar com grandes quantidades de processos que são criados pelo Sistema Operacional, no momento de serem executados na CPU, tornando o computador mais produtivo.

Os processos são programas em execução, ou seja, qualquer atividade do usuário no computador no *software* faz com que novos processos sejam criados pelo Sistema Operacional. Assim, o escalonamento de processos tem que lidar com os novos processos que chegam do Sistema Operacional.

Deste modo, todos os processos que são gerados pelo Sistema Operacional precisam ser executados na CPU, sendo que os processos possuem um ciclo de vida em que representam os estados possíveis. Através do escalonamento de processos serão vistos os tipos de algoritmos que lida com as alternâncias entre os processos ao acesso a CPU.

2.3 Algoritmos de Escalonamento de Processos

Os algoritmos de escalonamento lidam com a decisão de qual processo na fila de aptos deve ser alocado na CPU (SILBERSCHATZ; GALVIN; GAGNE, 2013). As políticas de escalonamento dos Sistemas Operacionais baseiam-se nos algoritmos de escalonamento para alocação de processos na CPU, de forma que as políticas podem unir mais de um algoritmo.

Em todo caso, é interessante apresentar que a seleção de qual processo deve ser alocado na CPU em escalonamento de processos é feita pelo escalonador de processo ou *Scheduler* da CPU. O escalonador de processo tem a função de selecionar os processos na fila de aptos para executar na CPU, conforme a política de escalonamento do algoritmo em questão (SILBERSCHATZ; GALVIN; GAGNE, 2013).

O escalonamento pode ser preemptivo e não-preemptivo. A política de escalonamento preemptiva segundo Jandl (2004), é um tipo de algoritmo de escalonamento em que o escalonador pode retirar um processo em execução na CPU, mesmo que não tenha finalizado o seu ciclo de execução, para inserir outro processo na CPU, no momento que for necessário. Nos algoritmos do tipo preemptivo, o escalonador de processos estão constantemente alternando os processos na CPU, sempre visando melhor desempenho do computador.

Jandl (2004) afirma que a política de escalonamento não-preemptiva é quando os processos executam até o final do seu ciclo, e assim, liberam o acesso da CPU para o escalonador de processo selecionar outro processo. Nos algoritmos do tipo não-preemptivo, os processos seguem o ciclo de execução normalmente na CPU, sem qualquer interferência por parte do escalonador de processos.

Neste trabalho, serão apresentados os quatro algoritmos de escalonamento de processos, sendo eles: first come first server, shortest job first, prioridade e round-robin. Estes algoritmos serão apresentados nas próximas seções com conceitos e exemplos.

Inicialmente, é interessante a definição de alguns campos nas tabelas que serão vistos em todos os algoritmos de escalonamentos de processos referentes aos exemplos. A seguir são detalhados os campos da Tabela Inicial e da Tabela de Resultado:

- processo: lista de processos na fila de aptos;

- próximo ciclo (PC): tempo de duração do processo na CPU, a forma de unidade do tempo de duração varia de acordo com o algoritmo em questão;
- momento de transição (MT): momento em que o processo é inserido na fila de aptos;
- diagrama do tempo: representa cada tempo de duração dos processos em uma linha do tempo;
- tempo total (TT): representa o tempo de execução no diagrama do tempo subtraído pelo momento de transição do processo;
- tempo de espera (TE): tempo em que o processo fica na fila de apto.

Em todos os algoritmos, a Tabela de Resultado possui os valores de Tempo Total e Tempo de Espera, sendo que o Tempo Total (TT) consiste no Momento que o processo terminou no diagrama do tempo subtraindo com o Momento de Transição, localizado na tabela inicial. Após esse cálculo, é feito o preenchimento do Tempo de Espera (TE), onde o Tempo Total é subtraindo pelo valor do Próximo Ciclo localizado também na tabela inicial novamente.

2.3.1 *First In, First Out*

O algoritmo *First In, First Out* ou FIFO é considerado um algoritmo não-preemptivo, sendo um tipo de "algoritmo de escalonamento mais elementar que consiste em simplesmente atender as tarefas em sequência, à medida em que elas se tornam prontas" (MAZIERO, 2014, p. 50). Pela forma que o FIFO ordena os seus recursos, fica clara a sua simplicidade de execução do algoritmo e a sua simplicidade de escrita.

O FIFO consiste em uma fila simples, onde os processos são alocados pela ordem de chegada, assim os primeiros processos vão sendo processados pela CPU, e assim sucessivamente com os demais processos (OLIVEIRA; CARISSIMI; TOSCANI, 2001). O FIFO é considerado um tipo de algoritmo não-preemptivo, onde os processos executam até o fim, só liberando a CPU quando finalizam o ciclo de execução.

Conforme Maziero (2014), dependendo da característica da ordem de chegada dos processos, o FIFO pode apresentar problemas quanto ao tempo de médio de execução dos processos. Na situação em que a execução é de acordo com a chegada dos processos, casos os primeiros processos sejam muito grandes e sejam os primeiros na fila de aptos, pode custar muito tempo para os demais processos, principalmente se houver processos de menor tempo na espera. Apesar do problema, o FIFO não apresenta uma solução com base nas suas regras. Outros algoritmos que apresentam soluções em relação a esse erro serão vistos nas próximas seções.

Na figura a seguir, será apresentada uma tabela inicial do algoritmo FIFO, contendo os processos na fila de aptos, o seu próximo ciclo e o momento de transição, demonstrando um exemplo básico de funcionamento do algoritmo.

Tabela Inicial:

Processo	Próximo Ciclo (PC)	Momento de Transição (MT)
A	10	0
B	6	3
C	12	5
D	8	6

Figura 3 - Tabela Inicial do FIFO e do SJF

Como é possível verificar, a Figura 3 mostra os processos que são executados na CPU da tabela inicial. Como é o FIFO, os primeiros processos serão os primeiros a serem executados. Logo depois, os valores desses campos serão utilizados na tabela de Resultado.

Diagrama do tempo:

A	B	C	D

10 16 28 36

Tabela de Resultado:

Processo	Tempo Total (TT)	Tempo de Espera (TE)
A	10	0
B	13	7
C	23	11
D	30	22
Média	19	10

Figura 4 - Diagrama do Tempo e Tabela de Resultado Final do FIFO

A Figura 4 apresenta o Diagrama do Tempo e a Tabela Final do algoritmo FIFO. A descrição do diagrama e da tabela de Resultado podem ser descritos a seguir:

- o processo A chega no momento 0 e executa todo o seu tempo de ciclo na CPU;
- o processo B chega no momento 3, mas espera o processo A terminar a sua execução, e somente o processo A liberou a CPU, no momento 10, momento em que o processo B passou a executar na CPU;
- o processo C chegou no momento 5, mas esperou o processo B terminar e quando este liberou a CPU, no momento 16, o processo C passou a executar na CPU;

- o processo D chega no momento 6, mas tinha o processo A, B e C para executar na CPU, somente quando o último processo, o processo C terminou sua execução, no momento 28, foi possível o processo D executar o ciclo na CPU;
- visto que não tinha outro processo para executar na fila de aptos, o algoritmo encerra sua atividade.

O algoritmo FIFO é considerado não-preemptivo, pois a ordem de execução dos processos é conforme as chegadas dos processos na fila de aptos, sem qualquer interferência pelo escalonador de processos.

2.3.2 *Shortest Job First*

O *Shortest Job First* (SJF) é "um algoritmo em que os processos em espera pelo processador são organizados numa fila segundo seu tempo de execução, sendo colocados à frente os menores processos *jobs*" (JANDL, 2004, p.76). É um algoritmo que associa cada processo para acessar a CPU à verificação do tamanho do próximo processo. Pelas suas características de funcionamento, o SJF pode ser caracterizado como um algoritmo não-preemptivo.

O funcionamento do SJF consiste na verificação do tamanho do próximo processo da fila de tarefas, onde os processos menores são ordenados para serem executados primeiro, diferentemente do FIFO em que os primeiros processos que chegam são os primeiros a serem executados na CPU.

Essas particularidades do SJF o caracterizam como um algoritmo ideal para o escalonamento de processos, pois é interessante identificar o tamanho do próximo processo. Porém, o SJF não é possível implementar devido ser ainda difícil prever o próximo processo, quanto mais o seu tamanho. A figura a seguir apresenta uma demonstração de um exemplo básico do algoritmo SJF, onde a tabela inicial vem do algoritmo FIFO, apresentado anteriormente.

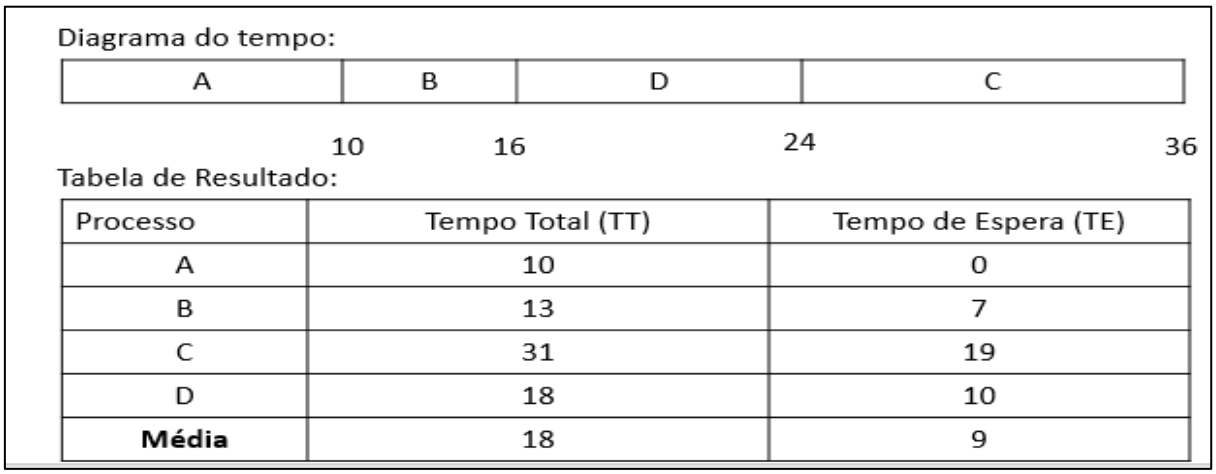


Diagrama do tempo:

A	B	D	C

10 16 24 36

Tabela de Resultado:

Processo	Tempo Total (TT)	Tempo de Espera (TE)
A	10	0
B	13	7
C	31	19
D	18	10
Média	18	9

Figura 5 - Diagrama do tempo e tabela de resultados final do SJF

Conforme a Figura 3, a tabela inicial do SJF neste exemplo é a mesma utilizada no Algoritmo FIFO, o Diagrama do Tempo e a Tabela Final do algoritmo SJF podem ser detalhadas na Figura 5 e logo abaixo a seguir:

- o processo A chega no momento 0 e executa todo o seu ciclo de execução na CPU;
- o processo B chega no momento 3, mas espera o processo A terminar e quando este liberou a CPU, no momento 10, o processo B passou a executar na CPU;
- o processo C chega no momento 5, visto que o Próximo Ciclo de C é maior que de B, esperou o processo A e B terminar e somente podia executar quando o processo B liberar a CPU, no momento 16;
- o processo D chega no momento 6, mas tinha processo A, B e C para executar na CPU, visto que o Próximo Ciclo do processo C é maior que processo D e que do processo B é menor que o processo D;
- o processo D passa a ter prioridade de execução em comparação ao processo C, porém ainda estava sendo executado o processo A, e tinha B na fila para ser executado na CPU, no momento;
- somente quando o processo B terminou sua execução, no momento 16, foi possível o processo D executar na CPU;
- quando o processo D terminou sua execução, no momento 24, foi possível para o processo C executar na CPU;
- visto que não tinha outro processo para executar na fila de aptos, o algoritmo encerra sua atividade.

No algoritmo SJF houve uma troca de posições na fila de aptos entre os processos C pelo D, pois o Próximo Ciclo do processo D é menor em comparação ao processo C, assim

houve uma troca de alocação na fila de aptos, sendo que a comparação de um processo ser menor que o outro é feito pelo tamanho do Próximo Ciclo (PC).

2.3.3 Prioridade

O algoritmo de Prioridade é quando “cada tarefa é associada a uma prioridade, na forma de um número inteiro, e os valores de prioridade são então usados para escolher a próxima tarefa a receber o processador, a cada troca de contexto” (MAZIERO, 2014, p.54). Este algoritmo pode ser considerado preemptivo e não-preemptivo, o que define ser um ou outro é a regra de implementação do algoritmo. Desta maneira, o algoritmo de Prioridades para cada processo define uma prioridade de execução, onde os processos possam ser executados nos recursos da CPU disponíveis, sendo o escalonador do Sistema Operacional responsável pela verificação das prioridades dos processos.

Nesse caso, é importante destacar como são definidas as prioridades para os processos, onde que são estabelecidas nas tarefas de prontas até escolher o próximo processo a ser executado. As prioridades são pré-definidas entre 0 até 6, onde o valor 0 representa uma maior prioridade e o valor 6 como uma menor prioridade.

O algoritmo de prioridade tanto pode ser caracterizado como preemptivo como não-preemptivo. No preemptivo a CPU pode paralisar um processo na linha de execução na CPU, e colocar outro no lugar dele, por ter mais prioridade. No não-preemptivo, é quando os processos executam normalmente sua execução até o fim. Nas figuras abaixo, são mostrados exemplos de como o algoritmo de prioridades funciona.

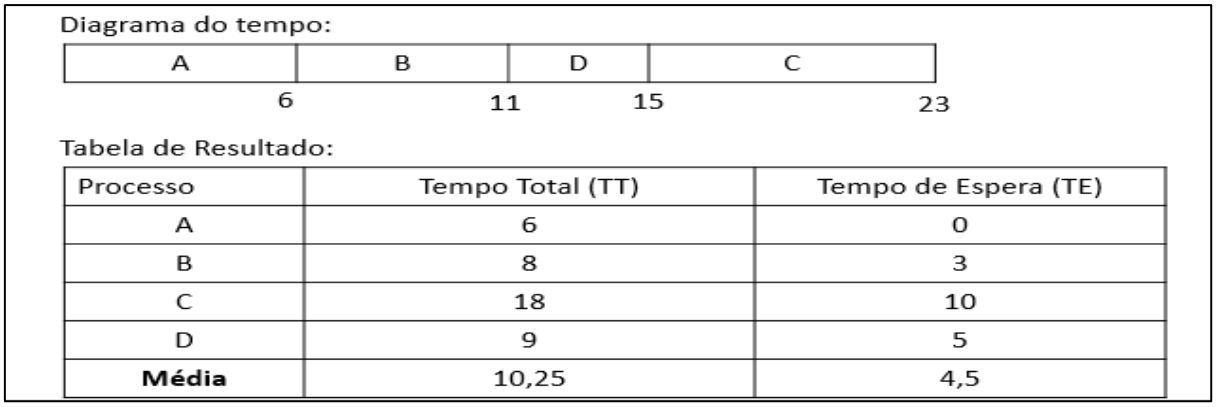
Tabela Inicial:

Processo	Próximo Ciclo (PC)	Momento de Transição (MT)	**Prioridade**
A	6	0	**1**
B	5	3	**0**
C	8	5	**3**
D	4	6	**2**

Figura 6 - Tabela inicial de Prioridades

A Figura 6 apresenta os valores dos campos da tabela inicial que servem tanto para o algoritmo não-preemptivo como para preemptivo, do algoritmo de Prioridade. As próximas figuras irão apresentar as diferenças entre o preemptivo e não-preemptivo.

Versão não-preemptiva:



Processo	Tempo Total (TT)	Tempo de Espera (TE)
A	6	0
B	8	3
C	18	10
D	9	5
Média	10,25	4,5

Figura 7 - Resultado final de Prioridade na versão não-preemptivo

A única mudança nesse caso, só seria a ordem de execução que será mostrada na Figura 7, no Diagrama do Tempo. O Diagrama do Tempo mostra os processos que foram executados, porém, o processo D foi primeiro que o C pois a sua Prioridade é menor que a C, no D é 2 já no C foi 3.

Desta maneira, o Diagrama do Tempo e a Tabela Final do algoritmo Prioridade da versão não-preemptiva podem ser descritos a seguir:

- o processo A, com prioridade 1, chega no momento 0 e executou todo o seu tempo de ciclo;
- o processo B, com prioridade 0, chega no momento 3, mas espera o processo A terminar e quando este liberou a CPU, no momento 6, o processo B passa a executar na CPU;
- o processo C, com prioridade 3, chega no momento 5, visto que sua prioridade é menor que B, não houve mudanças na fila de aptos e espera os processos A e B termina na CPU, para poder executar o ciclo, onde nesse momento seria no 11 no diagrama do tempo;
- o processo D, com prioridade 2, chega no momento 6, mas tinha os processos A, B e C para executar na CPU, visto que sua prioridade é maior que o processo C, porém menor que o processo B, houve uma troca de posição na fila de aptos entre os processos C e D, porém ainda estava sendo executado o processo A, e ainda tinha B na fila de aptos no momento;
- somente quando o processo B terminou sua execução, no momento 11, foi possível o processo D executar o seu ciclo na CPU;
- posteriormente, quando o processo D terminou sua execução, no momento 15, foi possível para o processo C executar o seu ciclo;

- visto que não tinha outro processo para executar na fila de aptos, o algoritmo encerra sua atividade.

Desta maneira, o algoritmo de Prioridades do tipo não-preemptivo ocorrem mudanças de posições entre os processos, somente na fila de aptos, onde o escalonador de processos sempre verifica as prioridades de cada processo.

Versão preemptiva:

Diagrama do tempo:

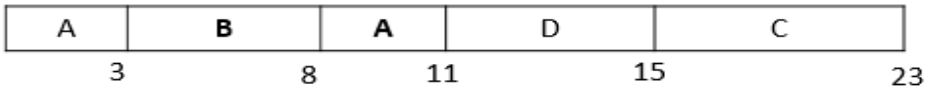


Tabela de Resultado:

Processo	Tempo Total (TT)	Tempo de Espera (TE)
A	11	5
B	5	0
C	18	10
D	9	5
Média	10,25	4,5

Figura 8 - Resultado final de Prioridade na versão preemptivo

Desta maneira, o Diagrama do Tempo e a Tabela Final do algoritmo Prioridade da versão não-preemptiva podem ser descritos a seguir:

- o processo A, com prioridade 1, chega no momento 0 e executa na CPU;
- o processo B, com prioridade 0, chega no momento 3, visto que tinha o processo A executando na CPU;
- porém, a prioridade do processo B é maior que o processo A, o escalonador de processos então encerra temporariamente a execução do processo A, e passa a CPU para o processo B, por possuir maior prioridade;
- o processo C, com prioridade 3, chega no momento 5, visto que sua prioridade é menor que A e B, passa a esperar a execução dos processos A e B na fila;
- o processo D, com prioridade 2, chega no momento 6, mas tinha os processos A, B e C para executar na CPU, visto que sua prioridade é maior que o processo C, porém menor que o processo A, houve uma troca de posição na fila de aptos entre os processos C e D, porém ainda estava na fila de aptos o processo A, e tinha o processo B executando na CPU;

- somente quando o processo B termina sua execução, no momento 8, foi possível o processo A executar o ciclo na CPU, ainda restavam na fila de aptos os processos D e C;
- quando o processo A termina o seu ciclo na CPU, no momento 11, o processo D executa na CPU;
- quando o processo D termina o seu ciclo na CPU, no momento 15, o processo C executa na CPU;
- finalizado o ciclo do processo C na CPU, no momento 23;
- visto que não tinha outro processo para executar na fila de aptos, o algoritmo encerra sua atividade.

A Figura 8 apresenta a versão preemptiva, em que ocorreram mudanças significativas em relação a versão não-preemptiva. Na Figura 8, como o processo B tem prioridade maior, sendo 0, quando iniciar algum processo, qualquer outro processo que tenha prioridade menor paralisa sua execução, e o processo de maior prioridade entra na CPU. Isto que aconteceu com o processo A, por ter menos prioridade, valor 1, quando B iniciou, o processo A parou sua execução na CPU, e cedeu o lugar ao processo B. Finalizado o processo B, não havendo outro processo com maior prioridade que A no momento, o processo A retorna para sua execução na CPU.

Existem casos no algoritmo de prioridade em que os processos com prioridades menores podem demorar muito tempo esperando para serem alocados na CPU, pois outros processos com maiores prioridades entram na fila e são alocados primeiros. Esse problema é resolvido, conforme SHAY (1996) afirma, através do algoritmo de envelhecimento, onde processos de menores prioridades vão tendo os seus valores de prioridades alterados, conforme o tempo, para que possam ser alocados, assim, em algum momento.

O algoritmo de envelhecimento funciona ao passar do tempo, quando o processo vem envelhecendo conforme as alterações ocorridas nos valores de prioridades dos processos. É o que mostra a demonstração do algoritmo de envelhecimento nas seguintes figuras.

Tabela Inicial:

Processo	Próximo Ciclo (PC)	Momento de Transição (MT)	**Prioridade**
A	3	0	**1**
B	6	3	**0**
C	10	6	**3**

Figura 9 - Tabela inicial do algoritmo de Prioridade quando ao envelhecimento

Na Figura 9, os valores da tabela inicial estão preenchidos com valores normais para o algoritmo de prioridades, com os dados dos processos: próximo ciclo; momento de transição e a prioridade. As mudanças que ocorrem com o algoritmo de envelhecimento em prioridade, podem ser visualizados na Figura a seguir.

Tabela com acréscimo de outros processos:

Processo	Próximo Ciclo (PC)	Momento de Transição (MT)	**Prioridade**
A	3	0	**1**
B	6	3	**0**
C	10	6	**3**
D	3	7	**1**
E	4	10	**1**
F	2	16	**0**

Processo C
3 para 2
3 para 1

Figura 10 - Tabela de acréscimo de processos

Na Figura 10, os processos da tabela inicial permanecem, porém, novos processos são acrescentados na fila de espera, sendo que esses processos D, E e F têm prioridade maior que o processo C. Assim, o algoritmo de envelhecimento subtrai a prioridade do processo C ao longo do tempo, como mostra a figura a seguir.

Diagrama do Tempo:

A	**B**	**D**	**E**	**F**	**C**
3	9	12	16	18	28

Figura 11 - Diagrama do tempo de envelhecimento

A Figura 11 apresenta o diagrama do tempo do algoritmo de envelhecimento com as prioridades para serem executados na CPU.

Tabela Final:

Processo	Tempo Total **(TT)**	Tempo de Espera (**TE**)
A	3	0
B	6	0
C	22	12
D	6	3
E	6	2
F	2	0
Média	**7,5**	**3,67**

Figura 12 - Tabela final do algoritmo de envelhecimento

Desta maneira, o Diagrama do Tempo e a Tabela Final do algoritmo de prioridade sobre o algoritmo de envelhecimento podem ser descritos a seguir:

- o processo A, com prioridade 1, chega no momento 0 e executa na CPU;
- o processo B, com prioridade 0, chega no momento 3, visto que o processo A tinha terminado o seu ciclo na CPU;
- o processo B passa a executar na CPU;
- o processo C, com prioridade 3, chega no momento 6, visto que sua prioridade é menor que o processo B, passa a esperar a execução do processo B na fila;
- um novo processo, denominado D, é criado no momento 7 com prioridade 1, maior que o processo C, único que está na fila de aptos no momento, então ocorre uma mudança de posição na fila de aptos entre os processos C e D, agora, quando o processo B termina, o processo D executa primeiro;
- o processo B termina no momento 9, sendo que nas fila de aptos no momento estão os processos D e C para serem executados na CPU, então o processo D passa a executar na CPU no momento 9;
- nesse momento, a prioridade do processo C é subtraída pelo algoritmo de envelhecimento;
- um novo processo, denominado E, é criado no momento 10 com prioridade 1, maior que o processo C, único que está na fila de aptos no momento, então ocorre uma mudança de posição na fila de aptos entre os processos E e C, agora, quando o processo D termina, o processo E executa primeiro;
- o processo D termina o seu ciclo na CPU, no momento 12;
- o processo E executa o seu ciclo na CPU, no momento 12;

- nesse momento, a prioridade do processo C é subtraída pelo algoritmo de envelhecimento;
- o processo E termina o seu ciclo na CPU, no momento 16;
- no momento 16, um novo processo é criado, denominado de processo F com prioridade 0;
- visto que tinha o processo C na fila de aptos, e como o processo F tem prioridade maior que o processo C, então o escalonador de processos seleciona o processo F para executar primeiro na CPU;
- o processo F termina o seu ciclo na CPU, no momento 18;
- o processo C executa na CPU, no momento 18;
- o processo C termina o seu ciclo na CPU no momento 28;
- visto que não tinha outro processo para executar na fila de aptos, o algoritmo encerra sua atividade.

A Figura 12 apresenta a tabela final do algoritmo de envelhecimento, em que alguns processos ao longo da linha do tempo têm suas prioridades alteradas, decorrente do algoritmo de envelhecimento. Existem casos, como os acréscimos de outros processos, em que ocorrem mudanças no Tempo Total e de Espera, que podem acontecer tanto com ou sem envelhecimento no algoritmo de prioridade.

2.3.4 Round-Robin

O Round-Robin é um algoritmo em que “cada processo recebe uma fatia de tempo do processador (*quantum*), podendo ser implementado através de uma fila simples, semelhante ao FIFO” (OLIVEIRA; CARISSIMI; TOSCANI, 2001, p. 67). O Round-Robin é considerado um tipo de algoritmo preemptivo. Nesse algoritmo, o fator do tempo se refere na unidade de tempo ou fatia de tempo, sendo definido na tabela inicial dos processos, e todos os processos executam na CPU conforme o valor definido pelo *quantuam*. Assim, o escalonador da CPU consegue percorrer toda a fila de processos, controlando os processos na hora de executar na CPU através do *quantum*.

Além disso, o algoritmo Round-Robin todos os processos são armazenados em uma fila circular. Assim, cada vez que o escalonador passa na fila de processos, seleciona um processo para executar na CPU e os outros processos, caso existam, esperam o processo atual executar na CPU. Dessa forma, sempre fica um ou outro processo esperando o retorno do processo atual

na CPU. Durante esse tempo, é possível surgir novos processos para a fila de aptos do algoritmo.

Dessa forma, o Round-Robin é um algoritmo de melhor média final entre os algoritmos de escalonamento (JANDL, 2004). Para isso, basta verificar a sua distribuição de carga de trabalho, devido ao controle de execução que tem sobre todos os processos. Por isso, é considerado um algoritmo preemptivo, pois consegue gerenciar o tempo de execução dos processos.

São apresentados nas Figuras, exemplos de execução do algoritmo de Round-Robin:

Tabela Inicial

Processo	Próximo Ciclo (PC)	Momento de Transição (MT)
A	6	0
B	5	3
C	8	5
D	4	6

Quantum = 4

Figura 13 - Tabela inicial de Round-Robin

A Figura 13 apresenta a Tabela Inicial do algoritmo Round Robin, com as listas de processos e o valor do *quantum* utilizado neste exemplo.

Diagrama do tempo:

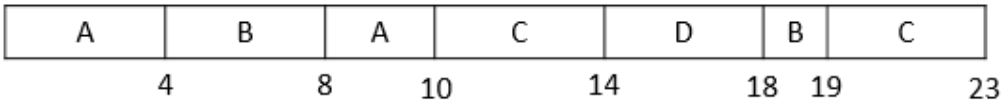


Tabela de Resultado:

Processo	Tempo Total (TT)	Tempo de Espera (TE)
A	10	4
B	17	12
C	20	12
D	12	8
Média	14,75	9

Figura 14 - Resultado final de Round-Robin

Nas Figuras 13 e 14, a definição do *quantum* ficou estabelecida em 4, onde cada processo executa 4 vezes na CPU. A Figura 13 mostra a Tabela Inicial que se inicia com os dados dos processos de Próximo Ciclo e Momento de Transição. Na Figura 14 é apresentado o

Diagrama do Tempo quando é alterado conforme o valor do *quantum*, onde são vistos que os processos estão sendo executados 4 vezes nas unidades da linha do tempo e, assim, vão sendo finalizados. Ainda na Figura 14 é apresentada a Tabela de Resultado em que se observam os valores do Diagrama do Tempo e da Tabela Inicial, além dos preenchimentos dos valores de Tempo Total e do Tempo de Espera, realizando os mesmos processos vistos nos algoritmos anteriores.

Desta maneira, o Diagrama do Tempo e a Tabela Final do algoritmo Round-Robin pode ser descrito a seguir:

- o processo A chega no momento 0 e executa 4 unidades de tempo na CPU, conforme o *quantum* estabelecido na Tabela Inicial, ainda restando o seu tempo de ciclo e entrou na fila de aptos, pois já tinha o processo B na fila;
- o processo B chegou no momento 3, esperou o processo A terminar e quando este liberou a CPU, no momento 4, o processo B passou a executar 4 unidades de tempo na CPU, restando ainda o seu ciclo, entrou na fila de aptos, onde tinha os processos A, C e D na fila de aptos;
- o processo A inicia de novo no momento 8, termina todo o seu ciclo, e passa para outro processo na fila de aptos, onde estão os processos na seguinte ordem C, D e B;
- o processo C chegou no momento 5, mas tinha os processos A e B executando, então ele inicia no Diagrama no momento 10, executa 4 unidades de tempo, restando o seu ciclo de execução, entra na fila de aptos, onde estão os processos D e B;
- o processo D chegou no momento 6, mas tinha os processos A, B e C executando na CPU, então ele inicia no Diagrama no momento 14, executa 4 unidades de tempo, terminando o seu ciclo de execução, passa para outro processo na fila de aptos, onde estão os processos B e C;
- o processo B inicia de novo no momento 18, termina sua execução;
- o processo C inicia de novo no momento 19, e termina sua execução.

Visto que não tinha outro processo para executar na fila de aptos, o algoritmo encerra sua atividade.

3 MATERIAIS E MÉTODOS

Esse trabalho teve como objetivo desenvolver um ambiente *web* sobre um conteúdo presente na disciplina de Sistemas Operacionais. Tal conteúdo se refere aos algoritmos de escalonamento de processos, onde são apresentados, especificamente, os exercícios e suas correções dos quatro principais algoritmos de escalonamento de processos. Dessa forma, a metodologia adotada para este projeto envolveu diversas fases de atividades de pesquisa, escrita e o desenvolvimento do ambiente *Web*, fases que serão descritas a seguir.

3.1 Desenho do Estudo

Para que o propósito desse trabalho, de implementar um ambiente *web* que execute e corrige as respostas dos usuários sobre os exercícios referentes aos algoritmos de escalonamento de processos fosse atingido, fez-se uso de um desenho de estudo para o processo do planejamento até a construção do ambiente, como ilustrado na Figura 15.

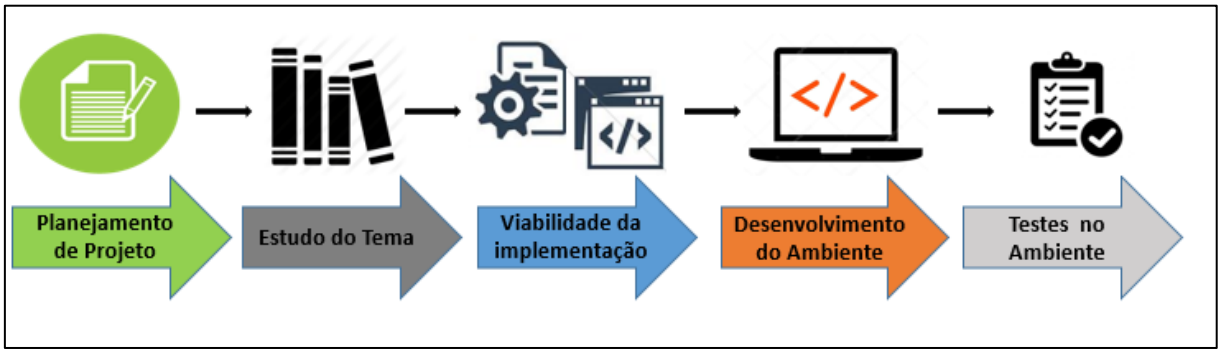


Figura 15 - Desenho do Estudo

Na Figura 15, o desenho de estudo inicia-se na etapa do planejamento do projeto que consistiu na definição do objeto de estudo deste trabalho, sendo este a disciplina de Sistemas Operacionais. Ainda, foram definidas as ferramentas de desenvolvimento *web* para a construção do ambiente proposto.

Após o planejamento inicial, a próxima etapa consistiu na definição de um conteúdo de Sistemas Operacionais para que fosse apresentado neste trabalho, que foi sobre os algoritmos de escalonamentos de processos. Após a escolha do conteúdo, foram realizadas leituras bibliográficas em pesquisas, artigos científicos, teses e livros da área de Sistemas Operacionais, além do estudo sobre as ferramentas utilizadas para a construção do ambiente *Web*. Além de conversas com a professora Madianita Bogo Marioti, responsável pela disciplina de Sistemas Operacionais nos cursos de Sistemas de Informação e Ciência da Computação do CEULP/ULBRA.

Após o estudo do conteúdo, a próxima etapa consistiu no planejamento da viabilidade da implementação de um ambiente *web*, que consistiu no desenvolvimento de um protótipo do ambiente, conforme mostrado no Apêndice A. Inicialmente, os protótipos do ambiente apresentam os campos de exercícios e das resoluções, mostrando os modelos das correções feitas pelo ambiente.

A próxima etapa consistiu no desenvolvimento propriamente dito do sistema, em que as informações adquiridas nas etapas anteriores foram aplicadas para atingir o objetivo deste trabalho. Nessa etapa, foram utilizadas as ferramentas para desenvolvimento *web*, adiante detalhadas, baseando nos protótipos desenvolvidos no Apêndice A.

A próxima etapa consistiu em testes funcionais do ambiente *Web* quanto aos objetivos gerais e específicos levantados neste trabalho, a fim de que o ambiente desenvolvido atingisse o objetivo de executar e avaliar as resoluções dos exercícios dos algoritmos de escalonamento de processos. Os testes funcionais avaliaram os exercícios e as resoluções dos exercícios dos algoritmos de escalonamento de processos.

3.2 Softwares

3.2.1 HTML 5

Segundo Flatschart (2011) é uma linguagem de marcação de hipertexto, utilizada para formatar a estrutura de apresentação dos conteúdos/informações na *web* para a interface do usuário. O HTML 5 possui recursos que definem a estrutura e a organização da página *Web*, de forma que as apresentações sejam o mais simples possível. Além de tornar possível integrar outras tecnologias de desenvolvimento web, como a linguagem de programação *JavaScript* (FLATSCHART, 2011).

3.2.2 JavaScript

O *JavaScript* (JS) é uma linguagem de programação interpretada, tipada e orientada a objeto para desenvolvimento *web*, que é intregada ao *HyperText Markup Language* - HTML (FLANAGAN, 2011). A integração do *JavaScript* ao HTML é feita na própria página de desenvolvimento, sendo necessário declarar a *tag* `<script>` na linha de código do HTML ou através de uso de um framework, sendo nesse caso ter uma referência ao framework no HTML, como é o caso do *AngularJS*. Porém, tanto o uso da tag `script` como o do framework, a aplicação *web* com a integração do *JavaScript* ganha mais funcionalidades para serem utilizadas, o que torna a programação mais interativa.

Segundo Grillo e Fortes (2008), as funcionalidades do *JavaScript* para o desenvolvimento *web* são diversas e simples de serem implementadas. As funcionalidades vão desde os elementos e recursos mais básicos ao mais complexos, estes que permitem criar objetos, animações manipulação de páginas e dentre outros para serem usados como interação ao usuário. Contudo, o *JavaScript* conta com uma documentação completa sobre desenvolver na, onde é apresentada toda a regra lógica e sequencial da linguagem.

Assim sendo, a linguagem *JavaScript* neste trabalho foi utilizada para gerenciar as regras lógica de funcionamento dos algoritmos de escalonamento de processos. Para cada algoritmo, a linguagem será responsável por receber os valores tanto dos usuários como do sistema, este que o sistema receberá da tabela inicial. Após as entradas dos processos, são feitos os cálculos sobre cada processo da tabela inicial, referentes ao tempo total e o tempo médio, conforme os atributos da tabela de resultado (nome do processo, tempo total e tempo médio). No final, é gerada uma saída contendo o nome do processo, o tempo total e o tempo de cada processo da tabela inicial, sendo estes valores comparados com a respostas dos usuários, posteriormente.

Por fim, todas as resoluções dos exercícios dos processos referente aos algoritmos de escalonamento são calculados pela linguagem de programação *JavaScript.* Desta forma, para mostrar dados dos resultados aos usuários, os dados em *JavaScript* foram distribuídos conforme a estrutura definida em HTML5 e no *Bootstrap*.

3.2.3 Hypertext Preprocessor (PHP)

O *Hypertext Preprocessor* (PHP) é "uma das linguagens de script mais populares para a criação de páginas *Web* dinâmicas no lado do servidor" (DEITEL, DEITEL, 2005, p. 462). O PHP dispõe de um código fonte aberto (*open-source*) além de diversos recursos para programação *Web* na qual se encontra disponível em sua documentação[1], o que auxilia as aplicações *Web* desenvolvida em PHP.

O PHP é representado em uma aplicação *Web* quando no final do arquivo estiver a sigla ".php", o que significa que está integrado à aplicação *Web* em questão. Após a integração, a programação em PHP pode compor outras tecnologias para a aplicação, tais como o HTML, *JavaScript* e dentre outras. Há casos em que o PHP se integra a aplicação *Web* de forma isolada, como por exemplo, servir apenas para realizar inserções ou consultas básicas ao banco de dados. É o caso semelhante neste trabalho, onde serve apenas para se comunicar ao banco de dados.

[1] Referência ao site do PHP contendo a documentação em Português, disponível em:< http://php.net/manual/pt_BR/index.php>.

O Deitel e Deitel (2006) confirmam que o PHP é uma ferramenta voltada para ser executada no lado do servidor, diferentemente do *JavaScript* que é executado no lado do cliente, por exemplo. Mas isto não significa que o PHP será utilizado apenas para realizar operações ao lado do banco de dados em uma aplicação. Existem situações em que o PHP é presente em toda aplicação, desde a parte da base de dados, da lógica de negócio e até a parte da visualização da página para o cliente. Claro, o PHP recebe o auxílio de outras tecnologias, como o *JavaScript* para trabalhar na parte lógica de negócio ou HTML e *Bootstrap* na parte de visualização da página do cliente.

Neste trabalho, o PHP é utilizado especificamente para realizar operações no banco de dados, como as consultas e as inserções a base de dados, pois o PHP, como uma linguagem ao lado do servidor, possui suporte para vários tipos de banco de dados, como o Oracle, PostreSQK, SQL Server e MySQL. Então, a escolha do PHP para este trabalho foi justificada por esta razão: servir de comunicação e gerenciamento ao banco de dados.

Assim sendo, através das instruções de código em PHP, foi possível realizar inserções e consultas ao banco de dados, distribuindo desta maneira: quatros consultas e uma inserção a base de dados. As quatros consultas foram distribuídas nos quatros algoritmos de escalonamento de processos: FIFO, SJF, Prioridades e Round-Robin. Cada consulta nesses algoritmos representava uma lista de exercícios sobre o algoritmo escolhido em questão.

A parte de inserção, o que algoritmo de PHP faz é armazenar um exercício por vez, sendo necessário o usuário passar como informação, uma lista de processos e o nome do algoritmo. Após isto, o usuário pode salvar no banco de dados o exercício.

3.2.4 AngularJs

O *AngularJs* é um framework do *JavaScript* para criação de aplicações *web* dinâmicas e interativas, que utiliza da página em HTML para prover sua extensão de aplicação (SCHMITZ, LIRA, 2016). O *AngularJs* é mantido pela Google[2] e distribuído pelo Apache[3], contendo o código fonte livre para qualquer aplicação na *web*. Por isso, que muitas aplicações desenvolvidas em *AngularJs* são compatíveis com as maiorias dos navegadores *web*.

Por padrão, o framework *AngularJs* para ser utilizado em qualquer aplicação *web*, precisa de sua referência nas linhas de códigos, como ocorre em HTML. Após a referência, novas funcionalidades e recursos de programação são disponíveis através da biblioteca do *AngularJS*, e tais recursos são disponíveis para serem implementadas na aplicação *web*.

[2] Referência ao site Angular mantido pela Google, disponível em: < https://angularjs.org/>.
[3] Referência ao site das Licenças Apache, disponível em: < http://www.apache.org/licenses/>.

O *AngularJs* como um framework padrão do *JavaScript*, segue um princípio básico, como é afirmado pelo Schmitz e Lira (2016, pág. 7): “para visualização de dados e informações, ele não possui a funcionalidade de prover dados dinâmicos, ou persistir informações em um banco de dados”. Assim, o *AngularJs* facilita que novas informações possam ser apresentadas de forma mais simples e interativas.

Neste trabalho, o *AngularJs* foi utilizado como um framework da linguagem de programação *JavaScript*. Através do *AngularJs* foram trabalhados diversos recursos e funcionalidades de programação de sua biblioteca padrão. Assim, com o *AngularJs* foi possível adicionar novas funcionalidades tanto no que se refere à construção de tabelas quanto como um meio de auxílio das resoluções dos exercícios dos algoritmos de escalonamento, juntamente com o *JavaScript*.

3.2.5 Bootstrap

O *Bootstrap* é um framework do *JavaScript* e HTML para projetos de desenvolvimento em aplicações *web*, tornando-as responsivas (SILVA, 2015). O framework *Bootstrap* permite a criação de novas páginas *web* responsivas e interativas, onde a aplicação *web* funciona em qualquer dispositivo, conforme o seu tamanho. Ou seja, a aplicação fica responsiva tanto para telas pequenas, como os celulares como para telas maiores, como os computadores e tablets.

O *Bootstrap* é muito aplicado para desenvolvimento *web* em *front-end*, em que a programação se preocupa com a parte visual da aplicação. Assim, com os recursos do Boostrap juntamente com o HTML, torna-se possível desenvolver *layouts* responsivos para todas as apresentações em uma página *web* (SPURLOCK, 2013).

O framework *Bootstrap* é utilizado também para desenvolvimento *mobile-first* (SILVA, 2015). Sendo assim, o desenvolvimento *web* são as aplicações suportadas por navegadores *web* quando acessados em aparelhos de formato maiores, como os computadores ou *notebooks*. Enquanto o *mobile-first*, são aplicações que envolvem o desenvolvimento de páginas para dispositivos menores, como os celulares. Essa transição de formato de maiores para menores ou vice-versa, o *Bootstrap* através de suas configurações de responsividades, suportadas pelo framework.

Dessa maneira, o uso do *Bootstrap* neste projeto foi voltado para criação de uma interface *web* interativa e responsiva, para justamente possibilitar ao usuário acessar com qualquer dispositivo. Além disso, o framework foi usado para configurar os formatos dos recursos das tabelas como imagens e ícones do HTML, facilitando o processo de explicação das respostas dos algoritmos de escalonamento de processos.

3.2.6 MySQL

O MySQL é um sistema de gerenciamento de banco de dados[4] (SGBD) relacional de código aberto, projetado para trabalhar com aplicações de pequeno, médio e grande porte (Milani, 2006). O MySQL faz o uso da linguagem *Structured Query Language*[5] (SQL), baseada em operações no banco de dados, como inserções, consultas, exclusões, atualizações e dentro outras operações.

O MySQL dispõe de duas licenças de usos, sendo uma licença livre e a outra comercial. A licença livre, conforme Milani (2006, p. 23) "é baseada nas cláusulas da GNU-GPL (*General Public Licence*), o qual estabelece o que se pode ou não fazer com a ferramenta e demais recursos". A licença livre dispõe das operações básicas para o gerenciamento de banco de dados.

A licença comercial, como diz Milani (2006), oferece serviços adicionais ao banco de dados como ferramentas ou pacotes, serviços que podem gerenciar aplicações comerciais de médio e grande portes, visto também para pequeno porte. Ou seja, a licença comercial tem um suporte diferencial comparado a licença livre.

O MySQL como um gerenciador de banco de dados tem algumas operações que são comuns comparadas aos outros bancos de dados, como o SQL Server ou Oracle. É o caso já citado dos armazenamentos, recuperações e atualizações de dados além do compartilhamento dessas informações. Estas operações são executadas pelo administrador do banco de dados ou responsável através do gerenciamento na base de dados, sendo inclusive uma das maneiras das realizações desses gerenciamentos através das instruções em códigos, como o uso da linguagem PHP, por exemplo.

Neste trabalho, o gerenciamento do MySQL foi realizado através de comandos da linguagem do PHP para realizar as instruções em SQL no banco de dados, sendo a versão do MySQL utilizada neste trabalho a livre, pois dispõe de todas as operações necessárias para realização deste trabalho.

Desta maneira, através das instruções em PHP foi possível realizar operações ao banco de dados, como a inserções e consultas aos dados relacionados aos algoritmos de escalonamento de processos. Essas inserções e consultas foram realizadas em uma única tabela criada dentro

[4] "Um sistema de gerenciamento de banco de dados, ou SGBD, é um software projetado para auxiliar a manutenção e utilização de vastos conjuntos de dados" (RAMAKRISHNAN, GEHRKE; 2011; p. 3)

[5] Referência ao site da *Structured Query Language Server*, disponível em: < https://www.microsoft.com/pt-BR/sql-server/>.

do banco de dados do MySQL. Esta tabela criada serviu para armazenar as informações da tabela inicial de um exercício de escalonamento de processos, sendo caso, os atributos comuns: nome do processo, próximo ciclo, momento de transição, prioridades e o nome do exercício.

2.3.7 Web Storage

O *Web Storage* é uma das APIs (ou *Application Programaming Interface*[6]) do HTML5 que oferece o suporte para armazenamento de dados e informações localmente no próprio navegador do usuário, quando este acessa uma aplicação *Web* contendo essa tecnologia (STUTZ, 2013). É o caso em que o navegador do cliente torna-se um banco de dados próprio, armazenando os dados localmente no próprio navegador, sem muitas vezes comprometer o desempenho da página *Web*.

Conforme Stutz (2013), o *Web Storage* é um objeto de dados em que estes são guardados localmente no navegador do cliente, podendo às vezes substituir por algum momento a necessidade de um banco de dados. Os dados ficam armazenados em um formato de objetos no *Web Storage*, seguindo um modelo do tipo chave e valor. A chave seria um identificador do objeto, e este seria os conjuntos de dados armazenados em questão.

A API do *Web Storage* implementa dois modos de armazenamento locais de dados, sendo eles o *Session Storage* e o *Local Storage.* Conforme Iamashita e Magalhães (2013), os dois modos de armazenamentos de dados, tanto o *Session* como o *Local Storage*, dispõem da mesma API de interface de programação, com os mesmos atributos e métodos. Mas, a diferença entre os dois modos está na questão do tempo de expiração dos dados armazenados localmente.

Sobre o *Session Storage*, Yamashita e Magalhães (2013) afirmam que o armazenamento de dados é temporário no navegador do usuário. Os dados são armazenados quando o usuário acessa uma aplicação *Web*, e quando finaliza/fecha a aplicação *web* ou o próprio navegador, os dados apagam do navegador. Assim, somente é possível armazenar os dados novamente, quando o usuário acessa a mesma aplicação *web*. Enquanto no *Local Storage* os armazenamentos de dados são permanentes no navegador do usuário, sem data de expiração para os dados armazenados, mesmo quando o usuário fecha o navegador ou a página *web* (IAMASHITA, MAGALHÃES, 2013).

Desta forma, o modo do *Web Storage* utilizado nesse trabalho foi o *Session Storage*. Ela serviu para armazenar as respostas dos usuários, os exercícios de algoritmos dos

[6] *Application Programaming Interface* é "uma interface de programação de aplicativos padronizada (ou API), implementado de forma nativa nos navegadores *web*, sem a necessidade da instalação de *plugins*" (STUTZ, p. 4, 2013)

escalonamentos de processos, o algoritmo escolhido para avaliação, dentre outras informações importantes para apresentar no sistema. A escolha do *Session Storage* foi porque ela armazena os dados temporariamente no navegador do usuário, e quando este finaliza o sistema (aplicação *Web*), os dados se expiram.

3.3 Arquitetura da Aplicação

O funcionamento do sistema *Web* que foi implementado apresenta a seguinte arquitetura, como mostra a Figura 16.

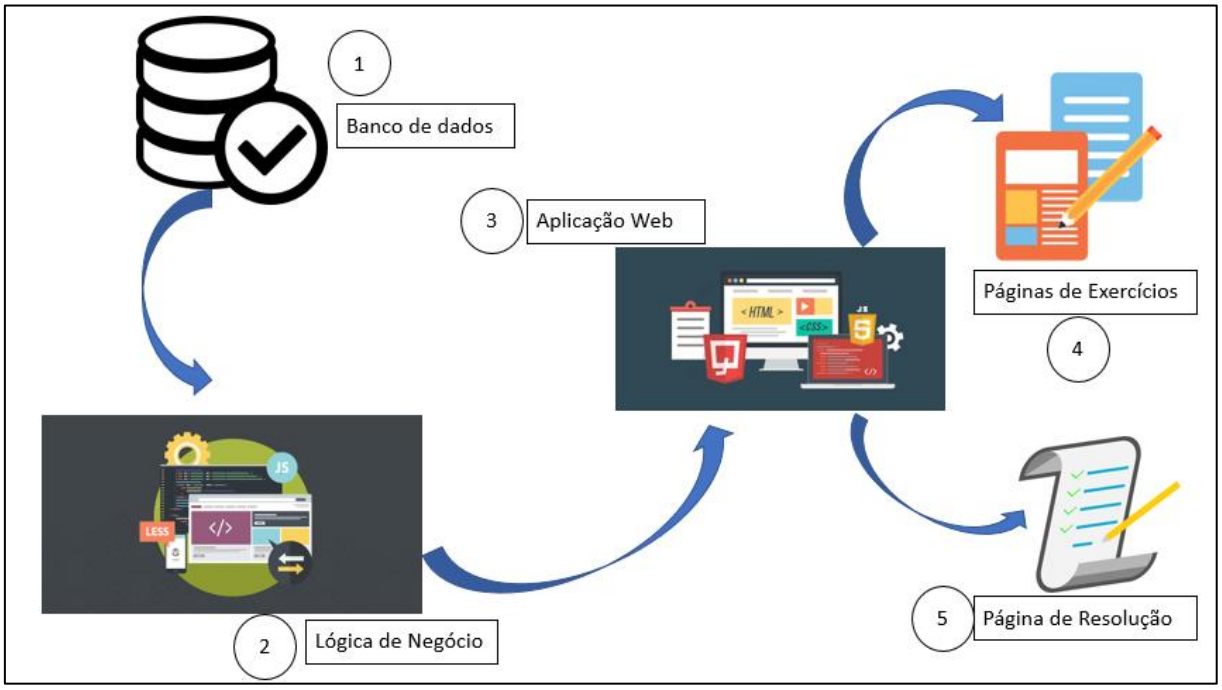


Figura 16 - Arquitetura de Software

Conforme ilustra a Figura 16, a arquitetura se compõe em Banco de dados, Lógica de Negócio, Aplicação *Web*, Página de Exercício e Página de Resoluções. Neste trabalho, o banco de dados relaciona com a Lógica de Negócio, através dos comandos em SQL no Banco de Dados e do PHP na Lógica de Negócio. Assim, os dados consultados são gerenciados pelo *JavaScript* e o *AngularJS* na Lógica de Negócio. Além disso, a parte Lógica de Negócio estão todas as operações de resoluções dos 4 algoritmos de escalonamento de processos em códigos. Como também algumas operações relacionadas às inserções de exercícios dos algoritmos de escalonamentos ao Banco de dados, bem como a parte das variáveis que serão utilizadas nas Aplicação *Web*.

A próxima ligação está em Lógica de Negócio e aplicação *Web*. Esta seria o caso do *Front End*, onde as informações são apresentadas para o usuário do sistema. Na aplicação *Web*, todos os dados para serem apresentados já foram gerenciados na Lógica de Negócio, então sua

função é apenas apresentar essas informações, mas de forma estruturada e otimizada. Por isso, os usos das tecnologias de HTML, *Bootstrap*, *Web Storage* e *JavaScript* também na parte da aplicação *Web*. Aliás, houve casos neste trabalho que precisou de buscar dados na aplicação *Web* para armazenar no Banco de dados, um processo que inicia na aplicação, passa pela Lógica de Negócio e chega posteriormente ao Banco de dados.

A aplicação *Web* contém duas distribuições de páginas importantes neste trabalho, que seriam as Páginas de Exercícios e de Resolução. Nas Páginas de Exercícios são apresentados os exercícios criados ou as opções de criar exercícios, além da possibilidade de responder um exercício em questão. A Página de Resolução apresenta as respostas do sistema para os usuários, sendo tais informações processadas na Lógica de Negócio e repassadas para a aplicação *Web*, que a distribui para a Página de Resolução.

4 RESULTADOS E DISCUSSÃO

Esta seção apresenta os resultados obtidos ao longo do desenvolvimento do sistema *web* sobre os exercícios e as resoluções dos algoritmos de escalonamento de processos, bem como artefatos e documentos gerados com base no planejamento inicial.

4.1 Estudo do contexto do ambiente

Esse sistema foi baseado em resolver exercícios dos algoritmos de escalonamento de processos. Assim, um exercício na disciplina de SOs é formado por uma tabela inicial de processo, além de um algoritmo de escalonamento de processos, dentre um dos quatro apresentados nesse trabalho. Dessa forma, o sistema disponibiliza para o usuário listas de exercícios cadastrados ou as opções de criar novos exercícios.

De qualquer maneira, o sistema consegue corrigir os exercícios dos algoritmos de escalonamento, apresentando ao final as resoluções dos exercícios. Porém, foi necessário planejar a fase inicial do desenvolvimento desse sistema, onde foram definidos alguns pontos importantes, nos quais podem ser vistos a seguir:

- armazenamento de dados: o sistema possui uma conexão com banco de dados para armazenar e gerenciar as informações referentes aos exercícios de escalonamento de processos. Além disso, o sistema tanto consegue gerenciar os dados pela base de dados, utilizando o MySQL, como pela parte da Lógica de Negócio, utilizando o *SessionStorage*. O *SessionStorage* se refere ao um banco de dados temporário localizado na maioria dos navegadores web, permitindo que esses dados armazenados sejam consultados por uma aplicação web.
- acesso ao ambiente: inicialmente, qualquer usuário poderá acessar o sistema, pois o sistema não tem o sistema de autenticação do usuário;
- funções do sistema: o sistema foi desenvolvido para resolver as questões dos algoritmos de escalonamento, analisar as respostas dos usuários e apresentar esses resultados;
- criar exercícios: o sistema disponibiliza aos usuários a função de criar novos exercícios, e poder armazená-los no banco de dados;
- exercícios cadastrados: apresenta uma lista de exercícios cadastrados, conforme a seleção de um algoritmo de escalonamento de processos.

Diante desse planejamento inicial, o sistema *Web* apresenta para o usuário as opções de inserir novos exercícios ao sistema ou de consultar exercícios cadastrados no sistema. Assim,

conforme o usuário vai acessando o sistema, os dados referentes aos algoritmos e aos exercícios vão sendo armazenados no MySQL como também no *SessionStorage*.

4.2 Sistema web implementado

O usuário não precisa se identificar ao acessar o sistema, visto que o sistema foi proposto inicialmente para ter acesso livre para todos, sem a necessidade de armazenamento de informações dos usuários. Desta forma, sem a identificação do usuário, a página inicial segue um padrão como apresentado na Figura 17.

Figura 17 - Página Inicial do Ambiente *Web*

Como ilustra a Figura 17, o sistema apresenta as opções de “Exercícios Cadastrados” e “Criar Exercícios”. A opção “Exercícios Cadastrados” é uma página que contém uma lista de exercícios cadastrados, onde o usuário pode escolher um exercício, além de um algoritmo, dentre os quatro algoritmos de escalonamentos apresentados.

Na opção “Criar Exercícios”, é possível o usuário inserir novos exercícios e escolher um algoritmo de escalonamento de processos para ser avaliado. Ao criar um novo exercício no sistema, o usuário pode salvar o exercício no banco de dados ou optar por somente em avaliar. Neste último caso, de optar por somente em avaliar, o usuário não precisa cadastrar o exercício no banco de dados para poder realizar a avaliação.

A solução do exercício, este que é proposto pelo usuário, é calculada dinamicamente pelo sistema, basta o usuário inserir um novo exercício. Dessa maneira, as opções “Exercícios Cadastrados” e “Criar Exercícios”, direciona o usuário para a página de respostas. A Figura 18 apresenta a interface da página de “Exercícios Cadastrados” e, posteriormente, de “Criar Exercícios”.

Figura 18 - Página de Exercício – Cadastrados

A Figura 18 apresenta ao usuário as opções de selecionar um algoritmo de escalonamento por vez, como mostra no canto esquerdo, sendo eles: FIFO, SJF, Prioridades ou Round-Robin. Após a escolha do algoritmo de escalonamento, o sistema disponibiliza ao usuário uma lista de exercícios conforme o algoritmo escolhido. Com isto, o usuário pode iniciar o processo de responder o exercício em questão, selecionando um exercício por vez.

Além disso, o usuário tem duas opções na tabela fornecida pelo sistema. A primeira opção é de “Visualizar”, onde o usuário acessa uma seção contendo detalhes da tabela inicial do exercício da linha. A outra opção é de “Responder” que redireciona o usuário para a página de avaliação de respostas. Assim, a Figura 19 mostra a seção de “Visualizar” detalhes do exercício.

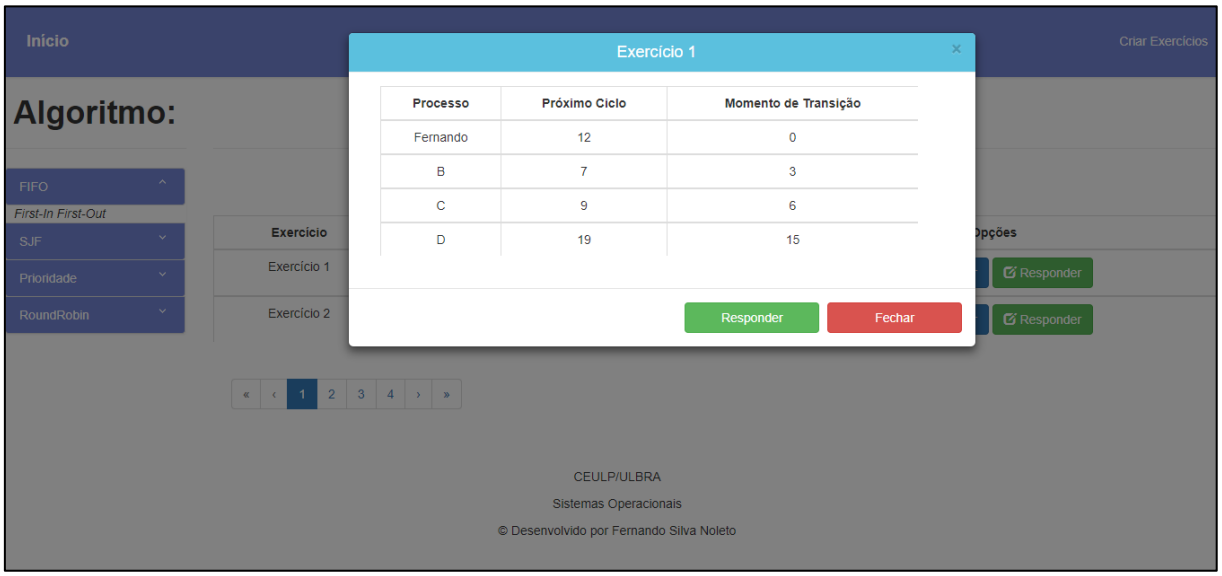


Figura 19 - Segunda Opção de Responder - Exercícios Cadastrados

Ao clicar na opção “Visualizar”, da Figura 18, o sistema apresenta o modelo da Figura 19 para o usuário. Nessa tela, o usuário visualiza a tabela inicial dos processos, com os nomes dos processos, os próximos ciclos e os momentos de transições de cada processo, em alguns casos, mostra as prioridades, quando usa o algoritmo de Prioridade. O usuário dessa forma, pode optar por responder o exercício ou voltar para página da Figura 18.

A outra opção apresentada na página inicial do sistema na Figura 17, “Criar Exercícios”, redireciona o usuário para a Figura 20. Nessa opção, é possível criar novos processos para compor uma lista de exercícios, formando ao final uma tabela inicial de processos.

Figura 20 - Página de Exercício – Criar Exercícios

A Figura 20 ilustra o que o sistema disponibiliza ao usuário quanto este escolhe a opção de criar novos exercícios. Na criação de um novo exercício, parte de o usuário escolher um algoritmo de escalonamento de processo dentre os quatro disponíveis, além de inserir os processos na tabela inicial. Conforme o usuário vai inserindo os processos, é possível visualizar as informações que foram informadas, sendo elas o nome do processo, o próximo ciclo e momento de transição, e em casos, a Prioridade. Os valores de próximo ciclo, momento de transição e de Prioridade são valores numéricos e assim só recebem valores inteiros.

A quantidade de processos que podem ser inseridos na lista de processos cadastrados é determinada pelo usuário. Ao final das inserções, o usuário pode optar por avaliar os seus processos inseridos e iniciar o processo de avaliação do exercício em questão ou optar por salvar a tabela inicial (o exercício) no banco de dados, para uma futura consulta.

A partir deste ponto, o usuário pode confirmar as suas entradas de dados dos processos e o sistema disponibilizará a página de resposta do exercício, conforme mostrado na Figura 21.

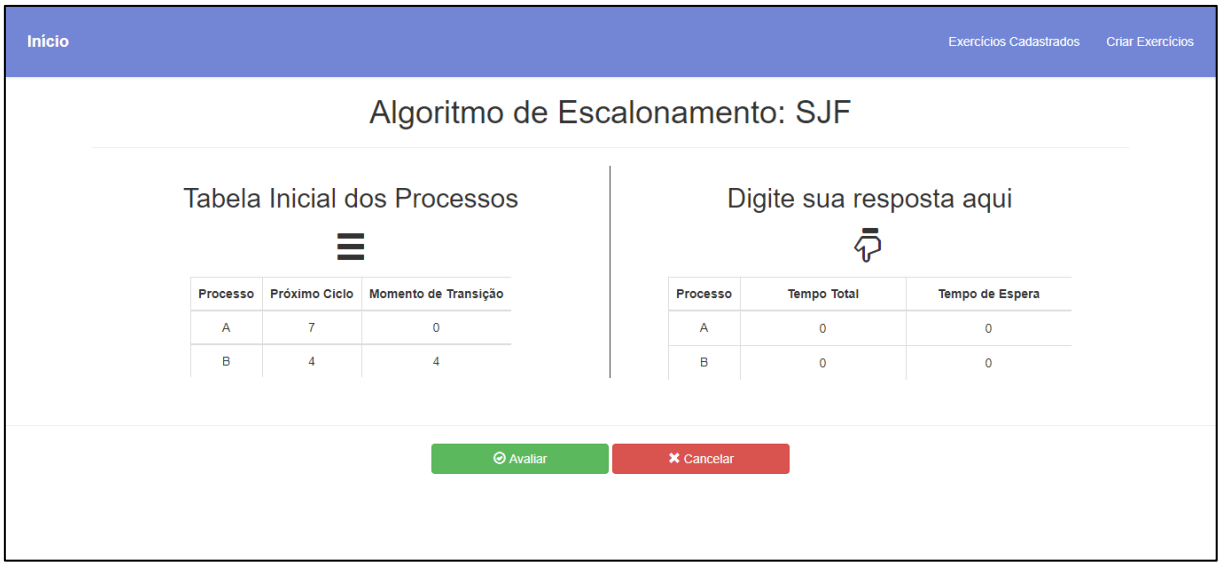


Figura 21 - Página de Análise do Algoritmo

A Figura 21 apresenta a página de análise do algoritmo de escalonamento de processos, que possui os dados do algoritmo de escalonamento, tabela inicial escolhida e a tabela de resposta. Os dados da tabela inicial são os mesmos escolhidos ou criados nas páginas de Exercícios Cadastrados ou de Criar Exercícios.

Como é mostrado na Figura 21, a tabela de resposta é onde o usuário poderá alterar os campos de Tempo Total e de Tempo de Espera. Assim, posteriormente, estes valores serem avaliados pelo sistema. As alterações da tabela de resposta do usuário serão baseadas na tabela inicial do algoritmo, tabela à esquerda, independentemente dessa tabela inicial ser originada da página de exercícios cadastrados ou de criar exercícios.

Quanto ao processo de alterações das respostas do usuário, o sistema neste projeto assegura um mecanismo em que torna os campos de Tempo Total e Espera editáveis bastando somente um clique na região do campo da tabela. Além disso, o sistema garante que os valores inseridos sejam numéricos, mesmo que o usuário tente inserir outros valores, sem ser números.

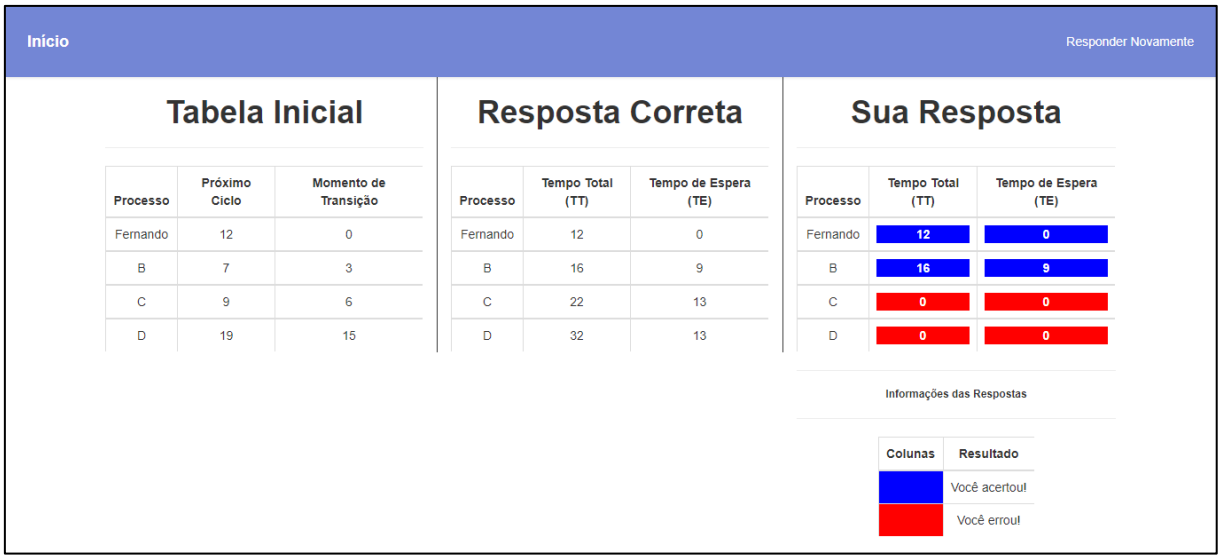


Figura 22 - Página de Resolução

A Figura 22 apresenta a página de resolução de exercício dos algoritmos de escalonamento de processos. Nessa página, o usuário pode visualizar a "Tabela Inicial" dos processos, a tabela de "Resposta Correta" e a tabela de "Sua Resposta". Na tabela de "Resposta Correta", o sistema apresenta os valores de tempo total e o tempo de espera corrigidos para cada processo contido na tabela inicial. Assim, basta então o usuário conferir as suas respostas, da tabela de "Sua Resposta", com os valores correspondentes da tabela de "Resposta Correta".

Além disso, outra informação que a tabela de "Sua Resposta" traz são as cores azul e vermelho. Essas cores estão presentes na tabela de "Informações das Respostas", localizada logo abaixo da tabela de "Sua Resposta" na Figura 22. As cores estão presentes na tabela de "Sua Resposta" para indicar se o usuário acertou ou errou algum valor (tempo total ou tempo médio) para cada processo. Dessa forma, a cor azul indica se o usuário acertou o tempo total ou o tempo médio de um processo. E a cor vermelha, indica se o usuário errou o tempo total ou tempo médio de um processo.

Nas próximas seções são apresentadas as partes de implementações desse sistema *Web*. Inicia com a parte do banco de dados e termina com as explicações dos algoritmos que resolvem os exercícios dos quatro algoritmos de escalonamento de processos.

4.3 Conexão com a Base de Dados

Nesta seção são apresentadas as duas fases de conexão com o banco de dados do MySQL, bem como o uso da linguagem PHP e da tecnologia do *SessionStorage*.

A Figura 23 apresenta algumas linhas de códigos que realizam consultas ao banco de dados, onde cada consulta retorna uma lista de exercícios. Essas consultas são realizadas através do uso do `$http` utilizando o método `post`, como é mostrado nas linhas 103, 110, 117 e 124.

```
102     $scope.updateDateExercise = function() {
103         $http.post('backend/consultas/listFIFO.php')
104         .success(function(data){
105             var exerciciosFIFO = [];
106             exerciciosFIFO = $scope.inserirExerciciosBDnoSession(data);
107             var arrayString = JSON.stringify(exerciciosFIFO);
108             sessionStorage.setItem("ArrayExerciseFIFO0001", arrayString);
109         });
110         $http.post('backend/consultas/listSJF.php')
111         .success(function(data){
112             var exerciciosSJF = [];
113             exerciciosSJF = $scope.inserirExerciciosBDnoSession(data);
114             var arrayString = JSON.stringify(exerciciosSJF);
115             sessionStorage.setItem("ArrayExerciseSJF0002", arrayString);
116         });
117         $http.post('backend/consultas/listPrioridade.php')
118         .success(function(data){
119             var exerciciosPrioridade = [];
120             exerciciosPrioridade = $scope.inserirExerciciosBDnoSession(data);
121             var arrayString = JSON.stringify(exerciciosPrioridade);
122             sessionStorage.setItem("ArrayExercisePrioridade0003", arrayString);
123         });
124         $http.post('backend/consultas/listRoundRobin.php')
125         .success(function(data){
126             var exerciciosRoundRobin = [];
127             exerciciosRoundRobin = $scope.inserirExerciciosBDnoSession(data);
128             var arrayString = JSON.stringify(exerciciosRoundRobin);
129             sessionStorage.setItem("ArrayExerciseRoundRobin0004", arrayString);
130         });
131         console.log("Consultando do DB do SQL:");
132     };
```

Figura 23 - Consultas aos algoritmos no banco de dados

A Figura 23 é apresentada as instruções de códigos referentes as consultas ao banco de dados que são realizadas nesse sistema. Cada consulta ao banco de dados, como mostram nas linhas de códigos 103 (consulta ao algoritmo FIFO), 110 (consulta ao algoritmo de SJF), 117 (consulta ao algoritmo Prioridade) e 124 (consulta ao algoritmo) faz referência a uma chamada a um arquivo específico escrito em PHP. O que pode ser visto em uma das consultas realizadas, como o `$http.post('backend/consultas/listFIFO.php')`.

O `$http` utiliza o método `post` para realizar consultas este `post` pode receber como um parâmetro de entrada um caminho da localização do arquivo. Nesse exemplo, a localização do arquivo para fazer a consulta referencia o algoritmo FIFO, porém, os outros arquivos que referenciam os algoritmos SJF, Prioridade e o Round-Robin.

Esses trechos de códigos da Figura 23 representam a comunicação da Lógica de Negócios com o Banco de dados. Assim, o que se espera do método `post` é um retorno de

algum valor do banco de dados, nesse caso, uma lista de exercícios, para uma variável da Lógica de Negócio. Como a Figura 23 só apresenta a parte Lógica de Negócio, a parte do Banco de Dados será apresentada nas próximas figuras. O procedimento realizado no Banco de Dados apresenta-se em duas fases: a primeira é a parte de inserções e consultas, ambas utilizadas neste trabalho, e a segunda fase seria a comunicação com a base de dados, o banco MySQL.

```php
<?php

include "../Database.php";

$query = "SELECT * FROM exercicios WHERE algoritmo = 'FIFO'";
$rs = $dbbanco->query($query);

while($row=$rs->fetch_assoc()){
    $data[] = $row;

}

print json_encode($data);

?>

```

Figura 24 - Fase um do Banco de Dados

Como foi apresentado na Figura 23, a parte do Banco de Dados apresenta duas fases, a primeira dela são as linhas de códigos apresentadas na Figura 24. Nessa fase, é feito o procedimento de consulta utilizando o comando `SELECT` da linguagem SQL, como é apresentado na linha na linha 5. Mas, antes de realizar esse procedimento, precisa ter uma ligação entre essa primeira fase com a outra fase, o que seria a comunicação com a base de dados. Essa ligação com a segunda fase inicia na linha de código 3, que referencia o arquivo `Database.php`, assim, esses trechos de códigos da Figura 24 estabelecem uma conexão com a segunda fase.

O próximo passo é declarar uma variável em PHP para receber o retorno dessa consulta em SQL, variável então criada que se chama `$query`. Esta variável espera receber de retorno através da instrução `SELECT * FROM exercícios WHERE algoritmo = 'FIFO'` uma lista de exercícios do algoritmo FIFO, da tabela de exercícios no banco de dados. Aos outros algoritmos de escalonamento de processos, foram realizados os mesmos procedimentos, só alterando o critério de seleção pelo nome do algoritmo. O final dessa instrução é um retorno de dados para a parte Lógica de Negócio.

A próxima etapa consistiu na espera do retorno da base de dados. Caso a consulta seja realizada com sucesso, os dados oriundos da base são passados para as variáveis criadas na Figura 24, e posteriormente são encaminhadas para a Lógica de Negócio, onde as variáveis estão aguardando os dados. A próxima figura apresenta a segunda fase da conexão e comunicação com o banco de dados.

```php
<?php
define("HOSTNAME","localhost");
define("USERNAME","root");
define("PASSWORD","");
define("DATABASE","algoritmo_escalonamento");

$dbbanco = new mysqli(HOSTNAME,USERNAME,PASSWORD,DATABASE) or die("Unable to Connect DB");
?>
```

Figura 25 - Fase dois do Banco de Dados

A Figura 25 ilustra os trechos de códigos em PHP que realizam o processo de comunicação com a base de dados do MySQL. Para isto, foi necessário criar uma variável chamada de `$dbbanco`, variável responsável pelos procedimentos de comunicações e conexões com a base de dados. Após realizada essa conexão, a `$dbbanco` passa a ser o meio de comunicação deste sistema com o banco de dados do `algoritmo_escalonamento`, da base de dados do MySQL.

A variável `$dbbanco` passa a ser utilizada para realizar qualquer procedimento junto ao banco de dados deste sistema, seja consulta ou inserção de informações. É o caso visto na Figura 24, onde a variável é utilizada para se comunicar com o banco de dados.

```javascript
$http.post('backend/consultas/listSJF.php')
.success(function(data){
    var exerciciosSJF = [];
    exerciciosSJF = $scope.inserirExerciciosBDnoSession(data);
    var arrayString = JSON.stringify(exerciciosSJF);
    sessionStorage.setItem("ArrayExerciseSJF0002", arrayString);
});
```

Figura 26 - Exemplo de armazenados de dados nas variáveis da Lógica de Negócio

A Figura 26 ilustra as variáveis da Lógica de Negócio que recebem os dados do banco de dados, instrução apresentada na Figura 24. Na Figura 26 é possível observar as variáveis `exerciciosSJF` e `arrayString`. A `exerciciosSJF` recebe os dados que vem do banco de dados e armazena os dados em uma variável do *JavaScript*. A `arrayString`, por outro lado, recebe os dados do `exerciciosSJF` e passa esses mesmos dados para as variáveis de

SessionStorage. Inclusive, o dado armazenado no *SessionStorage* na Figura 26 é chamado de `ArrayExerciseSJFF0002`, que se refere ao exercícios do algoritmo FIFO.

Esta variável do FIFO e como outras que representam os restantes dos algoritmos, são armazenadas no *SessionStorage*. Os detalhes dos valores armazenados no *SessionStorage* neste trabalho, podem serem vistos na Figura 27.

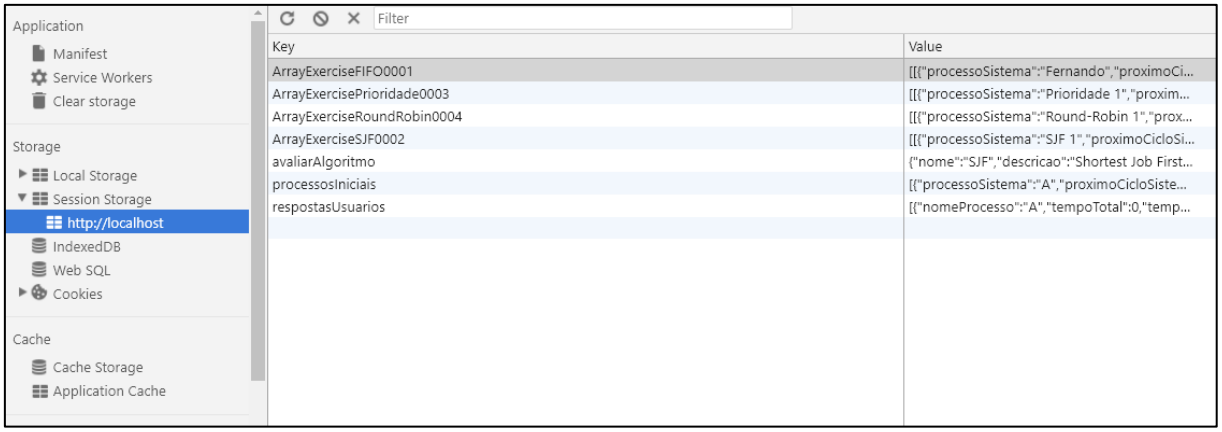


Figura 27 - Exemplo de variáveis no *SessionStorage*

Os dados armazenados no *SessionStorage* podem ser vistos através do *LocalStorage* na maioria dos navegadores da internet. A Figura 27 apresenta os dados utilizados neste trabalho que foram armazenados no *SessionStorage*. Isso apresenta que somente não foram os quatro exercícios de algoritmos de escalonamento que foram armazenados. Outros dados também foram armazenados, como respostas de usuários, o algoritmo escolhido pelo usuário para avaliação e dentre outros que são explicados na Figura 28.

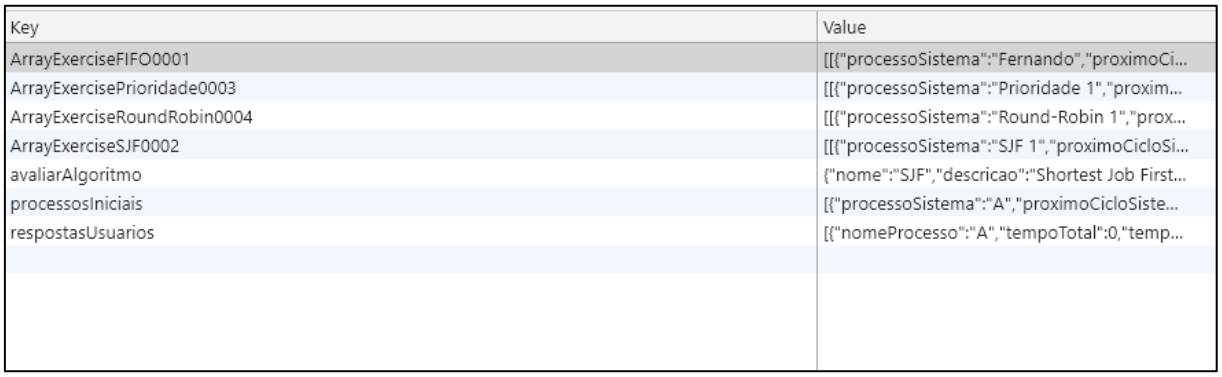


Figura 28 - Variáveis do *SessionStorage*

A Figura 28 apresenta os dados dos exercícios dos algoritmos e outros dados que foram citados na Figura 27, todos armazenados no *SessionStorage.* Cada um desses dados da Figura 28, representa um significado importante para o funcionamento desse sistema. Dessa forma, a seguir são detalhadas cada um dos dados armazenados:

- `avaliarAlgoritmo`: dados do algoritmo a ser avaliado pelo usuário, tipo nome e descrição;
- `processosIniciais`: uma lista de exercício que o usuário escolheu na página de Exercício, representadas nas Figuras 18 e 19 deste trabalho;
- `respostasUsuários`: uma lista que coleta as respostas dos usuários, dados oriundos do usuário;
- `ArrayExerciseFIFO0001`: uma lista de exercícios do algoritmo FIFO, dados oriundos da consulta ao banco de dados;
- `ArrayExerciseSJF0002`: uma lista de exercícios do algoritmo SJF, dados oriundos da consulta ao banco de dados;
- `ArrayExercisePrioridade0002`: uma lista de exercícios do algoritmo Prioridade, dados oriundos da consulta ao banco de dados;
- `ArrayExerciseRoundRobin0002`: uma lista de exercícios do algoritmo Round-Robin, dados oriundos da consulta ao banco de dados.

Assim, o *SessionStorage* neste trabalho serviu como um tipo de banco de dados temporário no navegador do usuário, quando este acessa o sistema. Com isso, várias operações de atualizações e alterações de dados foram sendo feitas através dos dados armazenados no *SessionStorage.*

4.4 Implementação dos algoritmos para resolver os exercícios de escalonamentos de processos

Nesta seção é apresentada a implementação dos algoritmos que foram utilizados para resolver os exercícios de escalonamentos de processos, estes que se referem ao FIFO, SJF, Prioridade e Round-Robin. Para cada algoritmo desenvolvido nessa seção, foi utilizado uma lógica voltada para encontrar o momento final dos processos.

Dessa forma, após identificar o momento final do processo, foi possível encontrar o tempo total e o tempo médio dos processos contido em uma tabela inicial. Diante disso, nas próximas seções são apresentadas as implementações dos algoritmos que foram utilizados para resolver os exercícios dos algoritmos de escalonamento de processos.

4.4.1 FIFO

A implementação do algoritmo FIFO foi baseada na estrutura de uma fila simples, onde o primeiro que chega, é o primeiro que executa na CPU. Assim, a sua implementação ficou conforme apresentado na figura 29.

```
73  $scope.avaliarFIFO = function($algoritmoFIFO) {
74      var algoritmoAvaliado = [];
75      var linhaTempo = 0;
76      var tempoOcioso = 0;
77      console.log($algoritmoFIFO);
78          for(var i=0; i < $algoritmoFIFO.length; i++) {
79              if(i != 0) {
80                  tempoOcioso = $algoritmoFIFO[i].momentoTransicaoResolucao - linhaTempo;
81                  console.log(tempoOcioso);
82              }
83              linhaTempo += $algoritmoFIFO[i].proximoCicloResolucao;
84              var momentoInicial = 0;
85              var momentoFinal = 0;
86              var tempoTotal = 0;
87              var tempoEspera = 0;
88              momentoInicial = $algoritmoFIFO[i].momentoTransicaoResolucao;
89              momentoFinal = linhaTempo;
90              if(tempoOcioso > 0) {
91                  momentoFinal += tempoOcioso;
92                  linhaTempo += tempoOcioso;
93              }
94              tempoTotal = momentoFinal - momentoInicial;
95              tempoEspera = tempoTotal - $algoritmoFIFO[i].proximoCicloResolucao;
96              algoritmoAvaliado.push({Processo : $algoritmoFIFO[i].processoResolucao, TempoEspera : tempoEspera, TempoTotal : tempoTotal});
97          }
98      return algoritmoAvaliado;
99  };
100
```

Figura 29 - Implementação do algoritmo FIFO

A Figura 29 apresenta o algoritmo FIFO bem como a lógica abordada para seu desenvolvimento. A princípio, a implementação seguiu um padrão da quantidade de processos, estes contidos na tabela inicial, como o limite do laço de repetição, como mostram as linhas de códigos 78 até 99. Assim, com este laço, foi possível identificar, para cada processo, o tempo total e o tempo de espera, além da chance da ocorrência de algum tempo ocioso[7].

Logo no início do laço de repetição, começa com uma condição, linha 79, para verificar se no momento da entrada de um processo na linha do tempo da CPU, se existia alguma ocorrência de tempo ocioso. Esta verificação de tempo ocioso, vale para todas as entradas dos processos, exceto para o primeiro processo da tabela inicial. O tempo ocioso em Sistemas Operacionais, significa que a CPU não tinha nenhum processo executando no momento. Este valor é importante, pois pode influenciar nos resultados finais no tempo total e o tempo médio.

Após essa verificação, são criadas algumas variáveis temporárias para receberem os dados de momentos iniciais e finais, e tempos totais e médios. A partir disso, logo na linha 87 já é coletado o momento inicial do processo. Depois, é coletado o momento final, linha 88, a partir da variável de `linhaTempo`, que armazena as informações na linha 83. Após isso, é feita

[7] Tempo ocioso é o tempo em que a CPU fica sem nenhum processo executando na linha do tempo (BITTERNCOURT, 2014).

uma verificação para saber se o tempo ocioso ocorreu, nas linhas 90 até 93. Caso tenha ocorrido, os valores da linha do tempo e o momento final são alterados.

Depois desses procedimentos, são realizados os cálculos de tempo total e de espera, nas linhas 94 e 95. Logo após, são passadas essas informações para uma variável, linha 96, que será utilizada na página do usuário e que conterá o nome do processo, o tempo de espera e o tempo total. Assim, segue normalmente para os restantes dos processos, até chegar no último processo da tabela inicial, encerrando este algoritmo.

O algoritmo FIFO utiliza uma estrutura de lista, que contém os processos da tabela inicial. Esta lista deve executar os primeiros processos na CPU, aqueles que estão nas primeiras posições da tabela inicial. Isto em um contexto de execução na CPU, seria dizer que os primeiros processos que chegam, são os primeiros processos a serem executados.

Por estas características, o algoritmo FIFO não interfere na ordem de chegadas dos processos na tabela inicial e não utiliza dos atributos de Próximo Ciclo dos processos como critério de ordenação dos processos, mas sim, do Momento de Transição, ao contrário do que acontece com os demais algoritmos de escalonamento de processos.

4.4.2 SJF

A implementação do SJF segue a mesma estrutura utilizada no algoritmo FIFO, mas, em SJF é feita uma ordenação na fila de aptos para executar os processos com menor quantidade de tempo de execução na CPU. Esta quantidade de tempo é representada na tabela inicial, para cada processo, como o Próximo Ciclo. Com bases nas informações dos Próximos Ciclos dos processos, é possível ordenar os processos, de acordo com a sua ordem de chegada (Momento de Transição).

```
103     $scope.ordenarSJF = function($filaAptos, $posicao) {
104         var processoAtual = $posicao + 1;
105         for(var i = processoAtual; i < $filaAptos.length; i++) {
106             for(var j = processoAtual; j <= $filaAptos.length-2; j++) {
107                 if($filaAptos[j].proximoCicloSJF > $filaAptos[j+1].proximoCicloSJF) {
108                     var valorTemporario = $filaAptos[j];
109                     $filaAptos[j] = $filaAptos[j+1];
110                     $filaAptos[j+1] = valorTemporario;
111                 }
112             }
113         }
114     };
```

Figura 30 - Ordenação do algoritmo SJF

A Figura 30 ilustra a ordenação do algoritmo do SJF, função que é chamada em tempo de execução no Algoritmo SJF (Figura 32). Na ordenação dos processos, há exceção para o primeiro processo da tabela inicial ou o processo atual que está executando na linha do tempo da CPU, que deve ser executado primeiro na CPU. Dessa maneira, são apresentadas nas próximas figuras, o algoritmo de SJF.

```
116  $scope.calcularSJF = function($listaAlterada, $algoritmoInicialSJF) {
117      var avaliadoSJF = [];
118      for(var i = 0; i < $algoritmoInicialSJF.length; i++) {
119          for(var j = 0; j < $listaAlterada.length; j++) {
120              if($algoritmoInicialSJF[i].processoResolucao == $listaAlterada[j].nmProc) {
121                  var temTotal = $listaAlterada[j].MFproc - $algoritmoInicialSJF[i].momentoTransicaoResolucao;
122                  var temEspera = temTotal - $algoritmoInicialSJF[i].proximoCicloResolucao;
123                  avaliadoSJF.push({Processo : $algoritmoInicialSJF[i].processoResolucao, TempoEspera : temEspera, TempoTotal : temTotal});
124                  break;
125              }
126          }
127      }
128      return avaliadoSJF;
129  };
```

Figura 31 - Calculo de ajuste do algoritmo SJF

A Figura 31 ilustra uma função que realiza os ajustes do algoritmo SJF e que é chamada no momento final do Algoritmo do SJF (Figura 32). A função da Figura 31 é chamada de `calcularSJF`, e ela recebe de entrada duas listas, sendo uma lista de processos alterada no algoritmo SJF e uma lista de processos da tabela inicial.

De posse dessas duas listas, a função realiza o cálculo do tempo total e o tempo médio de cada processo da tabela inicial, através das linhas de códigos 121 e 122. Para conferir se o nome do processo da lista alterada segue-se o mesmo processo da tabela inicial, através das comparações dos nomes, conforme linha de código 120 (Figura 31).

Como o Algoritmo do SJF é muito extenso, ele será explicado em duas partes, conforme as próximas figuras.

```
131     $scope.algoritmoSJF = function($algoritmoSJF) {
132         var linhadoTempo = 0;
133         var filaAptos = [];
134         var processosEmEspera = [];
135         var avaliarAlgoritmo = [];
136         var algoritmoFinalizado = [];
137         var encontrarNovoProcesso = true;
138         var encerrarAlgoritmo = false;
139         for(var i = 0; i < $algoritmoSJF.length; i++) {
140             linhadoTempo += $algoritmoSJF[i].proximoCicloResolucao;
141             linhadoTempo += $algoritmoSJF[i].momentoTransicaoResolucao;
142             processosEmEspera.push(
143                 {processoSJF: $algoritmoSJF[i].processoResolucao, proximoCicloSJF: $algoritmoSJF[i].proximoCicloResolucao,
144                     momentoTransicaoSJF: $algoritmoSJF[i].momentoTransicaoResolucao}
145             );
146         }
147         filaAptos.push(processosEmEspera[0]);
148         for(var linha = 1; linha <= linhadoTempo; linha++) {
149             for(var i = 0; i < filaAptos.length; i++) {
150                 if(filaAptos[i].proximoCicloSJF > 0) {
151                     filaAptos[i].proximoCicloSJF = filaAptos[i].proximoCicloSJF - 1;
152                         if(filaAptos[i].proximoCicloSJF == 0) {
153                             avaliarAlgoritmo.push(
154                                 {nmProc: filaAptos[i].processoSJF, MIproc: filaAptos[i].momentoTransicaoSJF,
155                                     MFproc: linha}
156                             );
157                             if(i == processosEmEspera.length - 1) {
158                                 encerrarAlgoritmo = true;
159                                 break;
160                             }
161                         }
```

Figura 32 - Algoritmo do SJF – Parte 1

A Figura 32 ilustra a Parte 1 do algoritmo de SJF que apresenta a função denominada de `algoritmoSJF`, que irá se estender na parte 2 do algoritmo, referente a Figura 33. A função `algoritmoSJF` recebe como entrada uma lista de processos da tabela inicial, referente ao exercício escolhido pelo usuário. Logo no início da função, são declaradas algumas variáveis para controlar e gerenciar os dados da lista no algoritmo SJF.

Após as declarações das variáveis, a função `algoritmoSJF` inicia um laço de repetição para gerenciar a linha do tempo da CPU e os processos (linha de código 148). Em seguida, é iniciado outro laço de repetição para representar os processos na fila de aptos (linha de código 148). Com estes dois laços de repetições, foi possível percorrer todos os processos da tabela inicial e encontrar os momentos finais dos processos.

Após iniciar os dois laços de repetições, é feita uma condição para verificar se o próximo ciclo do processo atual é maior que zero (linha de código). Caso seja maior que zero, é subtraído menos um do próximo ciclo do processo. Depois disto, é feita outra condição para verificar se o próximo ciclo é igual a zero (linha de código 150). Caso seja igual a zero, o momento final do processo atual é encontrado, ele é encerrado na linha do tempo da CPU.

Com isto, o processo atual com o seu próximo ciclo igual zero é passado para outra variável, chamada `avaliarAlgoritmo`. E é nesta variável que são feitos os ajustes e os cálculos dos tempos, ilustrados na Figura 31. Ainda dentro da condição de próximo ciclo igual zero, é feita outra verificação para saber se o processo atual é o último da fila de aptos e da

tabela inicial que foi avaliado (linha de código 157). Caso o processo atual seja o último, é atribuída uma variável com valor verdadeiro, variável que é utilizada na Figura 33, para representar a parada dos dois laços de repetições. Caso não seja o último processo, o algoritmo segue normalmente, para o próximo processo da fila de aptos.

Após passar pelas condições de próximo ciclo maior ou igual a zero, o resto do algoritmo é apresentado na Figura 33, que segue logo abaixo.

```
162                         for(var j = 1; j < processosEmEspera.length; j++) {
163                             if(processosEmEspera[j].momentoTransicaoSJF == linha) {
164                                 filaAptos.push(processosEmEspera[j]);
165                                 $scope.ordenarSJF(filaAptos, i);
166                                 break;
167                             }
168                         }
169                     encontrarNovoProcesso = false;
170                     break;
171                 }
172             }
173             if(encerrarAlgoritmo) {
174                 break;
175             }
176             if(encontrarNovoProcesso) {
177                 for(var j = 1; j < processosEmEspera.length; j++) {
178                     if(processosEmEspera[j].momentoTransicaoSJF == linha) {
179                         filaAptos.push(processosEmEspera[i]);
180                         break;
181                     }
182                 }
183             } else {
184                 encontrarNovoProcesso = true;
185             }
186         }
187       algoritmoFinalizado = $scope.calcularSJF(avaliarAlgoritmo, $algoritmoSJF);
188       return algoritmoFinalizado;
189     };
```

Figura 33 - Algoritmo do SJF – Parte 2

A Figura 33 é a continuação dos dois laços de repetições, e antes de terminar o segundo laço que representa os processos na lista de aptos, é iniciado um outro laço de repetição para identificar se existe algum processo que iniciou no momento atual da linha do tempo, esta definida no primeiro laço de repetição (linha de código 162).

Caso algum processo tenha iniciado no momento atual da linha do tempo (linha de código 163), é feita a inserção do processo novo na fila de aptos, e logo em seguida, é chamada a função `ordenarSJF`, da Figura 30. Após essa verificação, caso tenha encontrado algum processo ou não, é feita atribuições algumas variáveis, antes de terminar o segundo laço de repetição.

Após terminar o segundo laço de repetição referente a fila de aptos, é feita uma verificação para saber se foi atribuído um valor verdadeiro (linha de código 173 – Figura 33) à variável `encerrarAlgoritmo`. Caso a variável tenho o valor verdadeiro, o primeiro laço de

repetição é encerrado e então finalizado o processo de encontrar os momentos finais dos processos.

Caso a variável `encerrarAlgoritmo` não tenha o valor verdadeiro, segue normalmente a execução do primeiro laço de repetição, através da condição na linha 176, da Figura 33. A condição da linha 176 só é verdadeira quanto nenhum processo da fila de aptos tenha o valor do próximo ciclo maior que zero. Assim, é feito mais um laço de repetição, para encontrar se algum processo iniciou no momento atual da linha do tempo da CPU. Este laço da linha 176 é o mesmo laço de repetição apresentado na linha de código 162, da Figura 33.

Desta forma, caso tenha encontrado algum processo, ele é inserido na fila de aptos. E diferente do laço da linha de código 162, no laço da linha 176 não é feita a chamada da função `ordenarSJF`.

Por fim, quando o algoritmo de SJF terminar os dois laços de repetições, é realizada a chamada para função `calcularSJF`, para ajustar e calcular os tempos totais e médios dos processos. E a função `calcularSJF` passa como parâmetro duas listas, a primeira é a lista de processos alterados nas Figuras 32 e 33, e a segunda lista é dos processos da tabela inicial.

O problema principal no desenvolvimento deste algoritmo estava em como reaproveitar a mesma estrutura da lista e de cálculo (cálculo de tempo total e médio) do algoritmo FIFO. Foi necessário realizar algumas modificações quanto à ordem de posições (ou chegadas) dos processos na fila de aptos, antes de serem executados na CPU, além de utilizar como critério de ordenação, o atributo de Próximo Ciclo dos processos.

4.4.3 Prioridade

A implementação do algoritmo de Prioridade segue uma ordenação na fila de aptos de acordo com a prioridade dos processos. Essa prioridade é classificada do maior para o menor. Nos processos com mesma prioridade, a ordenação segue conforme a ordem de chegada (Momento de Transição).

```
168     $scope.ordenarPrioridade = function($algoritmoPri) { //algoritmo de prioridade
169         var filaAptos = [];
170         var filaAptos = $algoritmoPri;
171         var valorAutal;
172 
173         for(var j=0; j<filaAptos.length; j++) {
174             for(var i=0; i <= filaAptos.length - 2; i++) {
175                 if(filaAptos[i].prioridadeResolucao > filaAptos[i + 1].prioridadeResolucao)
176                     {
177                         valorAutal = filaAptos[i];
178                         filaAptos[i] = filaAptos[i + 1];
179                         filaAptos[i + 1] = valorAutal;
180 
181                     }
182                     else
183                     {
184                         valorAutal = filaAptos[i];
185                         filaAptos[i] = valorAutal;
186                     }
187             }
188         }
189     };
```

Figura 34 - Ordenação dos processos no algoritmo de Prioridade

A Figura 35 apresenta a função `ordenarPrioridade` que realiza uma ordenação dos processos contido em uma lista, conforme a prioridade dos processos. Esta função é chamada a partir do algoritmo de Prioridade (algoritmo que será apresentado mais adiante) que passa como parâmetro uma lista de processos. Nesse algoritmo, o parâmetro utilizado foi a fila de aptos (linha de código 169), e é nessa fila de aptos que são feitas as ordenações dos processos.

A variável `filaAptos` contém processos que estão em execução na linha do tempo no algoritmo de Prioridade, ou seja, em tempo de execução na CPU. No momento que entra um processo na `filaAptos`, é necessário chamar a função da Figura 35 para ordenar os processos contidos até então na fila de aptos. Assim, a função realiza, se for o caso, várias ordenações de posições dos processos para serem executados na CPU, de forma, a manter os processos com maiores prioridades nas primeiras posições para execuções. Ao final da ordenação, a `filaAptos` é retornada para o algoritmo de Prioridade.

```
191     $scope.organizarPrioridade = function($algoritmoPrioridade, $algoritmoSistema) {
192         var avaliadoPrioridade = [];
193         for(var i=0; i<$algoritmoSistema.length; i++) {
194             for(var j=0; j<$algoritmoPrioridade.length; j++) {
195                 if($algoritmoSistema[i].processoResolucao == $algoritmoPrioridade[j].nmProc) {
196                     var temTotal = 0;
197                     var temEspera = 0;
198                     temTotal = $algoritmoPrioridade[j].MFproc - $algoritmoSistema[i].momentoTransicaoResolucao;
199                     temEspera = temTotal - $algoritmoSistema[i].proximoCicloResolucao;
200                     avaliadoPrioridade.push({Processo : $algoritmoSistema[i].processoResolucao, TempoEspera : temEspera, TempoTotal : temTotal});
201                 }
202             }
203         }
204     avaliadoPrioridade.sort($scope.ordenarABC);
205     return avaliadoPrioridade;
206     };
```

Figura 35 - Organização dos Processos em Prioridade

A Figura 36 apresenta a função `organizarPrioridade` que tem como objetivo realizar os cálculos dos tempos totais e os tempos médios. Esta função recebe como parâmetros, duas entradas possíveis. A primeira entrada é o `$algoritmoPrioridade` que representa a lista de processos alterados pelo algoritmo de Prioridade. E a segunda entrada o `$algoritmoSistema` que representa a lista da tabela inicial dos processos.

Assim, a função inicia com dois laços de repetições, sendo o primeiro laço com critério de parada, a quantidade de processos contidos no `$algoritmoSistema`. O segundo laço de repetição tem como critério de parada a quantidade de processos contidos na variável `$algoritmoPrioridade`.

Após iniciados os laços de repetições, começa com uma condição para verificar se o nome do processo atual da variável `$algoritmoSistema` é o mesmo nome da variável `$algoritmoPrioridade`. Esta condição é necessária para identificar se no momento atual, o processo da variável `$algoritmoSistema` é o mesmo que da variável `$algoritmoPrioridade`. Casos seja, são criadas variáveis temporárias para receberem os valores de tempos totais e tempos médios.

Dessa maneira, a variável temporária do tempo total (linha 198) recebe como entrada o cálculo do momento final do processo atual na variável `$algoritmoPrioridade` menos o momento inicial da variável `$algoritmoSistema`. Logo após (linha 199) a variável do tempo de espera recebe como entrada, o cálculo do valor de tempo total menos o próximo ciclo do processo atual no `$algoritmoSistema`. Feitas essas operações, os valores dos tempos totais e médios, mais o nome do processo, são inseridos em uma variável na linha 200. Variável que retorna à função do algoritmo de Prioridade, após terminarem os laços de repetições.

Como o algoritmo de Prioridade apresenta uma grande quantidade de linhas de códigos, o código será dividido em partes para melhores explicações sobre o funcionamento do algoritmo.

```
208    $scope.algoritmoPrioridades = function($processoPrioridade) { //algoritmo de prioridade
209        var arrayProcess = [];
210        var filaAptos = [];
211        var linhadoTempoTotal = 0;
212        var arrayPrioridade = [];
213        var prioridadeAvaliado = [];
214        for(var i=0; i<$processoPrioridade.length; i++)
215        {
216            arrayProcess.push({processoResolucao: $processoPrioridade[i].processoResolucao,
217                proximoCicloResolucao: $processoPrioridade[i].proximoCicloResolucao,
218                momentoTransicaoResolucao: $processoPrioridade[i].momentoTransicaoResolucao,
219                prioridadeResolucao: $processoPrioridade[i].prioridadeResolucao});
220        }
221
222        for(var i=0; i<arrayProcess.length; i++) {
223            linhadoTempoTotal += arrayProcess[i].proximoCicloResolucao;
224            linhadoTempoTotal += arrayProcess[i].momentoTransicaoResolucao;
225        }
226
227        var contadorParada = false;
228        var processsoAdicionado = false;
229        var tamanhoArray = $processoPrioridade.length;
230        var contadorTempoOcioso = 0;
231        var ultimoProcesso = false;
232        var tempoOcioso = 0;
233        var PCAtualMaiorZero = false;
234        var processoAtualZero = false;
235        var validoQuanto = "Nao";
```

Figura 36 - Algoritmo de Prioridade – Parte 1

A primeira parte do Algoritmo de Prioridade é ilustrada na Figura 37, começando na linha 208. Essa parte do algoritmo, é composta pela entrada da tabela inicial da lista de processos do algoritmo de Prioridade, além das declarações das variáveis utilizadas nos restantes dos códigos.

Logo após dessas declarações, é iniciado o primeiro laço de repetição para inserções dos processos da tabela para a variável `arrayProcess`, que sofrerá alterações ao longo do algoritmo, enquanto a variável de entrada, permanecerá sem alterações. Após terminar o primeiro laço de repetição, começa outro, nas linhas 222 até 225, para somar a linha do tempo que será utilizada em outro laço de repetição.

```
236
237     filaAptos.push(arrayProcess[0]);
238         for(var linha=1; linha<linhadoTempoTotal+1; linha++) {
239             for(var j=0; j <= filaAptos.length; j++) {
240                 if(j < tamanhoArray) {
241                     processoAtualZero = false;
242                     if(filaAptos[j].proximoCicloResolucao > 0) {
243                         PCAtualMaiorZero = true;
244                         filaAptos[j].proximoCicloResolucao = filaAptos[j].proximoCicloResolucao - 1;
245                         if(filaAptos[j].proximoCicloResolucao == 0) {
246                             var tempoFinalReal = linha;
247                             arrayPrioridade.push({nmProc: filaAptos[j].processoResolucao, MIproc: filaAptos[j].momentoTransicaoResolucao,
248                                 MFproc: tempoFinalReal});
249                          }
250                     } else {
251                         PCAtualMaiorZero = false;
252                         if(j == filaAptos.length - 1) {
253                             ultimoProcesso = true;
254 
255                             var contultimoProcesso = j;
256                             contultimoProcesso++;
257                                 if(contultimoProcesso == tamanhoArray) {
258                                     contadorParada = true;
259                                     break;
260                                 }
261                         }
262                         processoAtualZero = true;
263                     }
264
```

Figura 37 - Algoritmo de Prioridade - Parte 2

Na Figura 37 são realizadas as declarações das variáveis, enquanto a Figura 38 ilustra as linhas códigos dos dois laços de repetições, além dos momentos de verificações para os lações de repetições, como as condições para verificar as prioridades dos processos.

Assim sendo, logo no início da Figura 38 é utilizada a variável de `filaAptos` (Figura 37) para inserir como primeiro valor na fila, o primeiro processo da tabela inicial (linha 237). Após essa inserção, é iniciada um laço de repetição (linha 238) que tem a condição de parada o limite da linha do tempo.

Após o primeiro laço, é iniciado outro laço de repetição (linha 239) que tem com critério de parada a quantidade de processos contidos na `filaAptos`. Ao decorrer do código do Algoritmo de Prioridade, outros processos da tabela inicial são inseridos na `filaAptos`, além do primeiro já inserido.

Aos iniciarem os laços de repetições, a primeira condição (240) verifica se todos os processos já foram executados na linha do tempo. Caso haja processos para serem executados, passa pela condição e são feitos os cálculos dos seus momentos finais e os tempos. Caso todos os processos já tenham sido executados, encerram-se os dois laços de repetição.

Ao entrar na condição da linha 240, é necessário adicionar algumas variáveis com o valor de verdadeiro, para poder calcular os momentos finais de forma correta. Enfim, após passar por isso é feita uma verificação para saber se o próximo ciclo do processo atual é maior que zero (linha 242).

Logo depois, é subtraído o próximo ciclo do processo atual (linha 244). Feita a subtração, é feita outra verificação para saber o próximo ciclo atual é igual (linha 245). Caso seja igual a zero, o processo então é encerrado, mas antes, é coletado o momento final do

processo encerrado, com base no valor da linha do tempo, que é o atributo do primeiro laço de repetição (Figura 38), que representa uma linha do tempo na CPU. Assim são adicionados para outra variável (linhas 247 e 248) os processos que encerram os próximos ciclos na linha do tempo da CPU.

Caso o processo atual não tenha o próximo ciclo maior que zero, significa que o processo encerrou na linha do tempo em outro momento (linha 242). Desta forma, é realizada outra condição para verificar se o processo atual é o último processo da fila de aptos, condição de parada do segundo laço de repetição.

Caso seja o último processo (linha 252), é declarada uma variável como verdadeira, para ser utilizada em outro momento deste código. Após isso, é feita mais uma comparação para saber se o processo atual é o último da fila de aptos e o último dos processos contidos na tabela inicial (linha 257). Caso seja, é declarada outra variável para encerrar o algoritmo de Prioridade, no momento que essa variável passar por uma condição mais adiante. Porém, se por acaso condição da linha 252 seja falso, a execução do algoritmo segue normalmente.

Essa parte do Algoritmo de Prioridade representa as condições para verificar se o processo tem algum valor no seu Próximo Ciclo ou não, além das condições de encerramentos dos dois laços de repetições, através de várias variáveis declaradas. Após isto, a próxima etapa representa a identificação de novos processos na tabela inicial, além das condições de paradas dos laços de repetição.

```
265                 if((PCAtualMaiorZero) || (ultimoProcesso)) {
266                     ultimoProcesso = false;
267                     PCAtualMaiorZero = false;
268                     for(var k = 0; k<arrayProcess.length; k++) {
269                         if(arrayProcess[k].momentoTransicaoResolucao == linha) {
270                             filaAptos.push(arrayProcess[k]);
271                             processsoAdicionado = true;
272                             $scope.ordenarPrioridade(filaAptos);
273                             contadorTempoOcioso = 0;
274                             break;
275                         } else {
276                             contadorTempoOcioso++;
277                         }
278                     }
279                     if(!processoAtualZero) {
280                         contadorTempoOcioso = 0;
281                         break;
282                     }
283                     if(processsoAdicionado) {
284                         processsoAdicionado = false;
285                         contadorTempoOcioso = 0;
286                         break;
287                     }
288                     if((contadorTempoOcioso == arrayProcess.length) && (processoAtualZero)) {
289                         contadorTempoOcioso = 0;
290                             if(j == filaAptos.length - 1) {
291                                 validoQuanto = "Sim";
292                                 break;
293                             } else {
294                                 //console.log("não entrou em validar tempo ocioso");
295                             }
296                     }
297                 } else {
```

Figura 38 - Algoritmo de Prioridade - Parte 3

As Figuras 37 e 38 apresentaram as partes 1 e 2, respectivamente, sendo que a parte 1 da Figura 39 representa as declarações das variáveis iniciais. Na parte 2 tem-se o início dos laços de repetições, além das declarações de várias variáveis que são usadas neste trecho de código da Figura 38, a parte 3 do algoritmo de Prioridade. Nessa terceira parte, apresenta o que é realizado dentro dois laços de repetição, responsável por verificar se existem novos processos na linha do tempo e pelas condições de paradas dos laços de repetição.

Assim sendo, as instruções de códigos da Figura 39 iniciam na linha de código 265, com uma condição utilizando duas variáveis. A primeira variável é o `PCAtualMaior`, que teve alterações nos seus valores nas linhas 241 e 251 (Figura 38). E a segunda variável `ultimoProcesso`, que teve os valores alterados na linha 253 (Figura 38). Caso as duas ou uma das duas tenham valores verdadeiros, entram na condição da linha 265.

Logo após, entra em um outro laço de repetição (linhas 268 até 278) para verificar a possibilidade de novos processos. Caso tenha outro processo na linha do tempo atual (linha 269), o novo processo então é inserido na variável `filaAptos`. Esta por sua vez, é obrigatória a chamar a função de `ordenarPrioridade`, ilustrada na Figura 35. Ao retornar desta função,

os processos da `filaAptos` devem estar ordenados e prontos para executarem na linha do tempo deste algoritmo de Prioridade. Caso não encontre um processo ao percorrer toda a lista da tabela inicial, o contador do tempo ocioso é acrescentado mais um.

Ao terminar o laço de repetição, onde verifica se existe novos processos (linha 268 até 278), continua normalmente os restantes dos códigos. E nesses códigos são feitas algumas condições para verificar se existe as possibilidades de terminar o segundo laço de repetição bem como o algoritmo de Prioridade. Como é o caso apresentado na condição da linha 290, onde verifica se atingiu o limite da quantidade de processos.

Caso tenha atingido o limite da quantidade de processos (linha de código 290), significa que todos os processos já foram executados na linha do tempo e agora só precisa encerrar os dois laços de repetições. Caso não, o algoritmo segue normalmente a execução dos processos que ainda faltam na CPU.

Dessa maneira, encerra a condição da Figura 39. A próxima figura, finaliza o algoritmo de Prioridade, onde é verificada a variável que encerra os dois laços de repetições, esta variável denominada de contadorParada, da Figura 38.

```
298                 }
299                 }
300         }
301         if(validoQuanto == "Sim") {
302             validoQuanto = "Nao";
303             tempoOcioso++;
304         }
305         if(contadorParada) {
306             break;
307          }
308     }
309     console.log(arrayPrioridade);
310     prioridadeAvaliado = $scope.organizarPrioridade(arrayPrioridade, $processoPrioridade);
311     return prioridadeAvaliado;
312     };
```

Figura 39 - Algoritmo de Prioridade - Parte 4

A Figura 40 ilustra a última parte do Algoritmo de Prioridade. Na quarta parte, o algoritmo mostra as saídas dos laços de repetições, bem como uso de variáveis para realizar as condições de paradas que se apresentam nas linhas de códigos 301 e 305.

Ao se encerrar os laços de repetições, o algoritmo de Prioridade chama a função de `organizarPrioridade` (linha 310), da Figura 36. O retorno desta função é uma lista dos processos ordenados conforme a ordem de chegada dos processos na tabela inicial, mas com os valores de tempos totais e tempos médios calculados.

O algoritmo de Prioridade não aproveitou de nenhuma estrutura utilizadas nos algoritmos de FIFO e SJF. Para implementação do algoritmo, foi necessário utilizar três

funções: uma para ordenar; outra para organizar o algoritmo e calcular os tempos; e outra função para o algoritmo de Prioridade chamar essas duas funções e identificar o momento final para cada processo. Ao juntar essas três funções, foram utilizadas no total nove laços de repetições e vinte e duas condições de verificações.

4.4.4 Round-Robin

A implementação do algoritmo Round-Robin baseou-se em uma lista de processos, onde cada processo tem um limite de execução na CPU determinando pelo valor de *quantum*. A ordenação do Round-Robin é conforme as entradas dos processos na fila de aptos. O processo que chega é inserido no final da fila e o primeiro é retirado para execução. Cada processo tem um tempo determinado pelo *quantum* para usar a CPU, e quando acaba este tempo, ou o processo volta para fila de aptos ou encerra sua execução na CPU.

```
336    $scope.organizarRoundRobin = function($algoritmoRoundRobin, $algoritmoSistema) {
337        var avaliadoRoundRobin = [];
338        for(var i=0; i<$algoritmoSistema.length; i++) {
339            for(var j=0; j<$algoritmoRoundRobin.length; j++) {
340                if($algoritmoSistema[i].processoAvaliado == $algoritmoRoundRobin[j].processoResolucao) {
341                    var temTotal = 0;
342                    var temEspera = 0;
343                    temTotal = $algoritmoSistema[i].momentoFinalRound - $algoritmoRoundRobin[j].momentoTransicaoResolucao;
344                    temEspera = temTotal - $algoritmoRoundRobin[j].proximoCicloResolucao;
345                    avaliadoRoundRobin.push({Processo : $algoritmoSistema[i].processoAvaliado, TempoEspera : temEspera, TempoTotal : temTotal});
346                }
347            }
348        }
349        avaliadoRoundRobin.sort($scope.ordenarABC);
350        return avaliadoRoundRobin;
351    };
```

Figura 40 - Implementação do algoritmo de Round-Robin

A Figura 41 ilustra uma função denominada de `organizarRoundRobin`. Esta função organiza e calcula o tempo total e médio de uma lista de processos que vem do algoritmo de Round-Robin. A função então recebe como entrada duas variáveis, a primeira variável chamada de `$algoritmoSistema`, que são os dados dos processos alterados pelo algoritmo de Prioridade e a segunda, a `$algoritmoRoundRobin`, que representa a lista de processos da tabela inicial.

Desta forma, a função inicia com dois laços de repetições, sendo o primeiro laço tendo como critério de parada a quantidade de processos contidos na variável `$algoritmoSistema` (linha 338). O segundo laço de repetição tem como critério de parada, a quantidade de processos contidos na variável `$algoritmoRoundRobin` (linha 339).

Após os laços de repetições, começa com uma condição para verificar se o nome do processo atual da variável `$algoritmoSistema` é o mesmo nome da variável

`$algoritmoRoundRobin` (linha 340). Caso sejam mesmo, são criadas variáveis temporárias para receberem os valores de tempos totais e tempos médios.

Logo depois, essas variáveis são utilizadas para receber o tempo total (linha 343), o cálculo do momento final do processo atual na variável `$algoritmoSistema` menos o momento inicial na variável `$algoritmoRoundRobin`. Em seguida, as variáveis recebem o tempo de espera a entrada do cálculo do tempo total menos o próximo ciclo do processo atual na `$algoritmoRoundRobin`.

Após essas sequências, os valores dos tempos totais e médios, mais os nomes dos processos são inseridos em uma variável na linha 345. Variável que irá retornar para a função do algoritmo de Round-Robin, após terminarem os laços de repetições.

Como o algoritmo de Round-Robin tem uma grande quantidade de linhas de códigos, ele foi dividido em partes para melhores explicações sobre o funcionamento do algoritmo.

```
353    $scope.ajustarRoundRobin = function($algoritmoRound, $quantum) {
354        var arrayInicialRound = []; //reavaliar
355        var algoritmoAvaliado = [];
356        var filaAptos = [];
357        var algoritmoRobinFinal = [];
358        var quantumRound = $quantum;
359        var controladorQuantum = 0;
360        var contadorMomentoFinal = 0;
361        var linhaTempo = 0;
362        var valorAutal;
363        var contadorParada = 0;
364        var pararAlgoritmo = false;
365        for(var i = 0; i < $algoritmoRound.length; i++)
366        {
367            linhaTempo += $algoritmoRound[i].proximoCicloResolucao;
368            linhaTempo += $algoritmoRound[i].momentoTransicaoResolucao;
369            arrayInicialRound.push({processoRound: $algoritmoRound[i].processoResolucao,
370            proximoCicloRound: $algoritmoRound[i].proximoCicloResolucao,
371            momentoTransicaoRound: $algoritmoRound[i].momentoTransicaoResolucao});
372        }
373        filaAptos.push(arrayInicialRound[0]);
```

Figura 41 - Algoritmo de Round-Robin - Parte 1

A Figura 42 apresenta a primeira parte do algoritmo de Round-Robin, contendo uma função que se chama de `ajustarRoundRobin`, que recebe como entrada, dois parâmetros/variáveis. O primeiro parâmetro/variável é a tabela inicial dos processos (`$algoritmoRound`) e a segunda variável é o valor do *quantum* (`$quantum`). Após isso, o restante dos códigos da primeira parte são declarações de variáveis temporárias utilizadas ao longo do algoritmo.

Depois das declarações das variáveis, inicia um laço de repetição para conhecer a linha do tempo utilizada como base para o tempo na CPU. Além das inserções dos processos da tabela inicial para outra variável, prevenindo que os processos da tabela inicial não sejam alterados pelo algoritmo de Round-Robin.

```
374         for(var tempo = 1; tempo < linhaTempo; tempo++) {
375             for(var j = 0; j < filaAptos.length; j++) {
376                 if(filaAptos[j].proximoCicloRound > 0) {
377                     if(controladorQuantum < quantumRound) {
378                         controladorQuantum++;
379                         filaAptos[j].proximoCicloRound = filaAptos[j].proximoCicloRound - 1;
380                         if(filaAptos[j].proximoCicloRound == 0) {
381                             contadorParada++;
382                             algoritmoAvaliado.push({processoAvaliado: filaAptos[j].processoRound,
383                             momentoFinalRound: tempo});
384                             controladorQuantum = 0;
385                             filaAptos.shift();
386                             if(contadorParada == arrayInicialRound.length) {
387                                 pararAlgoritmo = true;
388                             }
389                             break;
390                         } else {
391                             var quantumPrevisto = controladorQuantum;
392                             if(quantumPrevisto == quantumRound) {
393                                 valorAutal = filaAptos[j];
394                                 filaAptos.shift();
395                                 filaAptos.push(valorAutal);
396                                 controladorQuantum = 0;
397                                 break;
398                             } else { break; }
399                         }
400                     }
401                 }
402             }
403             for(var p = 0; p < arrayInicialRound.length; p++) {
404                 if(arrayInicialRound[p].momentoTransicaoRound == tempo) {
405                     filaAptos.push(arrayInicialRound[p]);
406                     break;
407                 }
408             }
409             if(pararAlgoritmo) {
410                 break;
411             }
412         }
```

Figura 42 - Algoritmo de Round-Robin - Parte 2

Após as declarações das variáveis na primeira parte do algoritmo da Figura 43. A segunda parte apresenta outro laço de repetição utilizado, tendo como critério de parada o total da variável linha, declarada na linha 361 e alterada nas linhas de códigos 367 e 368. Dentro desse laço de repetição, começa outro laço de repetição com o limite de parada a quantidade de processos contidos na `filaAptos`.

Aos iniciarem os laços de repetições, linha de código 376, é feita uma condição para saber se o Próximo Ciclo do processo atual é maior que zero. Caso seja maior, é obrigado a passar por outra condição, linha de código 377, para verificar se o contador do *quantum* é menor que o valor real do *quantum*. Caso seja menor, o contador do *quantum* é acrescido mais um.

Logo depois é feita a subtração de menos um com o próximo ciclo do processo atual (linha de código 379).

Ao subtrair o próximo ciclo do processo atual, é feita uma verificação para saber se o próximo ciclo depois da alteração é igual a zero (linha de código 380). Caso seja igual a zero, significa que o processo atual encerrou sua execução na CPU e que está disponível o valor do momento final para o armazenamento. Nesse caso, após entrar na condição da linha de código 380, o que seria o próximo ciclo igual a zero, é utilizada uma variável chamada de `contadorParada`, que controla o valor limite do *quantum*. Ao inserir o processo atual concluído na CPU, passa este para outra variável `algoritmoAvaliado` (linhas de códigos 382 e 383). Após isto, o segundo laço de repetição é encerrado.

Porém, se o próximo ciclo do processo atual for diferente de zero entra no (`senão`) da condição (linha 390), verifica a possibilidade de o valor do *quantum* ter sido atingido pelo processo atual (linha de código 392). Caso tenha atingido, o processo atual é trocado de posição na variável de `filaAptos`, passando da primeira para a última posição. Caso não tenha atingido o valor do *quantum*, significa que o processo não terminou sua execução na CPU, de acordo com o *quantum* definido, e inicia novamente na linha 376, com o valor de *quantum* não alterado.

A cada volta no segundo laço de repetição e antes de terminar o primeiro laço de repetição é realizada uma verificação para encontrar algum processo da tabela inicial na linha do tempo atual. Caso seja encontrado um processo, este é inserido na última posição da variável `filaAptos`. Depois disto, ao terminar o primeiro laço de repetição (linha 409) é feita uma verificação para identificar se o todos os processos já foram executados e finalizados na CPU.

A última parte do algoritmo de Round-Robin é apresentada na próxima figura, onde é feita a chamada à função de organização.

```
413         algoritmoRobinFinal = $scope.organizarRoundRobin($algoritmoRound, algoritmoAvaliado);
414         return algoritmoRobinFinal;
415     };
```

Figura 43 - Algoritmo de Round-Robin - Parte 3

A Figura 44 apresenta as duas últimas linhas de códigos do Algoritmo de Round-Robin, após passar por declarações de variáveis (Figura 42) e pelos laços de repetições (Figura 43). Nesse trecho de código, na linha de código 413, chama-se a função de `organizarRoundRobin`, da Figura 41, para organizar os algoritmos passando como parâmetros as listas de processos alterados com os seus momentos finais e uma lista de

processos da tabela inicial. Por fim, retorna esta lista para uma variável que será utilizada na página do usuário.

O algoritmo Round-Robin pode aproveitar algumas estruturas utilizadas no algoritmo de Prioridade. Neste algoritmo, os processos podem sofrer alterações nas filas de aptos bem como nas execuções dos processos na CPU, através do controlador do *quantum*. Visto isso, a característica deste algoritmo é quando um processo está em execução na linha do tempo ao atingir ou não o valor do *quantum*, podem acontecer duas situações possíveis.

A primeira situação é se o processo ao atingir o limite do valor do *quantum*, se está com o seu Próximo Ciclo (este que é subtraído a cada interação do laço do *quantum*) maior que zero, passando o espaço da CPU para outro processo. E a segunda condição, é se o processo encerrou o seu ciclo de vida (Próximo Ciclo igual a zero) na CPU, e passando para outro processo o espaço da CPU.

Assim, para realizar essas comparações e troca de processos na fila de aptos e na linha do tempo da CPU, o algoritmo utilizou-se de duas funções: uma para organizar e calcular os processos e a outra para representar o algoritmo de Round-Robin, além de chamar a função de Organização. Ao juntar essas duas funções, foram necessários utilizar seis laços de repetições e dez condições de verificações (`ifelse`), para encontrar os valores dos momentos finais, tempos totais e tempos médios dos processos, além da possibilidade de ocorrência do tempo ocioso.

5 CONSIDERAÇÕES FINAIS

Há vários métodos aplicados para auxiliar a aprendizagem do aluno quanto às disciplinas consideradas complexas ou abstratas, um dos métodos aplicados são as resoluções de vários exercícios (VIEIRA, 2010). Assim sendo, esse trabalho propôs-se então desenvolver um sistema *web* que corrige os exercícios dos algoritmos de escalonamento de processos, conteúdo presente na disciplina de Sistemas Operacionais.

Para o desenvolvimento desse trabalho teve que considerar o estudo sobre o conteúdo de escalonamento de processo e os modos de correções utilizados para resolver os exercícios dos quatro algoritmos de escalonamento de processos, para poder planejar a fase inicial de implementação desse sistema.

Com base nisso, foi possível então desenvolver um sistema em que o usuário pode selecionar um exercício e um algoritmo de escalonamento. Posteriormente, o sistema consegue corrigir o exercício e apresentar os resultados finais ao usuário. Mas, primeiramente, o sistema deve que oferecer as opções de “Exercícios Cadastrados” e de “Criar Exercícios”, para que o usuário pudesse selecionar um exercício e um algoritmo de escalonamento para avaliação.

Com essa possibilita de seleção, e inclusive é uma das vantagens apresentada nesse sistema, o usuário tem a livre escolha de vários exercícios, seja criando ou selecionado em uma lista, para resolver ao final. Por outro lado, para cada exercício resolvido o sistema apresenta o resultado final, onde é possível o usuário visualizar o seu próprio desempenho. A nível de informação, o modo de correção aplicado nesse sistema para resolver os exercícios de escalonamentos de processos, é o mesmo aplicado nas salas de aulas na disciplina de Sistemas Operacionais no CEULP/ULBRA.

Além disso, outra vantagem apresentada nesse sistema é de estar disponível na internet, sem o usuário precisar instalar programas ou extensões para o funcionamento desse sistema. Não apenas isso, esse sistema é responsivo e permite que o usuário utilize tanto em computadores como em dispositivos móveis. Dessa forma, o usuário pode aproveitar desse sistema para poder resolver os exercícios dos algoritmos de escalonamento de processos em diversos dispositivos conectados na *web*.

Para trabalhos futuros, sugere-se desenvolver novos métodos de implementação dos algoritmos que resolvem os exercícios dos quatro algoritmos de escalonamento de processos. Além de implementar um sistema de gerenciamento de usuário, com cadastro e dados dos usuários, a fim de tornar possíveis novas funcionalidades nesse sistema.

REFERÊNCIAS

BITTENCOURT, Paulo H. M. **Ambientes Operacionais**. Biblioteca Universitária Pearson. São Paulo: Pearson Education do Brasil, 2014.

BRASIL. Ministério da Educação. **Parecer CNE/CES Nº 136/2012, aprovado em 8 de março de 2012**. Diretrizes Curriculares Nacionais para os cursos de graduação em Computação. Disponível em:<http://portal.mec.gov.br/component/content/article?id=12991>. Acesso em: 30 maio 2017.

CARDOZO, Eleri; MAGALHÃES, Maurício F. **Introdução aos Sistemas Operacionais**. Universidade Federal de Uberlândia, 2002. Disponível em:<ftp://vm1-dca.fee.unicamp.br/pub/docs/ea876/so-apst.pdf>. Acesso em: 01 abril 2017.

DEITEL, Paul J.. DEITEL, Harvey M.. **Ajax, Rich Internet Applications e desenvolvimento Web para programadores**. São Paulo: Pearson Prentice Hall, 2008.

DEITEL, H. M; DEITEL, P .J; CHOFFNES. **Sistemas Operacionais**. 3. ed. São Paulo: Pearson Prentice Hall, 2005.

EDUARDO, Carlos. **HTML5 Local Storage**: armazenamento de dados no navegador. 2014. Disponível em:<http://www.kadunew.com/blog/html/html5-local-storage-armazenamento-de-dados-no-navegador>. Acesso em: 20 setembro 2017.

FLANAGAN, David. **JavaScript**: the definitive guide. O`Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol. 2011.

FLATSCHART, Fábio. **HTML5**: embarque imediato. 1. ed. Rio de Janeiro: Brasport, 2011.

GRILLO, Filipe D. N. FORTES, Renata P. D. M. **Aprendendo JavaScript**. São Carlos, 2008. Disponível em:<http://conteudo.icmc.usp.br/CMS/Arquivos/arquivos_enviados/BIBLIOTECA_113_ND_72.pdf>. Acesso em: 04 junho 2017.

IAMASHITA, Clodoaldo H; MAGALHÃES, Willian B. **HTML5**: um estudo sobre seus novos recursos. Universidade Paranaense de Paranavaí, Paraná, 2013. Disponível em: <http://antigo.unipar.br/~seinpar/2013/artigos/Clodoaldo%20Hiroiti%20Iamashita.pdf>. Acesso em: 20 setembro 2017.

JANDL, Peter Jr.. **Notas sobre Sistemas Operacionais**. 2004. Disponível em:<https://docente.ifrn.edu.br/rodrigotertulino/livros/notas-sobre-sistemas-operacionais>. Acesso em: 17 fevereiro 2017.

MAUGET, Lou. **HTML5 Canvas e Fabric.js**: reinforcing the HTML5 Canvas With Fabric.js. 2016. Disponível em:<https://keyholesoftware.com/2016/02/08/reinforcing-the-html5-canvas-with-fabric-js/>. Acesso em: 27 maio 2017.

MAZIERO, Carlos A.. **Sistemas Operacionais:** conceitos e mecanismos. 2014. Disponível em: <http://wiki.inf.ufpr.br/maziero/lib/exe/fetch.php?media=so:so-livro.pdf>. Acesso em: 01 fevereiro 2017.

MILANI, André. **MySQL**: guia de programador. São Paulo: Novatec Editora, 2006.

MILLER, Chase A. Anthony, Jhon. MEYER, Michelle M. MARTH, Gabor. Scribl: an HTML5 **Canvas-based graphics library for visualizing genomic data over the web**. Volume 29, página 381-383. Department of Biology, Boston College, 2012. Disponível em:<https://academic.oup.com/bioinformatics/article/29/3/381/256920/Scribl-an-HTML5-Canvas-based-graphics-library-for>. Acesso em: 30 maio 2017.

OLIVEIRA, Rômulo S. de; CARISSIMI, Alexandre da S.; TOSCANI, Simão S.. **Sistemas Operacionais**. 2. ed. Porto Alegre: Sagra Luzzatto, 2001. 20 p., il.

SCHMITZ, D.; LIRA, D. **AngularJS na prática**. 1. ed. AngularJS. 2016.

SHAY, William A. **Sistemas Operacionais**. São Paulo : Makron Books do Brasil Ed. Ltda, 1996. p., il.

SILBERSCHATZ, Abraham; GALVIN, Peter B.; GAGNE, Greg. **Fundamentos de Sistemas Operacionais**. 8. ed. Rio de Janeiro: Pearson Prentice Hall, 2013.

SILVA, Maurício S. **HTML5**: a linguagem de marcação que revolucionou a web. 2. ed. São Paulo, 2014. 149 p., il.

SILVA, Maurício S. **Bootstrap 3.3.5**: aprenda a usar o framework Bootstrap para criar layouts CSS complexos e responsivos. 1. ed. Novatec Editora, São Paulo, 2015.

SOUSA, Robson P; MOITA, Filomena M. C; CARVALHO, Ana B. G. **Tecnologias Digitais na Educação**. 1 ed. Campina Grande: EDUEPB, 2011.

SPURLOCK, Jake. **Bootstrap**. Editors: Simom St. Laurent and Meghan Blanchette, 2013. Disponível em:<http://wiki.ifs.hsr.ch/APF/files/bootstrap.pdf>. Acesso em: 30 maio 2017.

STUTZ, Dalmo. **Web Storage**: usando a API do HTML5 no armazenamento de dados locais de aplicações web. Estácio - Campus Nova Fribugo Rio de Janeiro, Brasil, 2013. Disponível em: <http://www.litteraemrevista.org/ojs/index.php/Littera/article/view/110/105>. Acesso em: 24 setembro 2017.

RAMAKRISHNAN, Raghu; GEHRKE, Johannes. **Sistemas de Gerenciamento de Banco de Dados**. tradução: Célia Taniwake. 3 ed - Dados Eletrônicos. Porto Alegre: AMGH, 2011.

RIBEIRO, Elvia N.; MENDONÇA, Gilda A. de A.; MENDONÇA, Alzino F. **A importância dos ambientes virtuais de aprendizagem na busca de novos domínios da EAD**. Instituto Federal de Goiás. 2007. Disponível em: <http://www.abed.org.br/congresso2007/tc/4162007104526AM.pdf>. Acesso em: 30 março 2017.

TANENBAUM, Andrew S.; BOS, Herbert. **Sistemas Operacionais Modernos**. 4. ed. São Paulo: Pearson Education do Brasil, 2016. 20 p., il.

TANENBAUM, Andrew S.; WOODHULL, Albert S.. **Sistemas Operacionais**: projeto e implementação. 2. ed. Porto Alegre: Bookman, 1999. Disponível em: <http://www.asfernandes.com/files/SistemasOperacionais_-_Tanenbaum_-_2Ed.pdf>. Acesso em: 01 março 2017.

VIEIRA, Fábia M. V. **Avaliação de software educativo**: reflexões para uma análise criteriosa. 2010. Disponível em: <http://tecnologiaeducativaup.blogspot.com.br/2010/10/concepcao-realizacao-e-avaliacao-de.html>. Acesso em 30 novembro 2017.

ZAYTSEV, Juriy**. Introduction to Fabric.js**. 2013. Disponível em: <https://www.sitepoint.com/introduction-to-fabric-js/>. Acesso em: 10 junho 2017.

APÊNDICES

APÊNDICE A - PROTÓTIPO DO AMBIENTE DE SIMULAÇÃO SOBRE EXERCÍCIOS E RESPOSTAS

O sistema irá apresentar um ambiente de exercícios, sendo estes referentes ao gerenciamento de processos, onde a única ação que o usuário irá fazer quando acessar o ambiente será de clicar no botão gerar exercício, onde será gerado um novo exercício, e apresentar os dados na tabela de processos para análise, conforme mostrado na figura 14. Logo em seguida o usuário poderá preencher os campos da tabela com suas respostas com os dados que lhe convém estarem corretos.

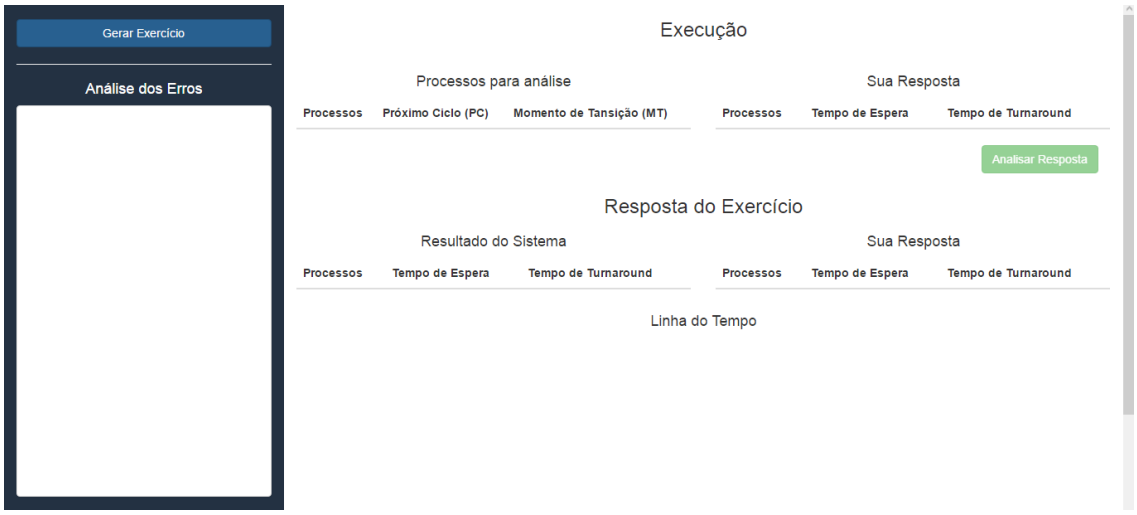


Figura 44 - Protótipo do ambiente de exercícios

Depois de preencher todos os campos, o botão analisar resposta será ativado para que o usuário possa clicar para verificar se sua resposta está correta ou não. O resultado das respostas será gerado na próxima tela, conforme a figura 45.

Figura 45 - Tela final do ambiente de exercício

A Figura 46, apresenta a tela do ambiente ao final do procedimento, onde é apresentado uma resposta do sistema (resposta correta) é a resposta do usuário para que o usuário possa fazer uma comparação entre as respostas. Também quando houver alguma resposta incorreta, será apresentado no campo à esquerda, indicando qual o erro, e uma explicação do porque está errado.