CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CAMPUS TIMÓTEO

Débora Cristina Ferreira

APISIM - UMA API RESTFUL PARA O GERENCIAMENTO DE RECURSOS DE SISTEMAS OPERACIONAIS

Timóteo

2017

Débora Cristina Ferreira

APISIM - UMA API RESTFUL PARA O GERENCIAMENTO DE RECURSOS DE SISTEMAS OPERACIONAIS

Monografia apresentada à Coordenação de Engenharia de Computação do Campus Timóteo do Centro Federal de Educação Tecnológica de Minas Gerais para obtenção do grau de Bacharel em Engenharia de Computação.

Orientador: Odilon Corrêa da Silva

Timóteo

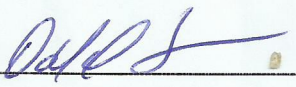
2017

Débora Cristina Ferreira

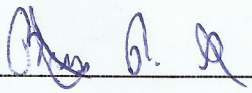
APISIM – UMA API RESTFUL PARA O GERENCIAMENTO DE RECURSOS DE SISTEMAS OPERACIONAIS

Monografia apresentada ao Curso de Engenharia de Computação do Centro Federal de Educação Tecnológica de Minas Gerais para a obtenção do título de Engenheiro de Computação.

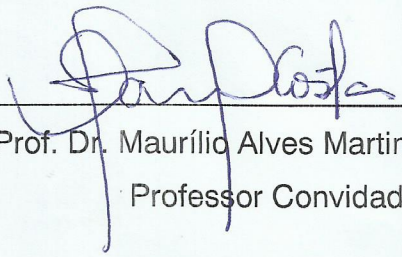
Trabalho aprovado. Timóteo, 16 de novembro de 2017:



Prof. Me. Odilon Corrêa Silva
Orientador



Prof. Dr. Bruno Rodrigues Silva
Professor Convidado



Prof. Dr. Maurílio Alves Martins da Costa
Professor Convidado

Timóteo
2017

Agradecimentos

Especialmente ao professor e orientador Odilon, pelo seu esforço, disponibilidade, dedicação para elaboração deste trabalho e incentivo para com os alunos de Computação.

À minha família, que sempre esteve presente na minha vida, como símbolo de união, dando incentivo e principalmente apoio mediante às dificuldades enfrentadas durante o meu período de graduação.

À minha querida avó, que não conseguiu me acompanhar até o fim da minha graduação, porém sempre estará em memória sendo parte da minha história e uma heroína que é um exemplo na minha vida.

À todos os amigos que estiveram ao meu lado nesse tempo, principalmente os que tive convívio na faculdade: Rainara, Marcos, Walace, Matheus, Gabriel, Rafael e Jefferson. Vocês foram de grande importância para a minha graduação, deixando lembranças positivas que levarei comigo por toda a vida.

Ao meu namorado Gustavo, pela paciência, apoio, incentivo e companheirismo nesta etapa da graduação, sempre me motivando e transparecendo positividade.

Aos professores que me incentivaram em suas disciplinas, sendo exemplos que quero seguir em minha carreira profissional.

E finalmente a todos que direta ou indiretamente contribuiram para minha formação.

“The world ain’t all sunshine and rainbows. It’s a very mean and nasty place and I don’t care how tough you are it will beat you to your knees and keep you there permanently if you let it. You, me, or nobody is gonna hit as hard as life. But it ain’t about how hard you hit. It’s about how hard you can get hit and keep moving forward. How much you can take and keep moving forward. That’s how winning is done!s”.

Rocky Balboa

Resumo

A criação de simuladores de Sistemas Operacionais para fins educativos tem por objetivo a apresentação de módulos visuais como suporte para o ensino e aprendizado de conceitos da disciplina. Com intuito de aperfeiçoar simuladores existentes, constantemente são desenvolvidas novas ferramentas que utilizam a mesma base existente na literatura de SO. Com o avanço da tecnologia, esses simuladores ficaram obsoletos, devido à arquitetura especificada no projeto. Foram realizadas análises de simuladores que utilizam a técnica de Simulação de Algoritmos e foi identificado um problema: a evolução do software fica restrita à especificação da arquitetura. Foi desenvolvido um serviço de gerência de processos aplicando o modelo arquitetural Rest. A validação da API foi realizada através da resolução de quatro exercícios, com o objetivo de exibir o resultado do escalonador. Por meio do estudo de caso, pode-se visualizar o escalonamento dos processos em plataformas diferentes. Constatou-se que a API pode ser utilizada sem nenhuma restrição de tecnologia, possibilitando o aproveitamento dos algoritmos disponilizados na APISIM. Sendo assim, a API possibilita a criação de novos simuladores sem estar preso à nenhuma especificação de arquitetura.

Palavras-chave: sistemas operacionais, simulação de algoritmos,políticas de escalonamento, processos.

Abstract

The creation of operating system simulators for educational purposes has as its objective the presentation of visual modules as support for teaching and learning concepts of operating systems. In order to optimize existing simulators, new tools are constantly being developed that use the same existing base in the literature of Operating Systems. With the advancement of technology, these simulators became obsolete due to the architecture specified in the project. Analyzes of simulators using the algorithm simulation technique were performed and a problem was identified: software evolution is restricted to the architecture specification. A process management service was developed applying the architectural model Rest. The validation of the API was accomplished through the resolution of four exercises, in order to show the result of the scheduler. Through the case study, one can visualize the scheduling of the processes in different platforms. It was found that the API can be used without any technology restriction and there is no need to implement the literature algorithms that were contemplated in this version. The API allows the evolution of the simulator without being attached to any architecture.

Keywords: operating systems, algorithm simulation, scheduling , processes.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	Application Programming Interface
CPU	Central Processing Unit
CSS	Cascading Style Sheets
FIFO	First in first outHTML - HyperText Markup Language
JDK	Java SE Development Kit
JNLP	Java Network Launch Protocolo
JRE	Java Runtime Environment
JSON	JavaScript Object
NPAPI	Netscape Plugin Application Programming Interface
PC	Programa Counter
PCB	Process Control Block
REST	Representational State Transfer
SJF	Shortest Job First
SO	Sistemas Operacionais
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
STS	Spring Tool Suite
TCB	Thread Control Blocks
URN	Uniform Resource Name
XML	Extensible Markup Language

Sumário

1 Introdução

A disciplina de Sistemas Operacionais (SO) é obrigatória em todos os cursos de Computação do Brasil. O ensino de SO objetiva apresentar aos alunos os conceitos e mecanismos utilizados por um sistema operacional (SBC, 2003). De acordo com PENNA (2013), o ensino de arquitetura de sistemas operacionais em cursos de graduação na área de computação é fundamental. No qual proporciona ao aluno um melhor esclarecimento dos modelos de programação sequencial e paralelo, conferindo os pré-requisitos necessários para o desenvolvimento de software e de arquitetura de Sistemas Operacionais.

De acordo com Maia (2001) um problema comum entre professores que lecionam a disciplina de sistemas operacionais, é a dificuldade em caracterizar a real dinâmica dos eventos computacionais. Apesar da qualificação do professor e domínio do conteúdo, constata–se uma dificuldade dos alunos no entendimento dos conceitos abordados na disciplina.

Segundo Maziero (2002), uma das principais particularidades da disciplina de Sistemas Operacionais é a relativa dificuldade em definir um sequenciamento didático claro entre seus diferentes tópicos. Com a identificação desse problema, verificou-se que existe uma necessidade do professor utilizar instrumentos pedagógicos que facilitem a capacidade dos alunos de construir abstrações, entender os mecanismos envolvidos da teoria de Sistemas Operacinais e proporcionar ao aluno uma compreensão dos conceitos fundamentais envolvidos. A construção de uma componente prática que enriqueça as aulas teóricas é importante para a apresentação de recursos de SO. A adoção de simuladores proporciona uma visão geral do funcionamento das gerências de recursos, com intuito de expor o aluno a uma metodologia de aprendizado prática. Nesse contexto, o uso de simuladores de arquiteturas vem sido apontado como uma boa ferramenta de aprendizado.

De acordo com Maia (2001), as principais técnicas utilizadas nas aulas práticas de sistemas operacionais são: uso de abstrações do núcleo, análise por inferência, simulação de algoritmos, exploração de código em sistemas reais e uso de núcleos simulados. Os simuladores apresentados neste trabalho utilizam a técnica de simulação de algoritmos, que consiste na criação de programas que simulem algoritmos, exibindo resultados ou animações gráficas.

Maziero (2002) enfatiza que o objetivo da técnica de simulaçao de algoritmos é facilitar a compreensão dos algoritmos comumente encontrados na literatura para a definição das políticas internas do núcleo do sistema operacional. De acordo com Maia (2001), os simuladores oferecem uma forma mais acessível a professores e alunos de estudar os mecanismos básicos de um sistema operacional, sem entrar em detalhes de instalação do software, da arquitetura de hardware e programação assembly.

O Minix é um sistema operacional gratuito desenvolvido por Tanenbaum (1987) criado com intuito de prover uma ferramenta de ensino para seus alunos. O Minix utiliza a técnica de exploração de código em sistemas reais. Tanenbaum observou que, o entendimento e exploração dessa ferramenta possui um alto nível de complexidade, e constatou uma maior

dificuldade de aprendizado por parte dos alunos. Foi criado então, o simulador Moss, disponibilizado também por Tanenbaum (2001) em conjunto com discentes, que consiste em um material suplementar ao seu texto (TANENBAUM; WOODHULL,), que utiliza a técnica de Simulação de Algoritmos para o ensino de SO. O software foi desenvolvido para estudantes e professores apresentando o conteúdo dos capítulos de Escalonamento, Deadlocking, Gerência de memória e Sistema de Arquivos. Após a criação do software Moss, outros autores detectaram a necessidade de implementar novas funcionalidades e adicionar novos módulos. Com o objetivo de enriquecer o trabalho de (ONTKO; REEDER; TANENBAUM, 2001) e outras ferramentas de aprendizado foram criadas utilizando a mesma técnica voltada para aulas práticas de sistemas operacionais.

A importância da construção de simuladores vem sido discutida entre autores da área de Sistemas Operacionais. As ferramentas estão sendo usadas em adição ao material teórico para que professores e alunos possam conhecer melhor os conceitos.Com intuito de facilitar a compreensão dos conteúdos abstratos da disciplina que possui uma grande ligação entre os módulos.

A evolução dos simuladores de sistemas operacionais tem sido assunto de pesquisa de forma recorrente. Como requisito, busca-se criar uma ferramenta que consiga englobar as funcionalidades existentes na literatura aliada a uma interface rica, simples e de fácil entendimento. A criação de novos simuladores com tecnologias novas que ilustrem os módulos de Sistemas Operacionais e simplifiquem a visualização dos recursos, que são objetos de estudo em trabalhos na área de computação. Entre outros, podem ser citados:

- MOSS: Modern Operating Systems Simulators (ONTKO; REEDER; TANENBAUM, 2001);
- SOsim: SIMULADOR PARA O ENSINO DE SISTEMAS OPERACIONAIS (MAIA, 2001);
- SISO: UMA PROPOSTA DE SOFTWARE EDUCACIONAL SIMULADOR PARA ENSINO DE SISTEMAS OPERACIONAIS (TEIXEIRA, 2001);
- Desenvolvimento ergonômico de uma interface gráfica para o software educacional SISO - Simulador de Sistemas Operacionais: Módulo de Detecção de Deadlock (MARTINS, 2005);
- TBC-SO WEB: Um Software Educacional para o Ensino de Politicas de Escalonamento de Processos e de Alocacao de Memoria em Operacionais (REIS, 2009);
- SimulaRSO – Simulador de Recursos de Sistemas Operacionais (ARAUJO, 2011);
- Simulador de Rotinas do Sistema Operacional para Auxílio às Aulas Teóricas (RIBEIRO; BERNARDES; LOBO, 2014).

1.1 Problema e sua importância

A recorrência de trabalhos desenvolvidos com objetivo de aperfeiçoar outros projetos, tornou-se algo repetitivo e trabalhoso. A implementação dos algoritmos existentes na bibliografia típica de Sistemas Operacionais sempre segue o mesmo padrão, tendo então uma evolução

nos trabalhos relativa a mudanças de linguagem de programação, estruturas e a plataforma no qual o simulador foi construído. Novos módulos são adicionados com intuito de englobar mais contéudo e interligá-los, de forma a oferecer simuladores mais robustos, porém sempre estão presos à arquitetura definida no trabalho.

As ferramentas criadas que utilizam a técnica de Simulação de Sistemas Operacionais foram disponibilizadas em diferentes ambientes (Web ou Desktop). Uma restrição importante identificada nesses simuladores é que foram desenvolvidos em arquiteturas especifícas. Devido a esse problema, a evolução da ferramenta fica presa à tecnologia definida pelo projeto. Resultando então, na dificuldade de evoluir suas funcionalidades e interface gráfica.

O simulador utilizado para fins educacionais vem mostrando resultados positivos e a criação de trabalhos voltados para o aperfeiçoamento dessas ferramentas ainda estão em pauta, com objetivo de fornecer materiais para o estudo de Sistemas Operacionais. Identifica-se então a ausência de uma base conjunta padronizada que contenha os algoritmos de Sistemas Operacionais para a utilização na criação de novas ferramentas. Nesse contexto, surge a seguinte questão: Como permitir que as bases de gerências dos simuladores de Sistemas Operacionais evoluam, sem enclausurá-las à sua especificação?

1.2 Objetivos

Para responder ao problema proposto, o objetivo geral deste trabalho é especificar, modelar e implementar uma API simples,extensível e incremental, tendo a sua base principal a gerência de processos, ou seja, as principais políticas de escalonamento de processos.

1.2.1 Objetivos específicos

Para alcançar o objetivo geral proposto, foram definidos os seguintes objetivos específicos:

1. Estudar os algoritmos de escalonamento e selecionar as principais políticas existentes na literatura de Sistemas Operacionais;
2. Investigar ferramentas que utilizam a técnica de Simulação de Algoritmos;
3. Especificar um modelo de arquitetura que resolva o problema proposto;
4. Implementar as políticas de gerência de processos selecionadas;
5. Aplicar o modelo arquitetural na API para a criação de um simulador;
6. Avaliar a API desenvolvida com testes;
7. Validar a API através de estudo de casos.

2 Materiais e Métodos

Para a realização da pesquisa a respeito de conteúdos teóricos práticos de Arquitetura de Sistemas Operacionais utilizou-se livros disponíveis na biblioteca da própria instituição. A parte relativa ao estudo de simuladores educacionais fez-se o uso dos seguintes materiais: dissertações, artigos técnicos, teóricos e científicos acessíveis no portal da Capes, google acadêmico e na internet. Para o desenvolvimento da API proposta para o desenvolvimento do simulador de Sistemas operacionais, foram usados os seguintes recursos:

Notebook com as seguintes especificações:

- Processador: Intel(R) Core(TM) i5-4200U CPU @1.60GHz 2.30GHZz
- Memória RAM: 8GB, Single Channel DDR3, 1333MHz (1x4Gb);
- Disco Rigido: 500GB, SATA (5400 RPM);
- Sistema Operacional: Windows 7 (Sistema Operacional de 64 bits, processador com base em x64).

Os seguintes recursos foram utilizados para a criação do serviço:

- Plataforma utilizada: Eclipse Neon.3 (4.6.3);
  - Spring Tool Suite [1] ;
  - Versão 3.8.4.RELEASE;
- Linguagem de programação: Java;
- Gerenciamento de dependências:
  - Maven 3.3.9 [2] ;
- Bibliotecas:
  - Swagger [3];
  - JARX- RS 2.0[4] ;

[1] https://spring.io/tools/sts/all
[2] http://maven.apache.org/download.cg
[3] https://swagger.io/
[4] https://mvnrepository.com/artifact/javax.ws.rs/javax.ws.rs-api/2.0

Para a criação do cliente foram utilizados os seguintes recursos:

- Plataforma utilizada: Eclipse Java EE IDE for Web Developers.
  - Versão Oxygen Release (4.7.0);
- Linguagem de programação: Java, JavaScript;

Recursos utilizados para a realização de testes:

- Postman for Chrome Version 5.2.1 [5] ;

As seguintes etapas foram realizadas para a elaboração do trabalho:

1. Estudo direcionado sobre gerências de processos de Sistemas Operacionais;
2. Estudo e seleção de simuladores de Sistemas Operacionais;
3. Seleção de algoritmos de escalonamento para serem implementados na API;
4. Estudo de arquiteturas para a criação do simulador;
5. Desenvolvimento e testes dos algoritmos de gerência de processos;
6. Definição de um modelo arquitetural para a implementação do Simulador de Sistemas Operacionais;
7. Elaboração do serviço de gerência de processos utilizando o modelo arquitetural definido;
8. Avaliação dos resultados do escalonamento de processos disponibilizado pela API, através de exercícios sugeridos;
9. Elaboração de clientes Web/Desktop para estudos de caso da API;
10. Documentação das atividades desenvolvidas;
11. Apresentação do trabalho para a banca examinadora;
12. Aplicação das correções do trabalho sugeridas pela banca examinadora.

[5] https://www.getpostman.com/

3 Fundamentação Teórica

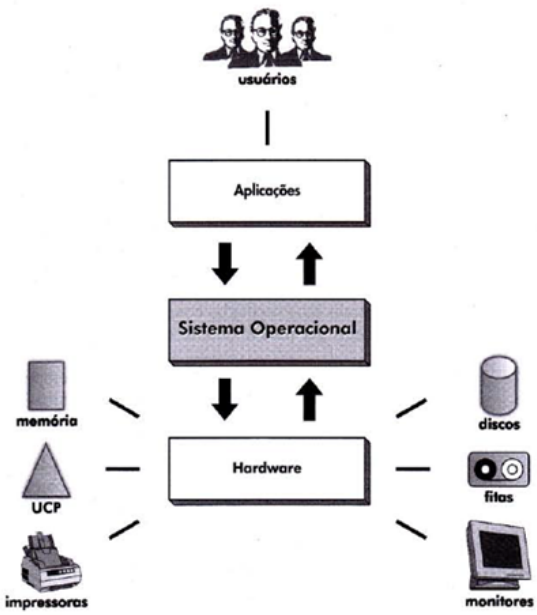
Este capítulo descreve os principais conceitos de Sistemas Operacionais e da arquitetura REST. Inicialmente, será apresentado o conceito de Sistemas Operacionais, seguido do ciclo de vida de processos, os principais critérios e algoritmos de escalonamento. Posteriormente, serão descritas as características de um web service e da arquitetura REST.

3.1 Conceitos de Sistema Operacional

O sistema operacional é responsável por gerenciar todos os recursos de um computador, alocando de forma ordenada e oferecendo o controle de processadores, das memórias e dos dispositivos de entrada e saída. Trata-se de uma camada de software criada por cima do hardware para isolar os usuários e programadores da complexidade do hardware. Com isso, foi possível oferecer uma interface de fácil compreensão e programação (TANENBAUM; WOODHULL,).

As aplicações geralmente são executadas de forma linear, com início, meio e fim. Um sistema operacional diferencia-se pela execução de rotinas de forma concorrente. Um sistema operacional diferentemente dos softwares aplicativos, a execução de suas rotinas ocorre de forma concorrente que são disparadas em função de eventos que ocorrem em qualquer instante (MACHADO; MAIA, 2007).

Figura 1 – Visão do Sistema Operacional



Fonte: MAIA, L. P.; MACHADO, F. B., 2007, p. 4

3.1.1 Sistemas de tempo compartilhado

O sistema de tempo compartilhado consiste na execução de programas através da utilização da fatia de tempo, que tem como objetivo a divisão do tempo do processador em curtos intervalos. Se após o término do intervalo a execução do programa não tenha sido concluída, o sistema operacional interrompe sua execução e o substitui por outro, enquanto aguarda pela nova fatia de tempo. A interrupção de um processo que estava sendo executado no processador é denominada preempção (MAZIERO, 2013).

Sistemas que implementam esse conceito são chamados sistemas preemptivos.Esse tipo de sistema permite a comunicação do usuário com o sistema operacional através de comandos, oferecendo tempo de respostas razoáveis à maioria desses comandos (MAZIERO, 2013).

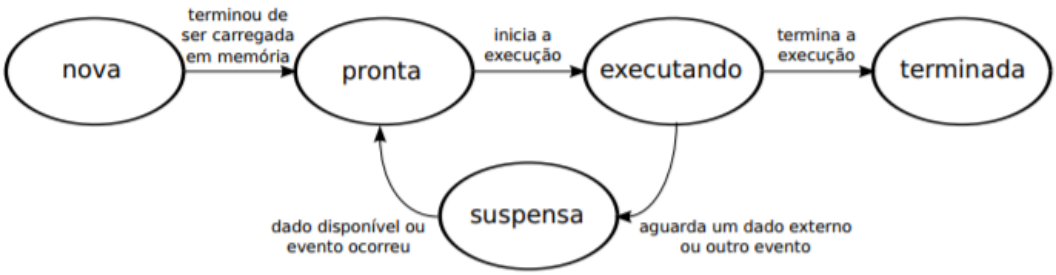
3.1.2 Processos

Um processo é um programa em execução que necessita de recursos: tempo de CPU, memória, arquivos e dispositivos de E/S – para realizar sua tarefa. Estes recursos são alocados ao processo quando ele é criado ou enquanto está sendo executado (SILBERSCHATZ; GALVIN; GAGNE, 2008).

Além de seu próprio código executável, cada tarefa ativa em um sistema de computação necessita de um conjunto de recursos para executar e cumprir seu objetivo. Entre esses recursos estão as áreas de memória usadas pela tarefa para armazenar seu código, dados e pilha, seus arquivos abertos, conexões de rede, etc. O conjunto dos recursos alocados a uma tarefa para sua execução é denominado processo (MAZIERO, 2013).

3.1.2.1 Estado do Processo

Figura 2 – Diagrama de estados de uma tarefa em um sistema multitarefas



FONTE: MAZIEIRO, 2013 , p. 8

Segundo Maziero (2013) os estados e transições do ciclo de vida de um processo têm o seguinte significado:

- **Nova**: A tarefa está sendo criada, i.e. seu código está sendo carregado em memória, junto com as bibliotecas necessárias, e as estruturas de dados do núcleo estão sendo atualizadas para permitir sua execução;

- **Pronta**: A tarefa está em memória, pronta para executar (ou para continuar sua execução), apenas aguardando a disponibilidade do processador. Todas as tarefas prontas são organizadas em uma fila cuja ordem é determinada por algoritmos de escalonamento;
- **Executando**: O processador está dedicado à tarefa, executando suas instruções e fazendo avançar seu estado;
- **Suspensa**: A tarefa não pode executar porque depende de dados externos ainda não disponíveis (do disco ou da rede, por exemplo), aguarda algum tipo de sincronização (o fim de outra tarefa ou a liberação de algum recurso compartilhado) ou simplesmente espera o tempo passar (em uma operação sleeping, por exemplo);
- **Terminada**: O processamento da tarefa foi encerrado e ela pode ser removida da memória do sistema;

3.2 Escalonamento de Processos

O escalonador de processo seleciona um processo disponível para execução na CPU. Para um sistema de processador único, nunca haverá mais do que um processo em execução. Se houver mais processos, o restante terá de esperar até que a CPU esteja livre e possa ser reescalonada.

Para que o processador possa interromper a execução de uma tarefa e retornar a ela mais tarde, sem corromper seu estado interno, é necessário definir operações para salvar e restaurar o contexto da tarefa. O ato de salvar os valores do contexto atual em seu TCB e possivelmente restaurar o contexto de outra tarefa, previamente salvo em outro TCB, é denominado troca de contexto. (MAIA,2001).

Um algoritmo de escalonamento tem como principal função decidir qual dos processos prontos para execução deve ser alocado à CPU. Cada sistema operacional necessita de um algoritmo de escalonamento adequado a seu tipo de ambiente (mobile, desktop ou servidor). A seguir, apresentaremos os principais critérios de escalonamento:

- Utilização da CPU - Na maioria dos sistemas é desejável que o processador permaneça a maior parte do seu tempo ocupado. Uma utilização na faixa de 30% indica um sistema com uma carga de processamento baixa, enquanto que na faixa de 90% indica um sistema bastante carregado, próximo da sua capacidade total.
- Throughput - representa o número de processos (tarefas) executados em um determinado intervalo de tempo. Quanto maior o throughput, maior o número de tarefas executadas em função do tempo. A maximização do throughput é desejada na maioria dos sistemas (MACHADO; MAIA, 2007).
- Tempo de turnaround - tempo que um processo leva desde sua admissão no sistema até ao seu término, levando em consideração o tempo de espera para alocação de memória, espera na fila de processos prontos para execução, processamento na CPU e operações de E/S (MACHADO; MAIA, 2007).

- Tempo de resposta - em sistemas interativos, o tempo de resposta é o tempo decorrido do momento da submissão de um pedido ao sistema até a primeira resposta produzida. Este tempo, geralmente, é limitado pela velocidade do dispositivo de saída. De uma maneira geral, qualquer algoritmo de escalonamento busca otimizar a utilização da CPU e o throughput, enquanto tenta diminuir os tempos de turnaround e de resposta (MACHADO; MAIA, 2007).
- Justiça - este critério diz respeito à distribuição do processador entre as tarefas prontas: duas tarefas de comportamento similar devem receber tempos de processamento similares e ter durações de execução similares (MAZIERO, 2013).
- Eficiência - indica o grau de utilização do processador na execução das tarefas do usuário. Ela depende, sobretudo da rapidez da troca de contexto e da quantidade de tarefas orientadas a entrada/saída no sistema (MAZIERO, 2013).

3.2.1 Algoritmos de Escalonamento

Ao ter mais de um processo para ser executado, o sistema operacional deve decidir qual será executado primeiro. A parte responsável por gerenciar as decisões é chamada de escalonamento de processos A quantidade de tempo de CPU disponível é finita, afinal de contas (TANENBAUM, 2001). Seguem abaixo os principais algoritmos encontrados na literatura:

3.2.2 First-Come First-Out – FIFO

O processo que requisite a CPU primeiro é executado. Quando um processo entra na fila de prontos seu PCB é associado ao final da fila. Quando a CPU está livre, ela é alocada ao processo na cabeça da fila. O processo em execução, em seguida, é removido da fila (SILBERSCHATZ; GALVIN; GAGNE, 2008).

3.2.3 Shortest Job First – SJF

Esse algoritmo associa a cada processo o tamanho do próximo burst de CPU do processo, quando a CPU estiver disponível, o processo que possui menor *burst* será executado. Se existirem *bursts* de CPU de dois processos iguais, o processo que requisitou a memória primeiro será executado. O algoritmo SJF é não-preemptivo (SILBERSCHATZ; GALVIN; GAGNE, 2008).

3.2.4 Shortest Remaining time

É o algoritmo SJF preemptivo que retira o processo atualmente em execução. A opção surge quando um novo processo chega à fila de prontos durante a execução de um processo anterior, o *burst* do novo processo pode ser menor do que resta do processo atualmente em execução (SILBERSCHATZ; GALVIN; GAGNE, 2008).

3.2.5 Round Robin (Escalonamento Circular)

A cada processo é atribuído um intervalo de tempo, que é o tempo que ele executa. Ao término do quantum, ocorre a preempção da CPU e ela é dada a outro processo. Se o processo terminou sua execução antes do término de seu *quantum*, é feita a comutação da CPU. Quando o processo utiliza todo seu quantum, ele vai para o final da fila (SILBERSCHATZ; GALVIN; GAGNE, 2008).

3.2.6 Escalonamento Circular por prioridades

O escalonamento circular com prioridades implementa o conceito de fatia de tempo de prioridade de execução associada a cada processo. Neste tipo de escalonamento um processo permanece no estado de execução até que termine seu processamento, e voluntariamente passe para o estado de espera ou sofra uma preempção por tempo ou prioridade.

Segundo Machado e Maia (2007), o algoritmo de escalonamento circular por prioridades permite o melhor balanceamento no Uso da CPU, com a possibilidade de diferenciar o grau de importância dos processos.

3.2.7 Escalonamento por prioridades

A cada processo é atribuído uma prioridade, e o processo executável com a maior prioridade recebe permissão para executar. Processos com a mesma prioridade são escalonados na ordem FIFO. Mesmo em um PC com um único proprietário, pode haver múltiplos processos, alguns mais importantes que outros (TANENBAUM, 2001).

O escalonamento por prioridades pode ser preemptivo ou não preemptivo. O preemptivo consiste em se apropriar da CPU se a prioridade do processo recém-chegado for mais alta do que a prioridade do processo em execução. O não preemptivo simplesmente colocará o novo processo no início da fila de pronto (SILBERSCHATZ; GALVIN; GAGNE, 2008).

3.3 Web Service

O Web Service é um tipo de aplicação para a web, seus componentes podem ser aplicados e executados de modo independente de plataforma. Sendo então, uma solução de conectividade para computação distribuída provendo interoperabilidade entre plataformas. A interoperabilidade é a capacidade de um sistema de se comunicar de forma transparente com outro sistema (KALIN, 2010).

Web Services são divididos em dois grupos, sendo o primeiro baseados em SOA e o segundo do estilo REST. O SOA é uma arquitetura orientada a serviços (SOA), no qual aplicações com arquitetura SOA combinam serviços para oferecer outros serviços. A principal forma de implementar SOAs hoje é através de Web Services. E a segunda é o REST (Representacional State Transfer), no qual, qualquer informação disponível é um recurso (KALIN, 2010).

3.3.1 Arquitetura REST

Fielding (2000) introduziu e definiu o estilo de arquitetura de Transferência de Representação de Estado, no qual, sua abstração central é denominada recurso. Um recurso é qualquer coisa que tenha URI, que é um identificador que satisfaça os requisitos de formatação. URI significa Identificador de Recurso, sendo assim as noções de URI e recursos estão interligadas (DIAS, 2016).

O modelo trata de uma abstração da arquitetura da Web, ou seja, consiste em regras que, quando seguidas, permitem a criação de um projeto com interfaces bem definidas. O estilo REST é uma abstração dos elementos arquiteturais de um sistema multimídia distribuído definidos como (FIELDING, 2000) :

- Elementos de dados: são classificados como os recursos, o metadado destes recursos e seus identificadores (URIs) e as suas representações que pode ser um documento HTML, XML, imagens etc;
- Conectores: são os vários tipos de conectores que REST utiliza para encapsular as atividades de acesso aos recursos e a transferência de uma representação de um recurso. Podemos classificar os conectores como clientes, servidores, cache e tunnel;
- Componentes: são classificados pelos seus papéis na ação de uma aplicação, os tipos de componentes são classificados como servidores de origem, *gateways*, *proxy* e *user agents*. Neste último caso, o exemplo mais comum é um navegador Web.
- Em uma arquitetura REST, dados e funcionalidades são considerados recursos e estes recursos são acessados via URI, geralmente via links. O uso de representações da arquitetura REST permite a portabilidade, transmitindo informações independente da tecnologia utilizada e facilita o entendimento devido ao fato de serem padrãos difundidos globalmente (MARTINS, 2015).

Quando criamos *web services* utilizando o estilo de arquitetura REST, estamos criando uma aplicação RESTful. Os *web services* RESTful expõem os recursos através de URIs e utilizam os métodos do HTTP para criar, retornar, alterar e excluir os seus recursos (MARTINS, 2015).

3.3.1.1 URI

Em computação, uma URI é uma *string* compacta única utilizada para identificar um recurso físico ou abstrato, geralmente na Internet. Uma URI pode ser classificada como um local (URL), um nome (URN) ou ambos. URL é o formato mais conhecido por ser o meio de localização de recursos que utilizamos ao navegar pela Internet. URN pode ser atribuído a um nome de uma pessoa ou pode ser utilizado para identificar um objeto, como, por exemplo, um código de barras ou o SBN de um livro. Enquanto uma URN define a identidade única de um item, a URL define a localização deste mesmo item na rede. Ela possui uma sintaxe genérica composta de quatro partes definidas pelo padrão (MARTINS, 2015).

3.3.1.2 Métodos

Quando fazemos uma requisição HTTP ao servidor web, podemos dizer que estamos enviando uma mensagem. O HTTP suporta diversos tipos de comandos de requisição e chamados de métodos http (MARTINS, 2015)

Toda mensagem HTTP possui um método. O método diz ao servidor qual a ação que ele deve tomar. Para tanto, foram disponibilizados alguns métodos HTTP, conhecidos também como verbos, pois o verbo irá determinar a ação a ser tomada no recurso. Na tabela 1 apresentamos os métodos mais comuns (MARTINS, 2015).

Tabela 1 – Operações REST

POST	Cria um novo recurso a partir de dados solicitados
GET	Lê um recurso
PUT	Atualiza
DELETE	Deleta

Fonte: (KALIN, 2010)

3.3.1.3 Código de Status

De acordo com **??**), toda resposta do servidor Web a uma requisição vem com um código de status utilizado para identificar o status da operação. Segue, na tabela 2, a classificação dos tipos de códigos de status que podemos receber de um servidor web:

Tabela 2 – Códigos de Status

Range definido	Descrição
100-101	Informacional
200-206	Sucesso
300-305	Redirecionamento
400-415	Erro do Cliente
500-505	Erro do Servidor

Fonte: Adaptado pelo próprio autor

3.3.1.4 Representações

Quando um recurso é solicitado por um cliente, o servidor executa uma série de atividades e retorna uma mensagem ao solicitante. Essa mensagem é de fato uma representação de um determinado recurso. A mesma não necessariamente precisa estar ligada a alguma parte concreta de um sistema, como por exemplo, uma tabela de banco de dados (KALIN, 2010).

As representações podem ser modeladas em vários formatos, como XML, JSON e YAML. A primeira representação (formato XML) é mais verbosa, exigindo um esforço extra por parte de quem está escrevendo. No segundo exemplo (formato JSON) já é algo mais leve de se escrever. Já o último (formato YAML), é praticamente como escrevemos no dia a

dia. Sendo assim, esse é o primeiro passo que precisamos dar para permitir a comunicação interoperável. As três representações são válidas atualmente, homens e máquinas podem ler, escrever e entender esses formatos (PIRES, 2017).

3.3.2 API RestFUL

O acrônimo API trata-se de um conjunto de rotinas e padrões estabelecidos e documentados por uma aplicação A, para que outras aplicações consigam utilizar as funcionalidades desta aplicação A, sem precisar conhecer detalhes da implementação do software.Desta forma, entendemos que as APIs permitem uma interoperabilidade entre aplicações. Em outras palavras, a comunicação entre aplicações e entre os usuários (PIRES, 2017).

De acordo com Reis (2016) os requisitos essenciais para a criação de uma API são:

- Possuir documentação adequada;
- Ser o mais simples e intuitivo possível (affordance), seja na URI, payload, request ou response;
- Deve-se levar em conta a experiência do usuário. Procurar seguir o padrão já utilizado pelo mercado, evitar “coisas” mirabolantes;
- Manter a padronização nas URIs, tipo de retorno, etc.

4 Estado da Arte

Simuladores tem como objetivo fornecer uma maior acessibilidade a professores e alunos de estudar os mecanismos básicos de uma área de conhecimento, sendo utilizada na engenharia, química, física, etc. Na computação eles auxiliam no aprendizado de redes de computadores,programação, arquitetura de computadores e sistemas operacionais.

No caso específico de simuladores para o estudo de sistemas operacionais existem os genéricos, que abordam a maioria dos módulos de gerência necessários a um sistema operacional, e os específicos, focados em algum módulo do sistema, como de escalonamento ou comunicação entre processos.

4.1 MOSS - Modern Operating Systems Simulator

Tanenbaum (2001) em conjunto com Ontko, Reeder desenvolveram um material suplementar ao livro (ONTKO; REEDER; TANENBAUM, 2001): o simulador de gerências Moss (Modern Operating Systems Simulators). É uma coleção de programas de simulação desenvolvidos em Java que ilustram conceitos chaves de Sistemas Operacionais. Os módulos desenvolvidos no simulador são: gerência de processos, deadlock, gerência de memória e sistemas de arquivos. Está disponível para sistemas Unix e Windows 95/98 / Me / NT / 2000 e sua instalação é feita através do prompt de comando do Sistema Operacional.

Na figura 3, podemos visualizar uma configuração de entrada de dados para o escalonamento de processos no MOSS:

Figura 3 – MOSS: Visão da interface gráfica do simulador

```
// # of Process
numprocess 3

// mean deivation
meandev 1100

// standard deviation
standdev 510

// process     # I/O blocking
process 100
process 500
process 30

// duration of the simulation in milliseconds
runtime 5000
```

Fonte: Adaptado pelo autor.

A seguir, o resultado da configuração do escalonamento da figura 3:

Figura 4 – MOSS: Resultado do escalonamento

```
Scheduling Type: Batch (Nonpreemptive)
Scheduling Name: First-Come First-Served
Simulation Run Time: 2750
Mean: 1100
Standard Deviation: 510
Process #       CPU Time        IO Blocking     CPU Completed   CPU Blocked
0               1372 (ms)       100 (ms)        1372 (ms)       13 times
1               689 (ms)        500 (ms)        689 (ms)        1 times
2               689 (ms)        30 (ms)         689 (ms)        22 times
```

Fonte:Adaptado pelo autor.

4.2 SOsim - Simulador para o Ensino de Sistemas Operacionais

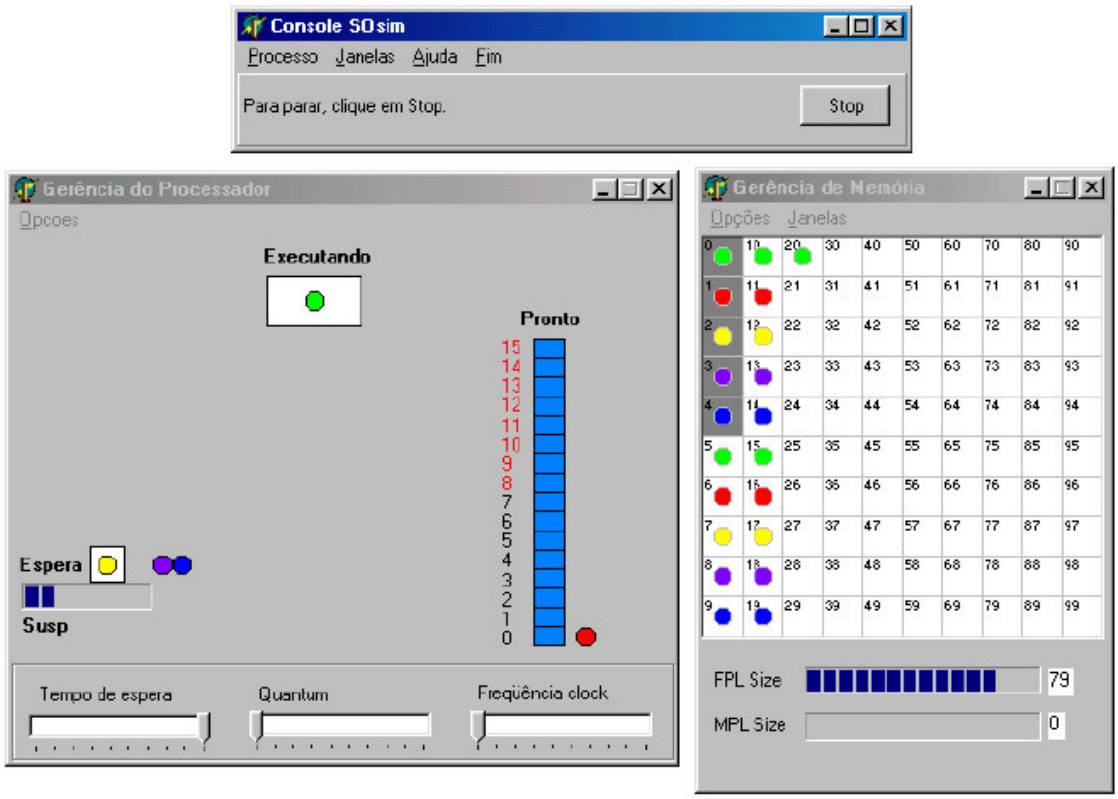
Maia (2001) criou um simulador gráfico (SOsim) genérico, que consiste em uma ferramenta visual de suporte efetivo para o ensino e aprendizado dos conceitos e técnicas implementados nos sistemas operacionais atuais. O simulador é de fácil instalação, não exige qualquer conhecimento de linguagem de programação para ser utilizado e pode ser executado em um computador pessoal (PC) que tenha o sistema operacional Microsoft Windows.

O software foi desenvolvido em Borland Delphi, utilizando-se o conceito de orientação a objetos, o que permite o fácil acesso ao código fonte e sua alteração.A última atualização do simulador foi em 2007, com ajustes para a integração ao livro (MACHADO; MAIA, 2007).

> O simulador permite visualizar os conceitos de multiprogramação, processo e suas mudanças de estado, gerência do processador (escalonamento) e a gerência memória virtual. A partir das opções de configuração, é possível selecionar diferentes políticas e alterar o funcionamento do simulador. Desta forma, o aluno tem a oportunidade de visualizar os conceitos teóricos apresentados em aula de forma simples e animados (MAIA, 2001).

Na figura 5, pode-se visualizar imagens do simulador SOsim:

Figura 5 – SOsim: Visão da interface gráfica do simulador



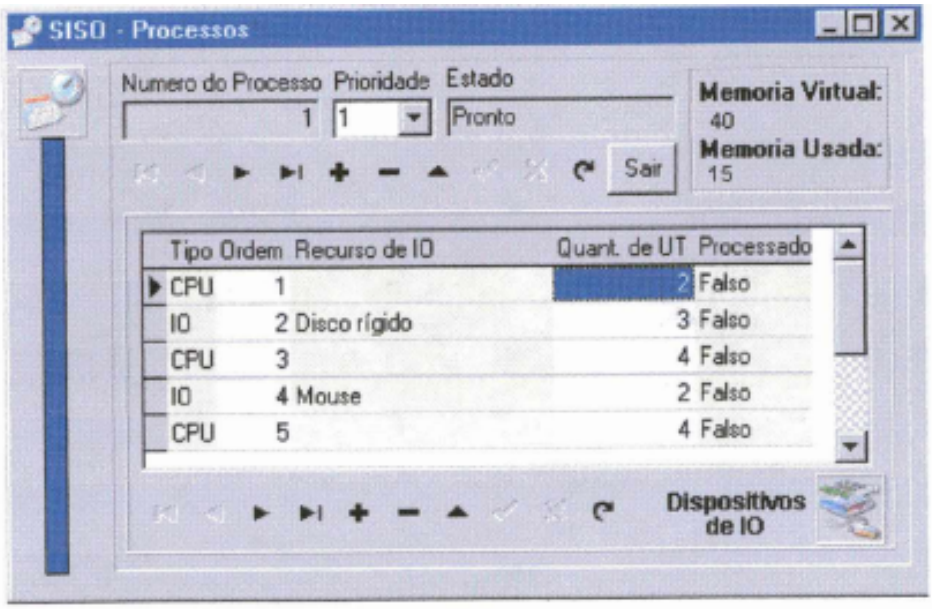
Fonte: Maia, 2001, p.16.

4.3 SISO : Simulador de Sistemas Operacionais

Teixeira (2001) em sua tese, modelou e implementou o protótipo de um software educacional que auxilia alunos na compreensão de assuntos específicos da disciplina Sistemas Operacionais. A temática escolhida foi o módulo de Gerência de Processos, especificamente nos assuntos: modelo de processo, estratégias de escalonamento de processador e detecção de situações de deadlock.

A autora, ao realizar um estudo dos simuladores existentes, visualizou que havia a necessidade de desenvolver um simulador focado na simulação de detecção de situação de deadlock. No estudo de deadlock, o aluno cria uma situação hipotética definindo o número de processos ativos e recursos existentes, realizando a alocação dos recursos para testar se uma situação apresenta deadlock ou não. O aluno pode observar em que situações específicas o sistema indica ocorrência de deadlock. Foi utilizada a linguagem de programação Delphi 5, utilizando o Banco de dados Paradox 7 no desenvolvimento do SISO. Na figura a seguir, visualiza-se a tela de criação de processos do simulador:

Figura 6 – Tela de entrada de processos



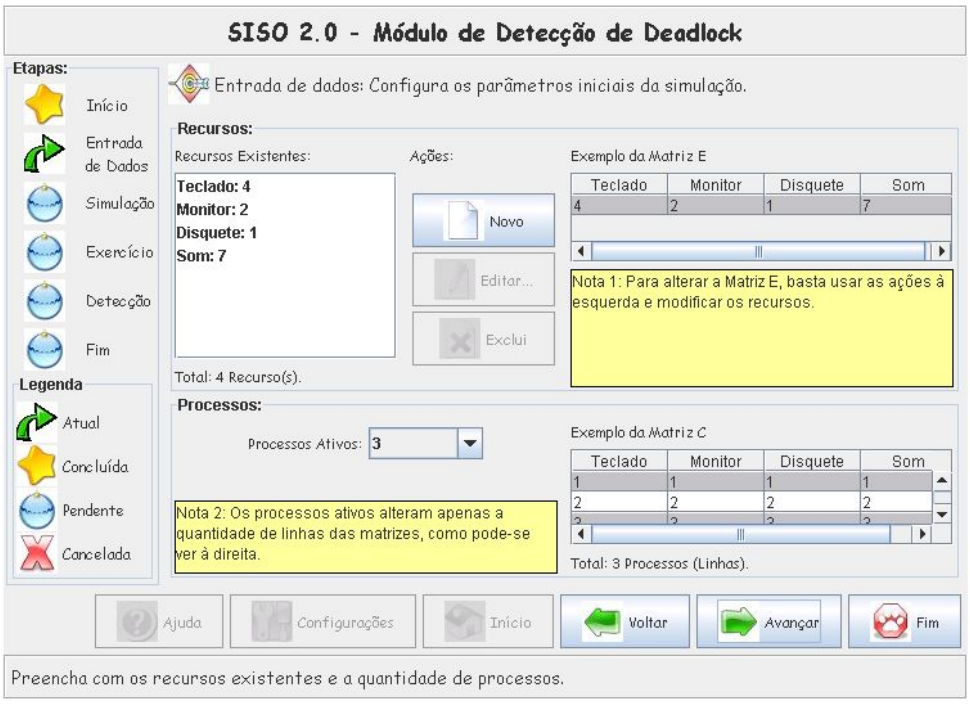
Fonte: Teixeira, 2001, p.139.

4.4 SISO 2.0: Simulador de Sistemas Operacionais

Martnis (2005) em seu trabalho buscou agregar maiores informações e requisitos no SISO, criando um novo simulador. Seu trabalho baseia-se na prática do uso do software, com ênfase em melhorias para o usuário. Elaborou um projeto de interface ergonômica e desenvolveu o simulador utilizando na linguagem Java. Anteriormente a linguagem utilizada pelo SISO era Delphi, com isso observa-se que o projeto foi construído novamente baseado na mesma fundamentação teórica.

O simulador utilizou a tecnologia de Java Applet, permitindo o acesso por qualquer browser de internet. Applets em Java podem rodar em um web browser usando uma Java Virtual Machine. Os requisitos de software mínimos do sistema são aos da instalação do Java Runtime Environment (JRE) 1.5. O acesso ao simulador não foi possível, pois a tecnologia Applet foi descontinuada pela empresa Oracle, e os navegadores não dão suporte mais para Applets. Pode-se visualizar na figura 7 como é o simulador:

Figura 7 – SISO 2.0 - Entrada de dados



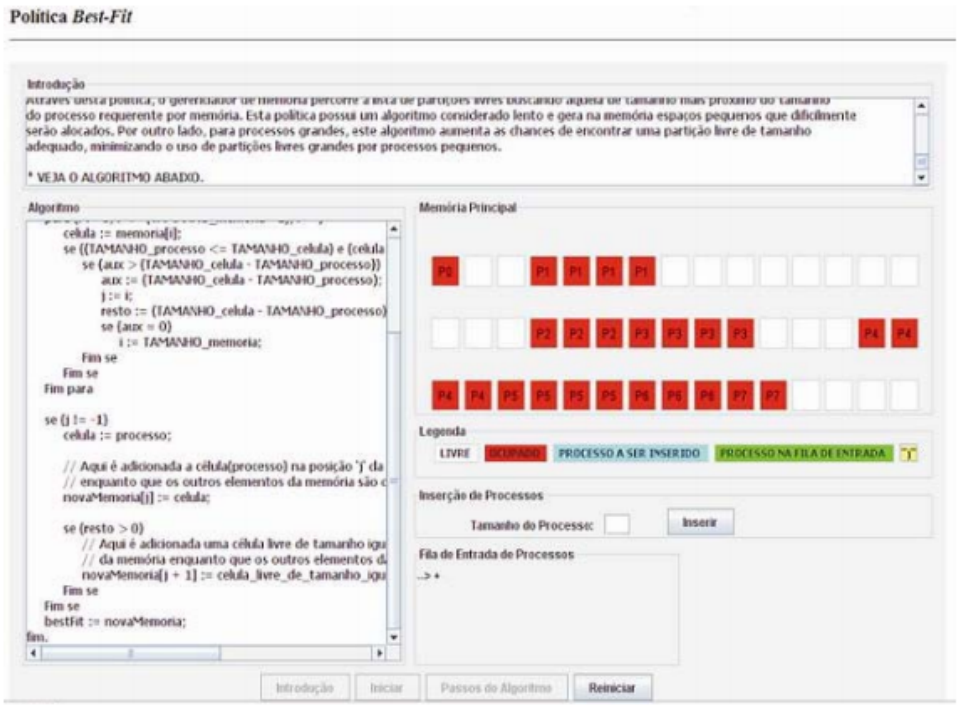
Fonte: Martins, 2005, p.82.

4.5 TBC-SO/WEB: Um Software Educacional para o Ensino de Políticas de Escalonamento de Processos e de Alocação de Memória em Sistemas Operacionais

Este trabalho simula as gerências de processos e de memória de Sistemas Operacionais. Foi realizado um levantamento de software existentes, que tratam o mesmo conteúdo. E, foi realizado uma construção de um novo simulador com o objetivo de estimular atualização das metodologias de ensino da área de Computação e Informática, com o desafio de melhorar a forma que os alunos assimilam o conteúdo. Este produto de software utiliza recursos gráficos animados com interface para a Web empregando a tecnologia Java (JSE – Java Standard Edition) para propiciar seu uso por várias pessoas e em qualquer lugar que tenha um computador com acesso a Web e a máquina virtual Java instalada (JVM – Java Virtual Machine) (REIS, 2009).

O simulador utilizou a mesma tecnologia utilizada no SISO 2.0 que é de Java Applet. Com isso o diferencial do TBC-SO em relação aos outros trabalhos realizados é a criação de uma interface mais simples de fácil entendimento dos usuários com apresentação de relatórios e visualização gráfica dos passos dos algoritmos tratados com apresentação de texto teórico explicativo. Na figura 8, pode-se visualizar a simulação da gerência de memória do algoritmo Best-Fit:

Figura 8 – Tela Inicial da Política de Gerência de Memória Best-Fit



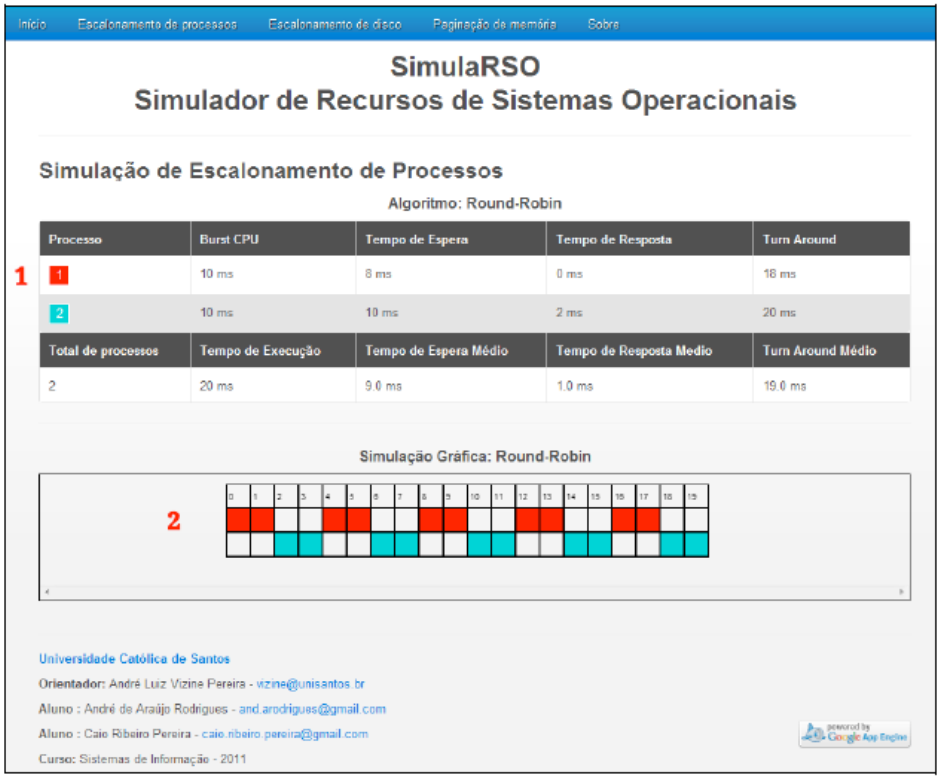
Fonte: REIS, 2009, p.80.

4.6 SimulaRSO – Simulador de Recursos de Sistemas Operacionais

SIMULA RSO desenvolvido por Araujo (2011) é um simulador genérico que realiza simulação de escalonamento de processos, escalonamento de requisições de disco e paginação de memória virtual. A ferramenta possui como diferenciais as funcionalidades para realização de comparações entre os algoritmos e suporte multiidioma. O autor deixa de usar applets devido a sua descontinuidade pela empresa Oracle, e usa ferramentas mais atuais para o desenvolvimento da ferramenta.

O simulador utiliza das seguintes tecnologias: HTML5, CSS3 e Javascript para desenvolver uma interface usuário prática e interativa com a exibição de gráficos. O código do projeto foi disponibilizado no github para outros colaboradores possam acessar o código fonte, tendo a possibilidade de agregar novas funcionalidades e novos módulos de simulação. A arquitetura definida no projeto baseia-se em Java, CSS3 e JavaScript, sendo obrigatório o conhecimento dessas tecnologias para alteração desses códigos. O acesso ao site não está disponível, pois a URL da página expirou no servidor, segue abaixo a figura do resultado do escalonamento de processos extraídas do artigo:

Figura 9 – Resultado de um escalonamento de processos



Fonte: ARAUJO, 2011, p.44.

4.7 Simulador de Rotinas do Sistema Operacional para Auxílio às Aulas Teóricas

O simulador foi construído baseado em uma análise dos simuladores utilizados para o aprendizado de Sistemas Operacionais. Feita essa análise, foi constatado que nenhum dos simuladores disponibilizados trabalha com mais de um núcleo ou mais de um processador. Foi incorporada essa especialidade no projeto, e adicionada a técnica de substituição de páginas que também não foram abordados por nenhum dos trabalhos analisados pelos autores.

Essa ferramenta em questão preocupou-se em enriquecer a interface com uma biblioteca gráfica recente, aperfeiçoando a exibição das gerências de memória e processos. Foi desenvolvido dois simuladores independentes um responsável pela gerência de memória e outro para a gerência de processador e ainda não é integrado. Foi utilizada a linguagem Java orientado a objetos com ferramentas encontradas na biblioteca OpenGL para auxiliar na implementação dos recursos gráficos (RIBEIRO; BERNARDES; LOBO, 2014).

Na figura 10 pode-se visualizar a simulação de processosde tempo compartilhado:

Figura 10 – Gerência de Processos: Simulação do Algoritmo Tempo Compartilhado

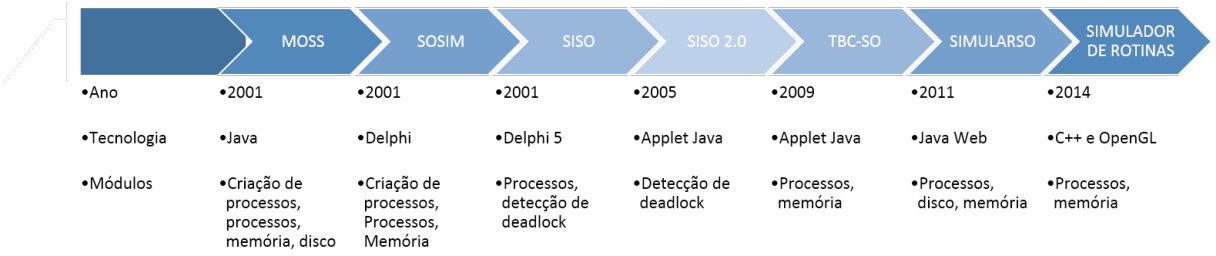
Fonte: RIBEIRO; BERNARDES; LOBO, 2014, p.745.

4.8 Análise dos simuladores apresentados

A criação de simuladores e o aperfeiçoamento dos softwares existentes tem sido recorrentes desde o trabalho de Tanenbaum (2001) que serviu como um gatilho para o desenvolvimento de novas ferramentas com o objetivo de simplificar e aperfeiçoar o ensino e o aprendizado de Sistemas Operacionais.

Pode-se visualizar que a mudança mais significativa dessas ferramentas são as interfaces, no qual, a evolução de linguagens de programação e bibliotecas gráficas possibilitou a melhoria de apresentação dos módulos. Observa-se que a gerência de processos é implementada na maioria dos softwares tendo mudança apenas da linguagem de programação e adição de novas funcionalidades existentes na literatura de Sistemas Operacionais. A figura a seguir resume a evolução dos simuladores apresentados neste capítulo.

Figura 11 – Linha do Tempo dos Simuladores de Sistemas Operacionais



Fonte: Elaborado pelo próprio autor

Pode-se concluir com os trabalhos de simuladores apresentados acima:

- A cada simulador criado houve um aperfeiçoamento da camada de apresentação para melhorar a interatividade do usuário com a simulação, disponibilizando gráficos mais detalhados. Pode-se visualizar que com o passar dos anos a interface evoluiu juntamente com as tecnologias;
- Mudança de plataforma em todas as situações, não tendo reaproveitamento dos módulos que foram criados. Essa mudança de plataforma deve-se ao intervalo de tempo que os projetos foram criados, onde tecnologias utilizadas em outros simuladores se tornaram obsoletas e com isso foram descontinuados;
- Os algoritmos são iguais em todos simuladores, alterando somente a linguagem de programação. Os conceitos de Sistemas Operacionais envolvendo módulos de processos, memória e discos não sofrem mudanças de uma literatura pra outra.
- A integração dos módulos dos Sistemas Operacionais foi classificada com grau de alta complexidade, devido ao grande trabalho de reimplementar os algoritmos, deixando essa junção dos componentes de SO como uma atualização de outra versão da simulação.
- Com a constante inovação tecnológica e novas ferramentas disponibilizadas não houve a preocupação de desenvolvimento para que todo o conteúdo apresentado pudesse ser utilizado em outras plataformas.

5 Especificação da API

5.1 Requisitos Funcionais

As APIs permitem que o desenvolvedor possa utilizar funcionalidades de aplicações existentes no desenvolvimento do seu projeto. Também facilita a comunicação, integração e a interoperação entre aplicações. A grande vantagem da API é que ela permite facilitar autilização de bibliotecas externas no software. Cada programador pode desenvolver seu código da maneira que quiser. Porém, quando um padrão é convencionado, o desenvolvimento é simplificado, certo? Assim, é possível prever exatamente o que as funções irão fazer (PIRES, 2017).

5.1.1 Modelo Arquitetural REST

A opção pelo modelo arquitetural REST para criar a API do simulador proposto, deve-se às seguintes justificativas apresentadas por Maciel (2014):

- Por ser menos burocrático e exigir menos formalismo para implementação;
- Devido a sua simplicidade e o crescimento do interesse pela comunidade para o uso na web;
- Pela relevância do interesse do termo na Web;
- Os Web services com REST são mais manuteníveis do lado do servidor.

As representações foram modeladas em JSON, pois sua leitura e escrita é simples. O cliente estabelece uma conexão com o servidor de aplicação Web, submete seu pedido, por meio de mensagem HTTP e mantém aberto o canal de comunicação até que o servidor envie uma resposta. Se o cliente não estiver interessado em obter resposta do servidor, essa comunicação pode ser encerrada pelo cliente a qualquer momento.

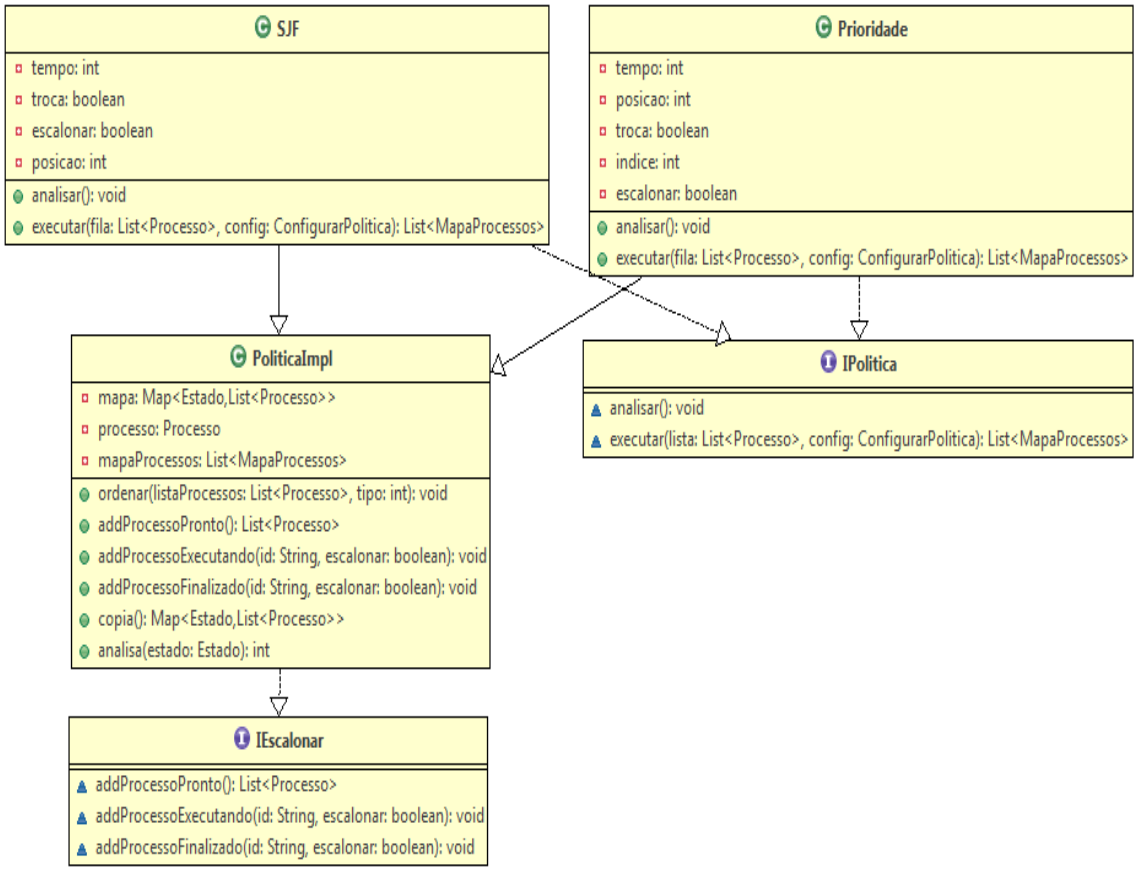
> O cliente estabelece uma conexão com o servidor de aplicaçãoWeb, submete seu pedido, por meio de mensagem HTTP e mantém aberto o canal de comunicação até que o servidor envie uma resposta. Se o cliente não estiver interessado em obter resposta doservidor, essa comunicação pode ser encerrada pelo cliente a qualquer momento (MACIEL, 2014).

5.1.2 Padrões de projeto

Para a criação de uma estrutura que seja utilizada e modificada por um número de pessoas indefinidas, é de grande importância a preocupação com a codificação. Os padrões de projeto, também conhecidos pelo termo original em inglês design patterns, descrevem soluções para problemas recorrentes no desenvolvimento de software, e quando utilizados de forma correta, refletem diretamente no aumento da qualidade do código, tornando-o mais flexível, elegante e reusável. Na arquitetura especificada neste projeto foi utilizado o padrão de projeto Strategy e Abstract Factory (GUERRA, 2013).

O padrão Strategy permite configurar uma classe com um de vários comportamentos, utilizando o conceito de OO chamado de composição.Permite que algoritmos variem independentemente entre clientes que os utilizam. Quando se necessita de um algoritmo que trata de modos diferentes os dados submetidos a ele. A figura abaixo mostra o diagrama de classes da parte teórica da APISIM responsável por gerenciar os processos.

Figura 12 – Padrão Strategy aplicado no Simulador



Fonte: Elaborado pelo próprio autor.

5.2 Implementação dos Recursos

5.2.1 Algoritmos de Escalonamento

Desenvolveu-se API do simulador de Sistemas operacionais responsável pela gerência de processos. Os algoritmos selecionados para a simulação foram: FIFO, SJF e Prioridade. Os recursos foram modelados para serem representações dos processos e do escalonador, com objetivo de gerar o resultado do escalonamento realizado pelo cliente. Ba tabela 3, pode-se visualizar uma relação entre o conteúdo téorico extraído de Sistemas Operacionais e o que foi utilizado para o seu desenvolvimento.

Tabela 3 – Algoritmos de Escalonamento X Implementação

Referencial Teórico	Implementação dos algoritmos		
	Processo	Escalonador	Algoritmos de Escalonamento
Processos	x		
Escalonamento de Processos		x	
FIFO			x
SJF			x
Circular			
Circular com prioridade			
Prioridade			x

Fonte: Elaborado pelo próprio autor.

5.2.2 Recursos

Os recursos disponibilizados pelo simulador seguem os padrões convencionados pela arquitetura REST. O que permite o usuário realizar as operações necessárias para realizar o escalonamento de processos. As operações utilizadas no simulador foram: POST, GET, DELETE e PUT. Na figura abaixo, visualiza-se os recursos disponíveis no simulador mapeadas através do Swagger:

Figura 13 – Recursos do Simulador

Fonte: Elaborado pelo próprio autor.

5.2.2.1 Recurso Política

O recurso política é utilizado para catalogar todas as políticas disponíveis do gerenciamento de processos. Na tabela 4, são apresentadas as URIs e métodos do HTTP suportados no recurso política.

Tabela 4 – Recurso Política

Métodos Recurso Política				
URIs canônicas do recurso	PUT	GET	DELETE	POST
/politica		X		

Fonte: Elaborado pelo Autor.

Na figura abaixo, pode-se visualizar a documentação do Recurso Política relativa ao método GET:

Figura 14 – Recurso Política - Descrição Método GET

Curl

```
curl -X GET --header 'Accept: application/json' 'http://localhost:8080/politicas'
```

Request URL

```
http://localhost:8080/politicas
```

Request Headers

```
{
  "Accept": "*/*"
}
```

Response Body

```
[
  "FIFO",
  "SJF",
  "PRIORIDADE",
  "CIRCULAR"
]
```

Response Code

```
200
```

Response Headers

```
{
  "date": "Mon, 02 Oct 2017 21:42:24 GMT",
  "transfer-encoding": "chunked",
  "x-application-context": "application",
  "content-type": "application/json;charset=UTF-8"
}
```

Fonte: Elaborado pelo próprio autor.

5.2.2.2 Recurso Processo

O recurso processo é utilizado para armazenar os processos que serão gerenciados pelo escalonador. Na Tabela abaixo, são apresentadas as URIs e métodos do HTTP suportados no recurso Processo.

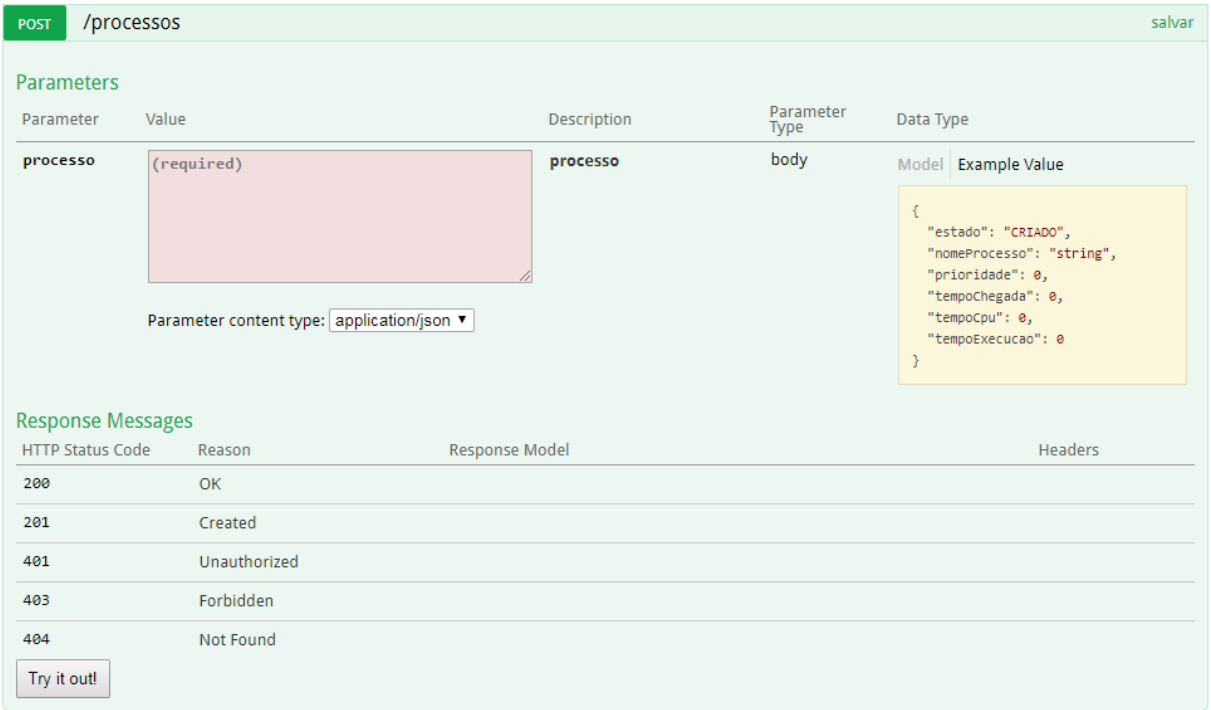
Tabela 5 – Recurso Processo

Métodos Recurso Processo				
URIs canônicas do recurso	PUT	GET	DELETE	POST
/processo	X	X		
/processo/id			X	X

Fonte: Elaborado pelo Autor.

Na figura abaixo, pode-se visualizar o método POST:

Figura 15 – Recurso Processo - Descrição Método POST



Fonte: Elaborado pelo próprio autor.

5.2.2.3 Recurso Escalonador

O recurso escalonador é responsável por escalonar os processo, de acordo com as entradas realizadas pelo cliente. Sendo necessária a criação dos processos e definição da política. Na tabela abaixo, são apresentadas as URIs e métodos do HTTP suportados no recurso Escalonador.

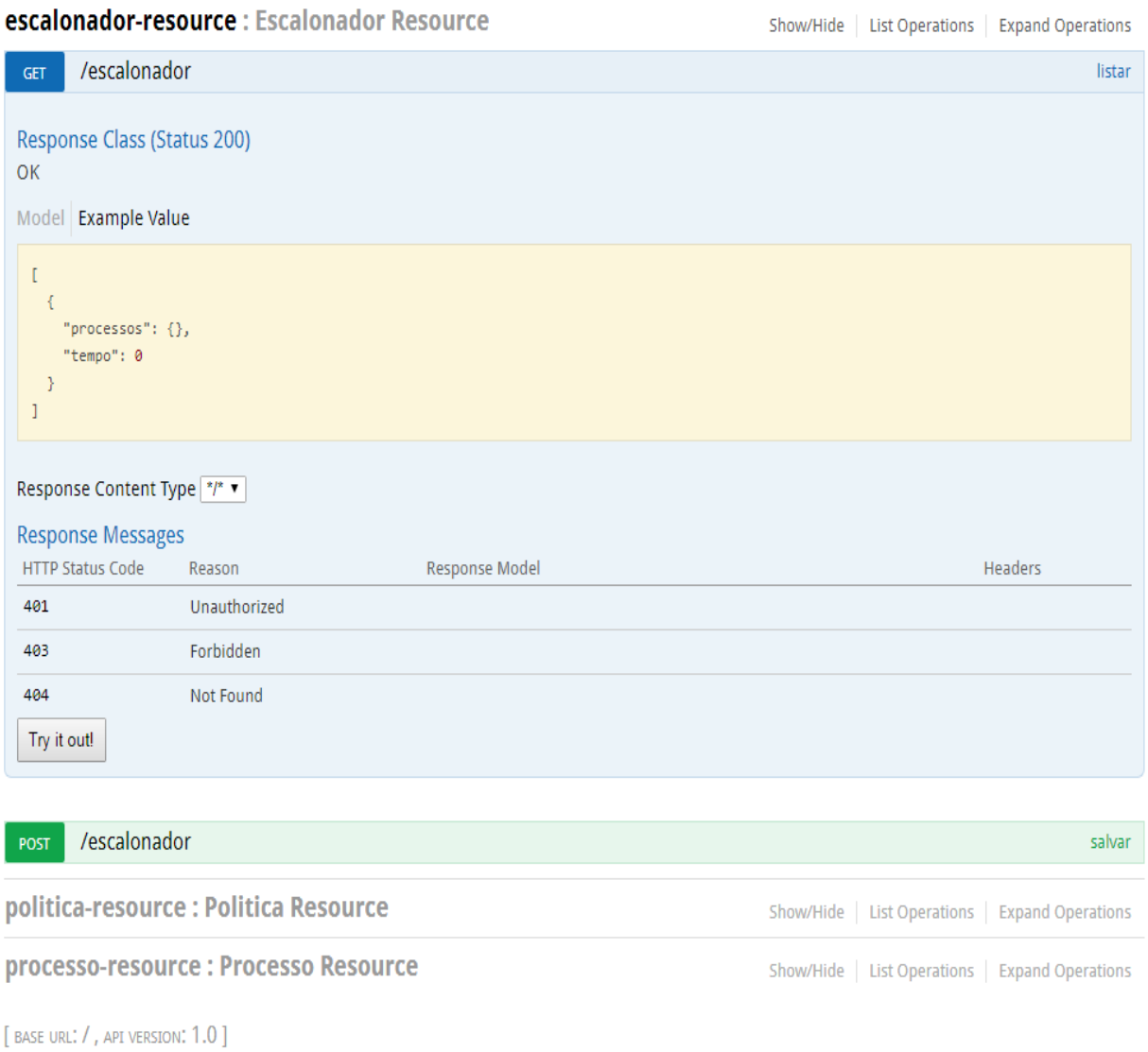
Tabela 6 – Recurso Escalonador

Métodos Recurso Escalonador				
URIs canônicas do recurso	PUT	GET	DELETE	POST
/escalonador	X	X		

Fonte: Elaborado pelo Autor.

Na figura abaixo, pode-se visualizar a documentação do Recurso Escalonador relativa ao método GET:

Figura 16 – Recurso Escalonador - Descrição Método GET



Fonte: Elaborado pelo próprio autor.

6 Avaliação da API

Com o objetivo de consumir os serviços disponibilizados pelo simulador, foi utilizado o software Postman para a realização da entrada dos dados que são mostrados nos exercícios presentes neste capítulo.

A entrada é realizada de acordo com a documentação da API RESTful apresentada no capítulo anterior, no qual descreve todos os métodos, tipos de entrada e as saídas relativas aos dados que foram escritos. Sendo assim, testou-se os serviços sem a criaçao de um cliente para consumir o que foi disponibilizado pela APi, no qual o objetivo é obter o json com a resposta do escalonador com o resultado da gêrencia dos processos.

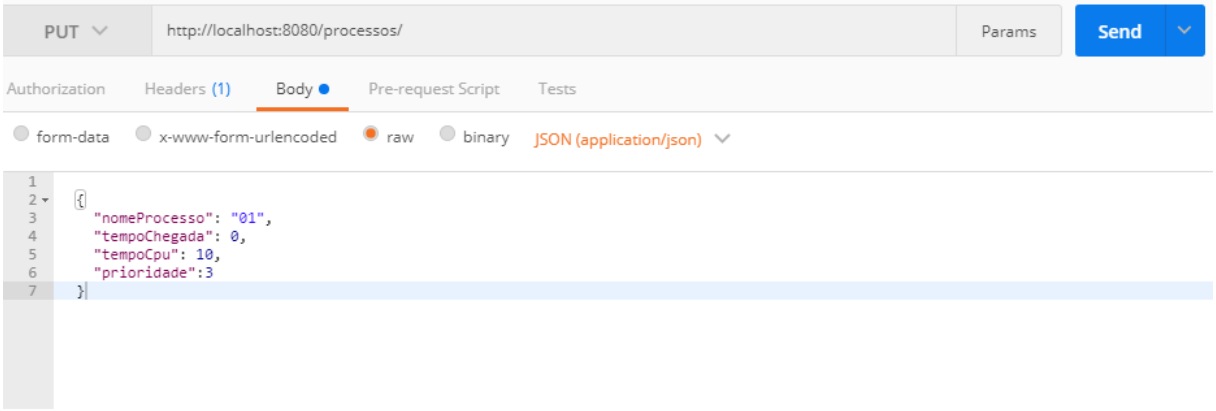
6.1 Validação do serviço

O problema básico de escalonamento em sistemas operacionais é como satisfazer simultaneamente objetivos conflitantes: tempo de resposta rápido, bom throughput para processos background, evitar starvation, conciliar processos de alta prioridade com de baixa prioridade. O conjunto de regras utilizado para determinar como, quando e qual processo deverá ser executado é conhecido como política de escalonamento (MEIRA, 2008).

Com intuito de aperfeiçoar o aprendizado para satisfazer os objetivos citados por Meira (2008) e compreender como é o funcionamento do escalonamento de processos em Sistemas Operacionais, foram criados ou adaptados exercícios teóricos para realizar o teste do serviço referente à gerência de processos. O serviço possibilita o cliente de através dos resultados analisar visualmente, criar sua forma de visualização de acordo com o JSON gerado e identificar quais são os melhores critérios de escalonamento.

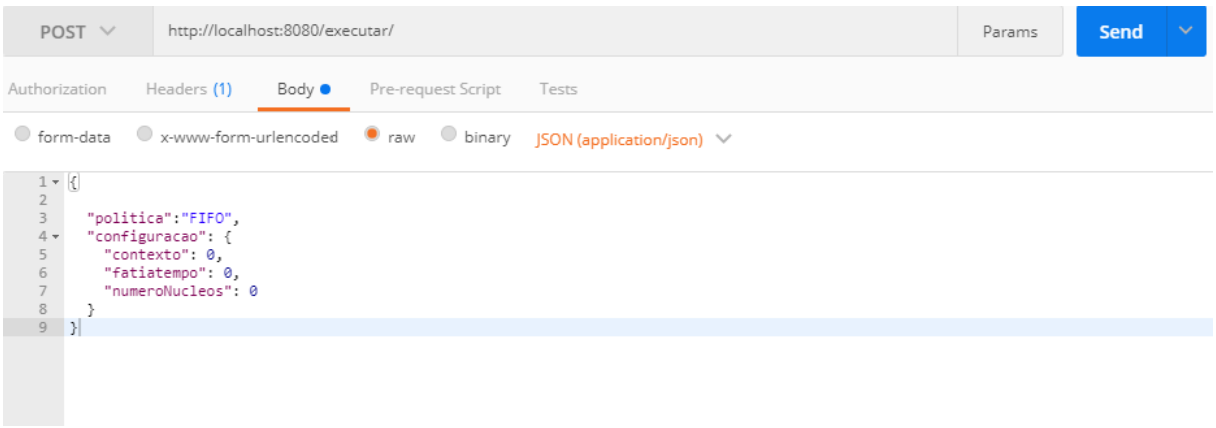
Para validar o serviço de gerência de processos realizou-se testes com a utilização de entradas de exercícios que serão apresentados nessa sessão. O objetivo dessa sessão é mostrar que é possível que o cliente visualize o resultado independente do tipo de cliente que é utilizado. Nesse caso, nenhum cliente foi construído para a realização dos testes. As figuras a seguir exibem como são as entradas de dados da criação de processos, definição de configurações do escalonador, e a saída do resultado do escalonador.

Figura 17 – Criação de Processos



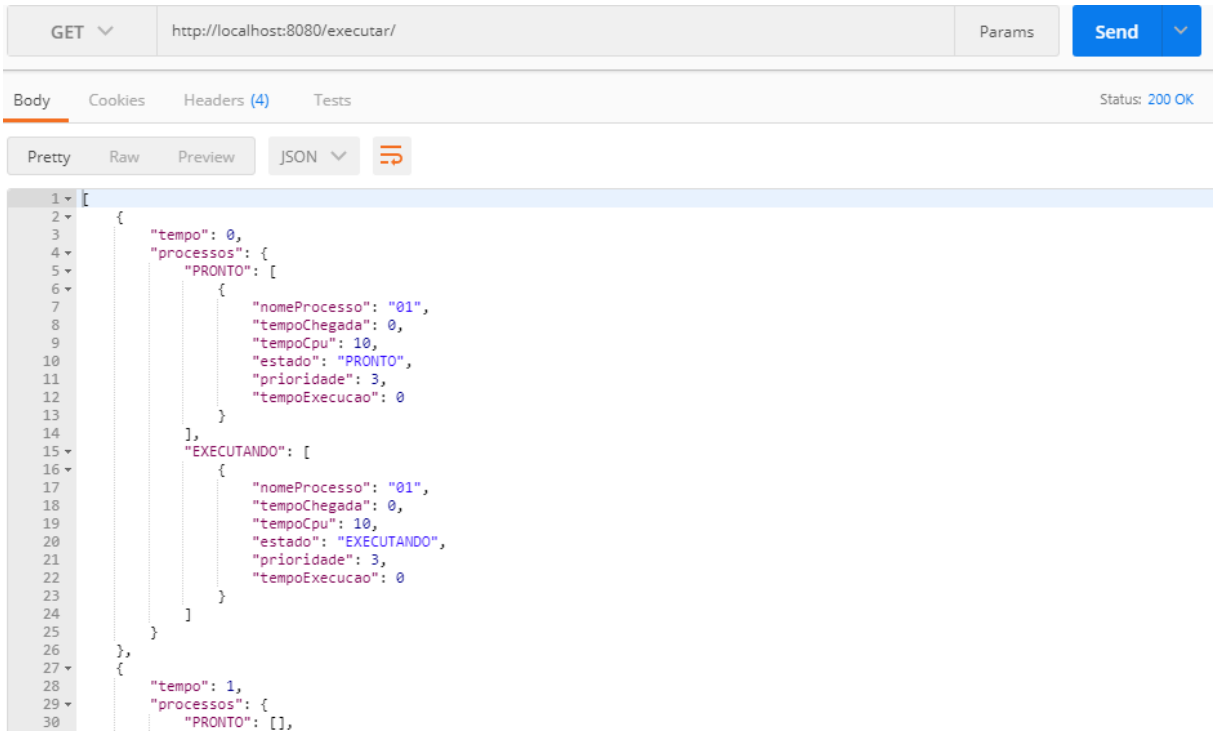
Fonte: Elaborado pelo próprio autor

Figura 18 – Definição de configurações do Escalonador



Fonte: Elaborado pelo próprio autor

Figura 19 – Resultado do Escalonamento



Fonte: Elaborado pelo próprio autor

O resultado de um escalonamento simples mostrado na figura 19 gera um json extenso, por causa da estrutura da saída que exibe o estado de cada processo a cada unidade de tempo no processador. Devido a essa pecularidade do resultado, para mostrar os resultados do escalonador optou-se por usar o Excel para ilustrar o escalonamento de processos. Os diagramas foram feitos manualmente sem nenhum tipo de ferramenta para ler o json, somente foi analisado o estado de cada processo na unidade de tempo e colocado no gráfico.

6.1.1 Testes

6.1.1.1 Exercício 1

(MACHADO; MAIA, 2007) Considere que cinco processos sejam criados no instante de tempo 0 (P1, P2, P3, P4 e P5) e possuam as características descritas na tabela a seguir:

Tabela 7 – Exercício 1

Processo	Tempo de UCP	Prioridade
P1	10	3
P2	14	4
P3	5	1
P4	7	2
P5	20	5

Fonte: Elaborado pelo próprio autor

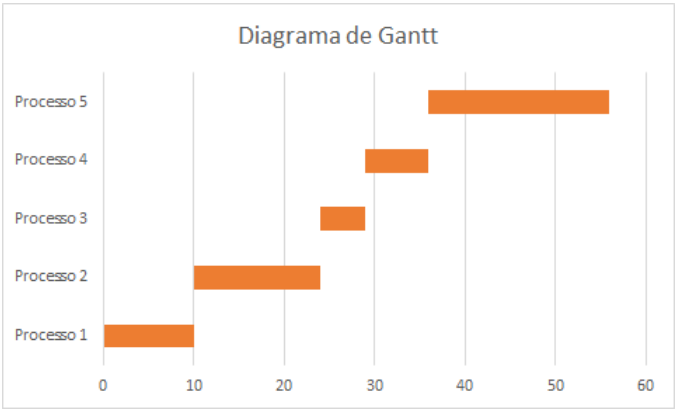
Desenhe um diagrama ilustrando o escalonamento dos processos e seus respectivos tempos de turnaround, segundo as políticas especificadas a seguir. O tempo de troca de contexto deve ser desconsiderado.

a) FIFO

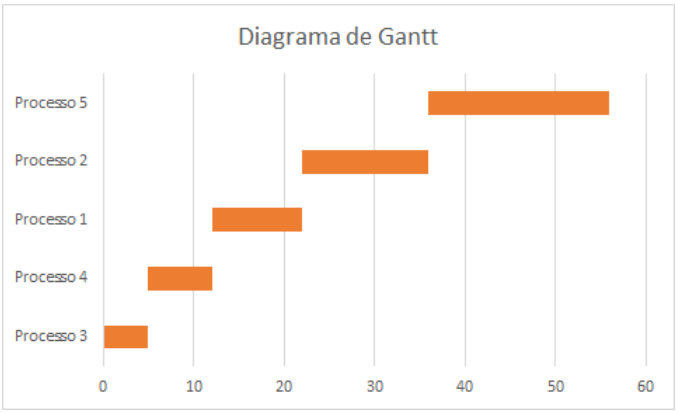
b) SJF

c) Prioridade (número menor implica prioridade maior)

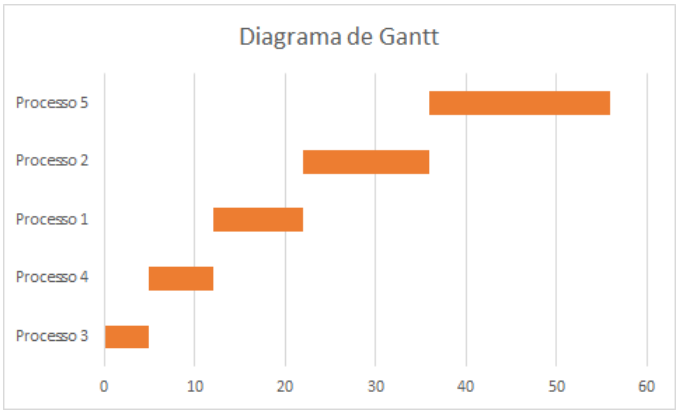
Figura 20 – Resultado Exercício 1



(a) Escalonamento algoritmo FIFO



(b) Escalonamento algoritmo SJF



(c) Escalonamento algoritmo Prioridade

Fonte: Elaborado pelo próprio autor

6.1.1.2 Exercício 2

(MACHADO; MAIA, 2007) Considere a tabela a seguir onde:

Tabela 8 – Exercício 2

Processo	Tempo de UCP	Prioridade
P1	40	4
P2	20	3
P3	50	1
P4	30	3

Fonte: Elaborado pelo próprio autor

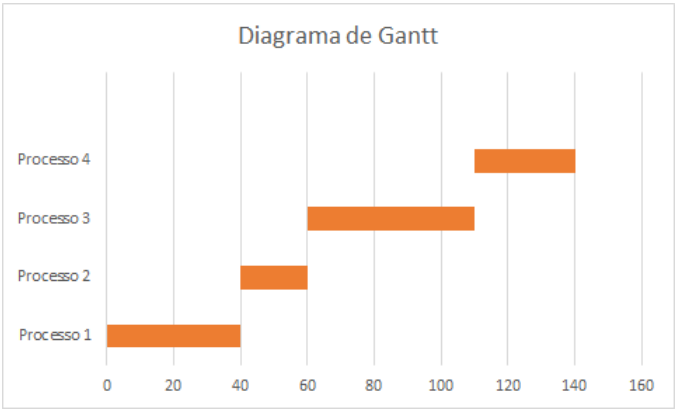
Qual o tempo de turnaround médio dos processos, considerando o tempo de troca de contexto igual a 0 e a 5 u.t para os seguintes escalonamentos:

a) FIFO

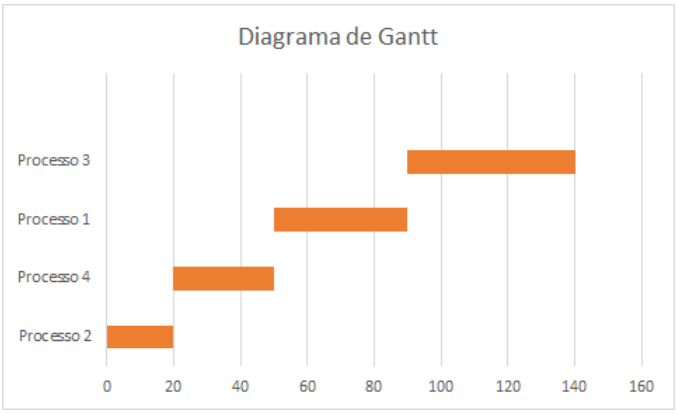
b) SJF

c) Prioridade (número menor implica prioridade maior)

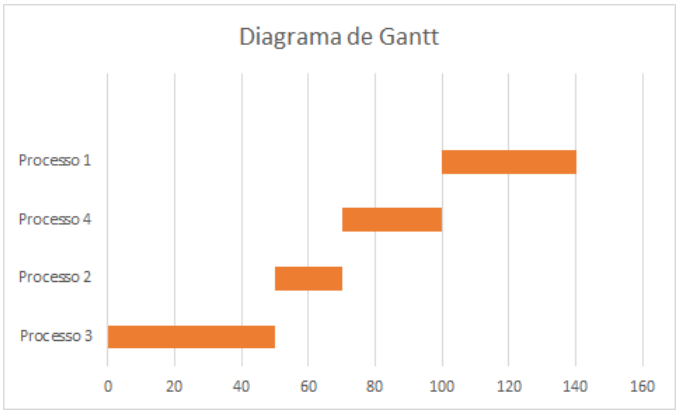
Figura 21 – Resultado Exercício 2



(a) escalonamento algoritmo FIFO



(b) escalonamento algoritmo SJF.



(c) escalonamento algoritmo Prioridade

Fonte: Elaborado pelo próprio autor

6.1.1.3 Exercício 3

Cinco processos são criados na seguinte ordem: P1 , P2 , P3 , P4 e P5, com os seguintes tempos:

Tabela 9 – Exercício 3

Processo	Tempo de Chegada	Tempo de UCP	Prioridade
P1	0	13	3
P2	4	11	4
P3	5	7	1
P4	7	8	2
P5	10	16	5

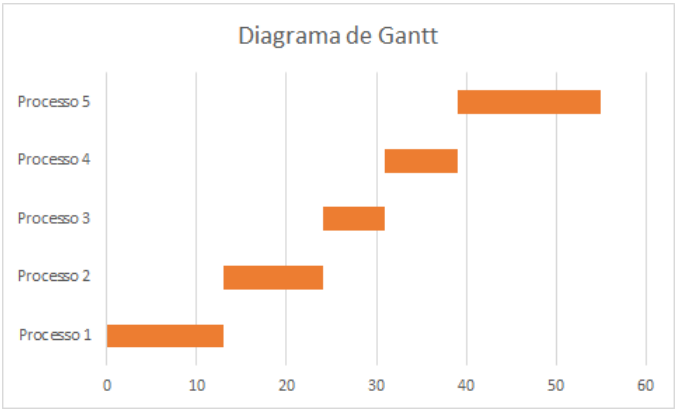
Fonte: Adaptado pelo próprio autor

a) FIFO

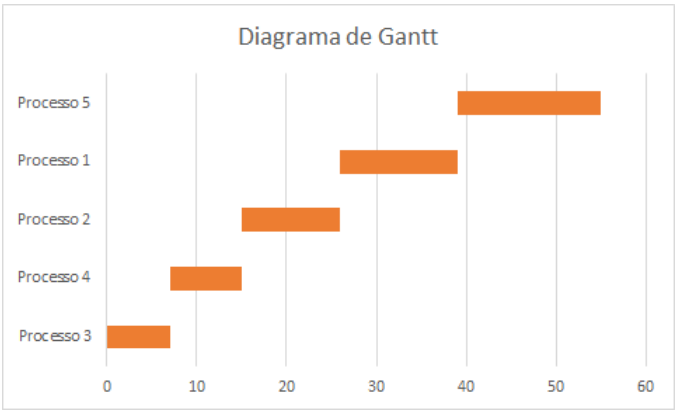
b) SJF

C) PRIORIDADE

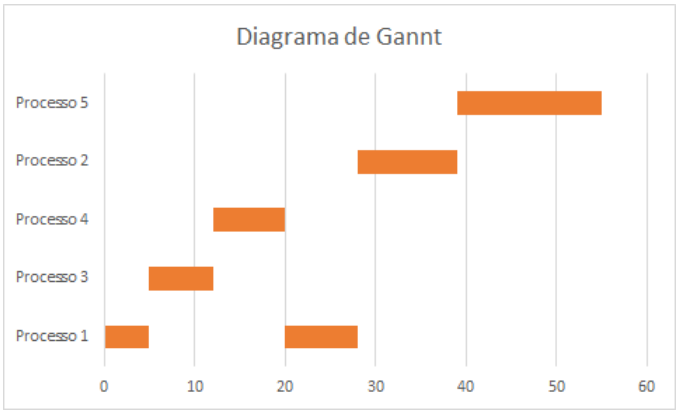
Figura 22 – Resultado Exercício 3



(a) escalonamento algoritmo FIFO



(b) escalonamento algoritmo SJF



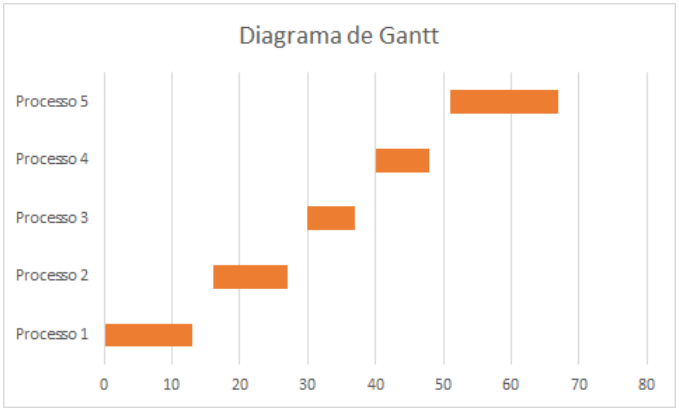
(c) escalonamento algoritmo Prioridade

Fonte: Elaborado pelo próprio autor

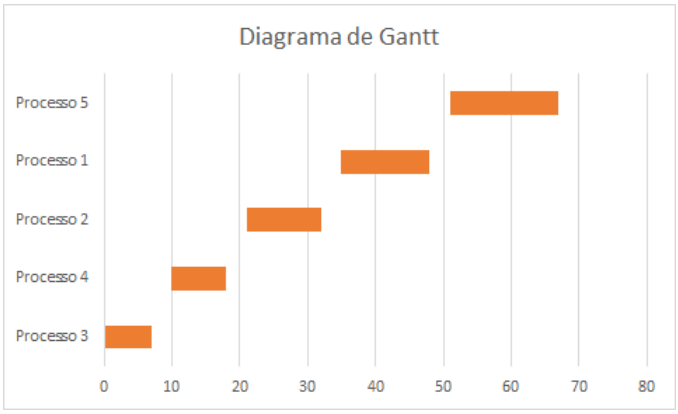
6.1.1.4 Exercício 4

De acordo com a tabela do exercício 3 desenhe quatro gráficos que ilustrem a execução desses processos usando FCFS, SJF, prioridade preemptiva (um número de prioridade menor significa uma prioridade mais alta) com troca de contexto = 3.

Figura 23 – Resultado Exercício 4



(a) escalonamento algoritmo FIFO.



(b) escalonamento algoritmo SJF

Fonte: Elaborado pelo próprio autor

6.2 Resultados

Pode-se observar de acordo com os diagramas dos exercícios apresentados, que as saídas dos algoritmos foram condizentes com o resultado que se espera de cada escalonamento satisfazendo então o gerenciamento de processos.

O resultado foi gerado independentemente de uma interface gráfica ou de qualquer linguagem de programação, sendo possível o escalonamento sem a necessidade de criação de códigos e especificação de uma arquitetura. Com a validação dessa etapa, a próxima fase é a criação de uma interface gráfica com uma linguagem definida, visando testar a aplicação para a conexão dela com outra linguagem/plataforma.

7 Estudos de caso

Segundo PIRES (2017) a API trata-se de um conjunto de rotinas e padrões estabelecidos e documentados por uma aplicação A, para que outras aplicações consigam utilizar as funcionalidades desta aplicação A, sem precisar conhecer detalhes da implementação do software.

7.1 Testes

Para a realização dos testes, seguiram-se os seguintes passos para validar os serviços oferecidos pela API:

- Criação dos processos para o escalonamento;
- Verificar as políticas existentes;
- Selecionar a política para escalonar os processos;
- Executar o escalonamento de processos;
- Receber o resultado do escalonamento;
- Visualizar o resultado graficamente;

7.1.1 Cliente Desktop

Criou-se um cliente desktop para realizar o primeiro teste da api. Para a verificação dos itens apresentados na lista anterior, foi desenvolvido uma aplicação em Java para consumir os dados da API. Segue abaixo uma figura do acesso a URI através do cliente:

Figura 24 – Acesso à URI do Escalonador

```
public MapaProcessos[] listar() {

    RestTemplate restTemplate = new RestTemplate();

    RequestEntity<Void> request = RequestEntity

            .get(URI.create("http://localhost:8080/escalonador"))

            .header("Authorization", "Basic YWxnYXdvcmtzOnMzbmg0").build();

    ResponseEntity<MapaProcessos[]> response = restTemplate.exchange(request,
            MapaProcessos[].class);

    mapa = Arrays.asList(response.getBody());
    return response.getBody();

}
```

Fonte: Elaborado pelo Autor

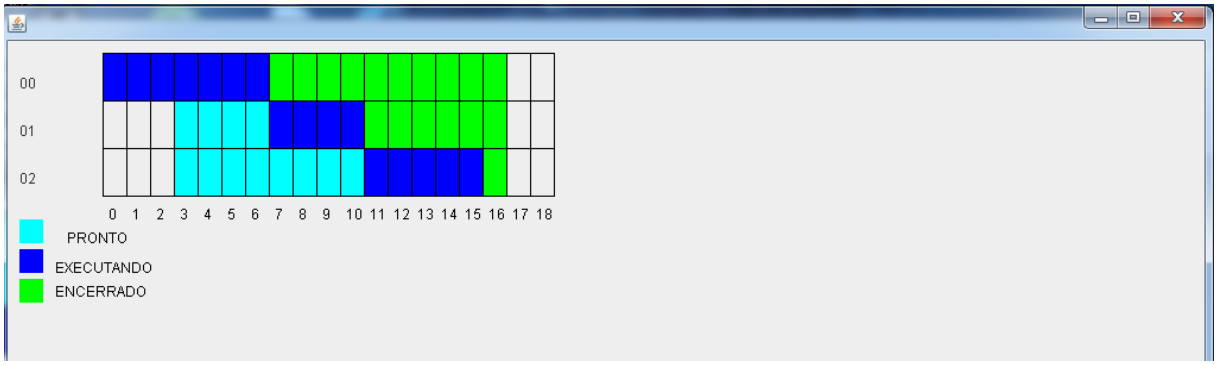
As respostas às requisições do cliente ficam armazenadas nos objetos da aplicação e podem ser acessadas através da criação dos métodos, definindo qual tipo de operação a aplicação irá utilizar. A tabela 10 apresenta os dados utilizados para a realização do teste e as figuras 25,26 27 exibem os resultados relativos ao escalonamento de acordo com a política selecionada.

Tabela 10 – Dados Escalonamento Desktop

Nome	Tempo de Chegada	Tempo de CPU	Prioridade
00	0	7	0
01	3	4	1
02	3	5	2

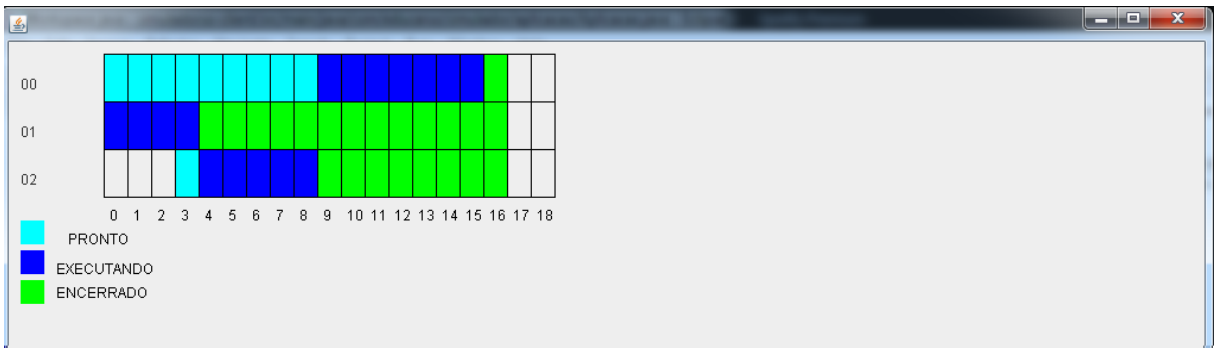
Fonte: Adaptado pelo próprio autor

Figura 25 – Resultado Escalonamento FIFO



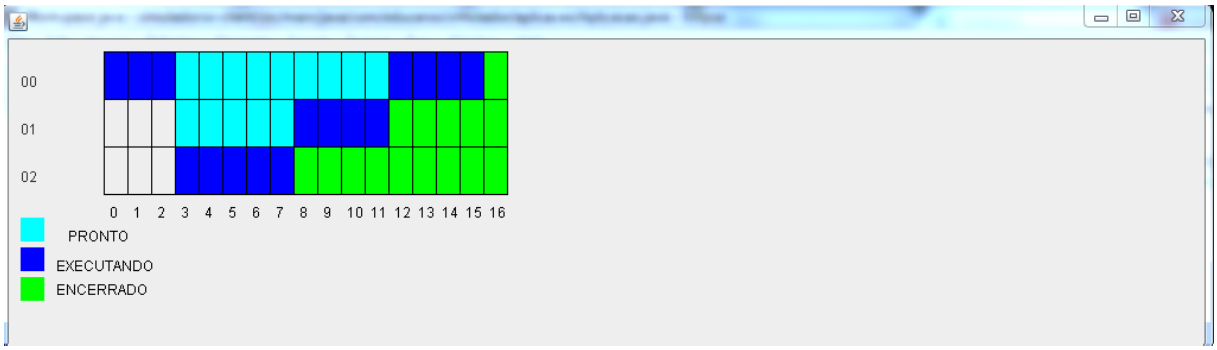
Fonte: Elaborado pelo próprio autor

Figura 26 – Resultado Escalonamento SJF



Fonte: Elaborado pelo próprio autor

Figura 27 – Resultado Escalonamento Prioridade



Fonte: Elaborado pelo próprio autor

7.1.2 Cliente Web

O cliente web foi desenvolvido com a utilização de Ajax, HTML e Javascript. A comunicação REST foi realizada utilizando a arquitetura AJAX. A arquitetura funciona da seguinte forma (RICHARDSON; RUBY, 2008):

- Um usuário, controlando um navegador, realiza uma requisição com a URI da aplicação;
- O servidor serve uma página da Web que contém um script incorporado;
- O navegador processa a página da Web e executa o script ou espera que o usuário desencadeie uma das ações do script com uma operação de teclado ou mouse;
- O script faz uma solicitação HTTP assíncrona para algum URI no servidor. O usuário pode fazer outras coisas enquanto o pedido está sendo feito e provavelmente nem sequer percebe que o pedido está acontecendo;
- O script analisa a resposta HTTP e usa os dados para modificar a exibição do usuário;

Para realizar o teste do servidor, os seguintes dados foram utilizados:

Tabela 11 – Dados Escalonamento Web

Nome	Tempo de Chegada	Tempo de CPU	Prioridade
00	0	7	0
01	3	4	1
02	3	5	2

Fonte: Adaptado pelo próprio autor

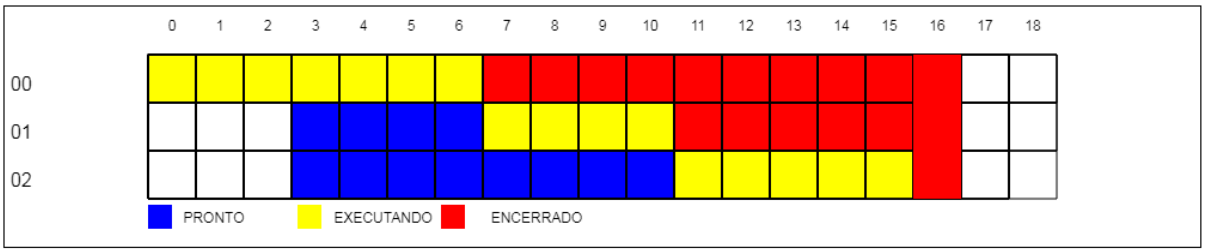
Com a finalização da criação dos processos, pode-se visualizar os processos criados na seguinte figura em formato JSON:

Figura 28 – Método GET Recurso Processo

Fonte: Elaborado pelo próprio autor

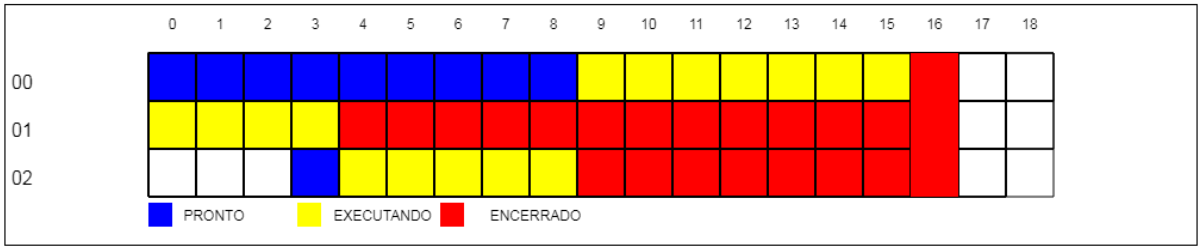
Realizou-se o escalonamento de processos com a seleção da política disponível pela API. As figuras a seguir mostram o resultados da gêrencia de processos dos algoritmos de prioridade, FIFO e SJF:

Figura 29 – Escalonamento FIFO



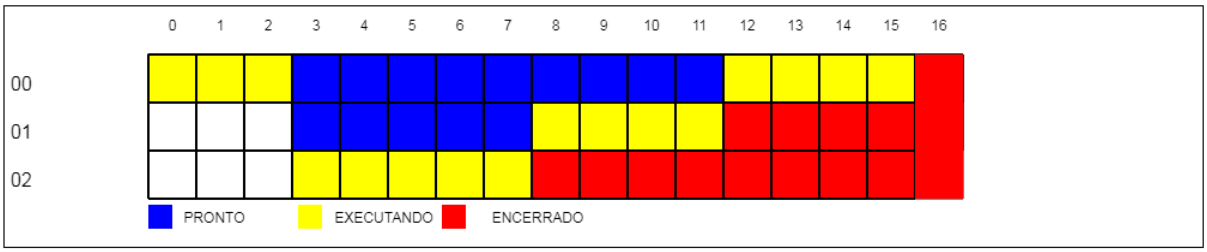
Fonte: Elaborado pelo próprio autor

Figura 30 – Escalonamento SJF



Fonte: Elaborado pelo próprio autor

Figura 31 – Escalonamento Prioridade



Fonte: Elaborado pelo próprio autor

7.2 Resultados

De acordo com o que foi apresentado nesse capítulo, visualiza-se que as entradas dos dados foram realizadas em sistemas diferentes e o escalonamento foi realizado pela API. Cada sistema enviou os processos e a API retornou o resultado do escalonador. Sendo assim, a resposta enviada tanto para o cliente desktop quanto para o sistema é padronizada, o que facilita a mudança de tecnologia e a criação de novas interfaces. É possível mediante o que foi desenvolvido, evoluir interfaces e funcionalidades, pois o cliente não tem necessidade de conhecer as funcionalidades do sistema, somente saber o que está sendo disponibilizado para ser acessado através dos recursos e métodos. A criação de novas gerências e funcionalidades pode ser feita através da criação de novos recursos e métodos em novas versões da API, reduzindo o trabalho de implementar novamente todo o conteúdo já existente.

Sendo assim, a APISIM possui as seguintes características:

- Simplicidade: Qualquer usuário visualiza as funcionalidades do sistema sem a necessidade de conhecer o código e podem ser utilizados através da URI;
- Interoperabilidade: Devido à padronização dos dados dos recursos, o sistema comunica com qualquer tipo de aplicação. Sendo assim, o usuário consome os seus serviços independentemente da plataforma;
- Incremental: Novas funcionalidades podem ser adicionadas à API sem prejudicar a versão atual, com a geração de novas versões.

8 Conclusão

A partir do estudo bibliográfico realizado a respeito de simuladores de sistemas operacionais, o trabalho em questão foi elaborado com a proposta de desenvolver uma API de gerência de recursos de SO. Foi realizada uma pesquisa de ferramentas na internet para este fim e o modelo arquitetural Rest foi designado para ser aplicado no simulador. Permitindo assim, que o objetivo geral do trabalho fosse alcançado. Com o uso desse modelo, foi possível disponibilizar o serviço de gerência de processos. Deste modo, foi criado uma API que escalona processos e pode ser utilizada sem nenhuma restrição de arquitetura.

De forma geral, a API de simulação de sistemas operacionais, além de ser uma ferramenta de aprendizado, permite o usuário que crie sua própria ferramenta a partir dos recursos da versão atual. Sendo assim, as principais contribuições deste trabalho são:

- Disponibilização da APISIM[1] com o serviço de gerência de processos, que pode ser utilizada para:
  - Aprendizado: Utilização de seus recursos para o escalonamento de processos com foco no aprendizado;
  - Evolução de módulos de Sistemas Operacionais: novos recursos podem ser adicionados pelo usuário.

Devido ao conteúdo extenso de Sistemas Operacionais e à conexão entre as gerências, podem ser exploradas do presente trabalho em futuros estudos os módulos que não foram contemplados neste trabalho. Dentre essas, podem ser destacados:

- Adicionar novas gerências de recursos computacionais;
- Conectar os módulos, com intuito de transformá-lo em um simulador genérico;
- Enriquecer a versão atual com novas funcionalidades.

[1] https://github.com/debora-cristina/simuladorso

Referências

ARAUJO, C. R. P. André de. *SimulaRSO – Simulador de Recursos de Sistemas Operacionais*. 2011. Monografia (Bacharel em Sistemas de informação),Universidade Católica de Santos , Santos, Brasil. Citado na página 13.

DIAS, E. *Desmistificando Rest com Java*. [S.l.]: AlgaWorks Softwares, 2016. Citado na página 22.

FIELDING, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. 2000. Tese (Doutorado) — University of California, Irvine, 2000. DISSERTATION submitted in partial satisfaction of the requirements for the degree of DOCTOR OF PHILOSOPHY in Information and Computer Science. Citado na página 22.

GUERRA, E. *Design Patterns com Java*. 71. ed. Vila Mariana,São Paulo: Casa do Código, 2013. Citado na página 35.

KALIN, M. *Java Web Services*. 1. ed. Rio de Janeiro: Alta Books, 2010. Citado nas páginas 21 e 23.

MACHADO, F. B.; MAIA, L. P. *Arquitetura de Sistemas Operacionais*. 4. ed. Rio de Janeiro: LTC Editora, 2007. ISBN ISBN 978-85-216-1548-4. Citado nas páginas 17, 19, 20, 21, 26, 43 e 45.

MACIEL, B. I. F. *Web service RESTful para Manipulação, Catalogação, Publicação na Web e Eventual Manutenção de Dados Abertos Governamentais*. 2014. Tese (Doutorado) — Universidade Federal de Pernambuco, Recife, 2014. Dissertação de Mestrado. Citado na página 34.

MAIA, L. P. *SOsim: SIMULADOR PARA O ENSINO DE SISTEMAS OPERACIONAIS*. 3 2001. Dissertação (Mestrado) — Universidade Federal do Rio de Janeiro, Rio de Janeiro, 3 2001. An optional note. Citado nas páginas 12, 13 e 26.

MARTINS, M. P. do E. S. *Desenvolvimento ergonômico de uma interface gráfica para o software educacional SISO - Simulador de Sistemas Operacionais - Módulo de Detecção de Deadlock*. 2005. Monografia (Especialista em Engenharia de Sistemas), Centro Universitário do Estado do Pará , Belém, Pará. Citado na página 13.

MARTINS, M. P. do E. S. *RESTful Web Services e a API JAX-RS*. 2015. Arquitetura do portal Infoq Brasi. Citado nas páginas 22 e 23.

MAZIERO, C. Reflexões sobre o ensino prático de sistemas operacionais. In: CONGRESSO DA SBC, XXII., 2002, Florianópolis. *Anais do X Workshop sobre Educação em Computação*. [S.l.], 2002. Citado na página 12.

MAZIERO, C. A. *Sistemas Operacionais: Conceitos e Mecanismos*. Curitiba PR: Carlos Alberto Maziero, 2013. Citado nas páginas 18 e 20.

MEIRA, M. V. D. (Ed.). *Campo Digit@l*. 2008. Disponível em: <revista.grupointegrado.br/revista/index.php/campodigital/article>. Citado na página 40.

ONTKO, R.; REEDER, A.; TANENBAUM. *Modern Operating Systems Simulators*. 1. ed. Boston, EUA, 2001. An optional note. Citado nas páginas 13 e 25.

PIRES, J. *O que é API? REST e RESTful? Conheça as definições e diferenças!* 2017. Disponível em: <https://becode.com.br/o-que-e-api-rest-e-restful/>. Citado nas páginas 24 e 34.

REIS, A. A. dos. *BOAS PRÁTICAS PARA DESENVOLVIMENTO DE APIS REST*. 2016. Disponível em: <http://www.matera.com/br/2016/01/21/boas-praticas-para-desenvolvimento-de-apis-rest/>. Citado na página 24.

REIS, F. P. *TBC-SO/WEB: SOFTWARE EDUCATIVO PARA APRENDIZAGEM DE GERÊNCIA DE PROCESSOS E DE GERÊNCIA DE MEMÓRIA EM SISTEMAS OPERACIONAIS*. 2009. Monografia (Bacharel em Ciência da Computação), Universidade Federal de Lavras , Lavras, Minas Gerais. Citado nas páginas 13 e 29.

RIBEIRO, T. P.; BERNARDES, R. L.; LOBO, E. A. Simulador de rotinas do sistema operacional para auxílio às aulas teóricas. In: SIMPóSIO BRASILEIRO DE SISTEMAS DE INFORMAçãO, X., 2014, Londrina. *Anais do SBSI 2014*. [S.l.], 2014. Citado nas páginas 13 e 31.

RICHARDSON, L.; RUBY, S. *RESTful Webservices*. 1. ed. California, EUA: O'Reilly Media, 2008. ISBN 978-0-596-52926-0. Citado na página 52.

SBC. *Currículo de Referência da Sociedade Brasileira de Computação para Cursos de Graduação em Computação e Informática*. 2003. Disponível em: <http://www.sbc.org.br/documentos-da-sbc/send/131-curriculos-de-referencia/761-diretrizes-curriculares-consulta-publica>. Citado na página 12.

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. *Sistemas Operacionais com Java*. [S.l.]: Elsevier, 2008. Citado nas páginas 18, 20 e 21.

TANENBAUM, A. S. *Modern Operating System*. 2. ed. Upper Saddle River, New Jersey: Prentice-Hall, 2001. ISBN 0-13-000000-0. Citado nas páginas 13, 20 e 21.

TANENBAUM, A. S.; WOODHULL, A. S. *Sistemas operacionais: Projeto e implementação*. 2. ed. [S.l.]: Bookman. ISBN 85-7307-530-9. Citado nas páginas 13 e 17.

TEIXEIRA, J. de F. *UMA PROPOSTA DE SOFTWARE EDUCACIONAL SIMULADOR PARA ENSINO DE SISTEMAS OPERACIONAIS*. 9 2001. 155 p. Dissertação (Mestre em Ciência da Computação) — Universidade Federal de Santa Catarina, Florianópolis, 9 2001. An optional note. Citado na página 13.