Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

MARCOS ROBERTO ALVES MEDEIROS

UM SIMULADOR PARA SISTEMA OPERACIONAL COM ÊNFASE NO ESCALONAMENTO DE PROCESSOS E GERÊNCIA DE MEMÓRIA VIRTUAL

Assis – SP

2015

MARCOS ROBERTO ALVES MEDEIROS

UM SIMULADOR PARA SISTEMA OPERACIONAL COM ÊNFASE NO ESCALONAMENTO DE PROCESSOS E GERÊNCIA DE MEMÓRIA VIRTUAL

Trabalho de conclusão de curso apresentado ao Instituto Municipal de Ensino Superior de Assis, como requisito no Curso de Bacharelado em Ciência da Computação.

Orientador: Me. Douglas Sanches da Cunha.

Área de Concentração: Ciência da Computação.

Assis – SP

2015

FICHA CATALOGRÁFICA

MEDEIROS, Marcos Roberto Alves.

Um simulador para sistema operacional com ênfase no escalonamento de processos e gerência de memória virtual. / Marcos Roberto Alves Medeiros / Fundação Educacional do Município de Assis – FEMA – Assis, 2015.

82 pág.

Orientador: Me. Douglas Sanches da Cunha.

Trabalho de Conclusão de Curso – Instituto Municipal de Ensino Superior de Assis – IMESA.

1. Sistemas operacionais. 2. Simulador. 3. Escalonador de Processos. 4. Gerência de Memória Virtual.

CDD: 001.6

Biblioteca da FEMA.

UM SIMULADOR PARA SISTEMA OPERACIONAL COM ÊNFASE NO ESCALONAMENTO DE PROCESSOS E GERÊNCIA DE MEMÓRIA VIRTUAL

MARCOS ROBERTO ALVES MEDEIROS

Trabalho de Conclusão de Curso apresentado no Instituto Municipal de Ensino Superior de Assis – FEMA/IMESA, como requisito do Curso de Graduação, analisado pela seguinte comissão examinadora:

Orientador: Me. Douglas Sanches da Cunha.

Avaliador: Prof. Esp. Célio Desiró

Assis – SP

2015

DEDICATÓRIA

Dedico este trabalho aos meus pais Maria Benedita Alves Medeiros e Claudinei Donizeti Medeiros e meus queridos irmãos Angelo, Larissa e Amanda.

AGRADECIMENTOS

Ao Me. Douglas Sanches da Cunha, não apenas por ter orientado este trabalho, mas também por nesses quatro anos de graduação, ter dado as aulas mais dinâmicas, não nos deixando desanimar nem mesmo nas aulas aos sábados no período vespertino.

À minha avó Maria Josefa, carinhosamente apelidada de Zefinha, por ser uma pessoa maravilhosa e a força que une toda a família.

Aos meus amigos do Projeto Rede Ciranda, que durante todos os dias me perguntavam sobre este trabalho.

Às minhas ex-professoras do Ensino Médio na escola E.E. Prof. Francisco Oliveira Faraco: a professora de inglês Penha Rampinelli e professora de história Beth. Ambas nos meus anos de Ensino Médio sempre me incentivaram a trilhar essa jornada.

E por fim, a todos que estiveram envolvidos neste trabalho, neste momento tão importante da minha vida.

“Nessa estrada não nos cabe conhecer ou ver o que virá, e o fim dela ninguém sabe bem ao certo onde vai dar. Vamos todos numa linda passarela, de uma aquarela e um dia enfim, descolorirá.”

(Toquinho, Aquarela)

RESUMO

Este trabalho apresenta o processo de desenvolvimento de um sistema computacional cujo objetivo é simular um sistema operacional nas suas atribuições de gerenciamento de processos e memória virtual, visando auxiliar professores a lecionar o conteúdo e também alunos a aprenderem os tópicos abrangidos por este trabalho na disciplina de Sistemas Operacionais. Este simulador é desenvolvido tendo como um dos objetivos a flexibilidade e desta maneira, o usuário pode interagir diretamente com as simulações (e escrever suas próprias rotinas), e coletar informações em tempo de execução sobre essas simulações através de elementos gráficos e textuais.

Palavras-chave: Sistemas operacionais, Simulador, Escalonador de Processos, Gerência de Memória Virtual.

ABSTRACT

This paper shows the development process of a computer system aimed to simulate an operating system on its assigned functions like: process management and virtual memory management. It aims to help teachers to teach the content and students to learn the topics that are covered by this paper. This simulator is developed with flexibility in mind. The user can interact with the simulation (and also could write his own scheduling algorithm) and collect information about it at runtime through textual and visual elements.

Keywords: Operating System, Simulator, Process Scheduling, Virtual Memory Manager.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

Um computador sem um software é uma peça de metal inútil, com software um computador pode realizar diversas tarefas como armazenar e recuperar informações, exibir conteúdo multimídia, entre outras funções. Podemos dividir os softwares em duas espécies: os aplicativos e o sistema operacional. (TANENBAUM; WOODHULL; 2000; p.17).

O sistema operacional é responsável por gerenciar a memória, processos que concorrem entre si para serem executados, dispositivos de entrada e saída, e os mais diversos dispositivos conectados ao computador. Sua função não é apenas de gerenciamento, mas também facilitar o acesso a todos esses recursos, pois sem um sistema operacional o usuário que quisesse interagir com o sistema precisaria ter profundo conhecimento do hardware que compõe o equipamento, o que tornaria seu trabalho não apenas demorado, mas também com grandes possibilidades de erro (MACHADO; MAIA; 2007; p1.1).

Levando em consideração o papel fundamental que um sistema operacional sempre teve para um computador, ele se torna material de estudo em cursos de Tecnologia e de Informação. Embora o sistema operacional facilite o acesso aos recursos da máquina, ainda é interessante estudar seu funcionamento para que se possa entender o que acontece ao realizar algumas tarefas. Com esse conhecimento um desenvolvedor pode tirar muito mais proveito de um sistema operacional, projetando e implementando softwares mais otimizados.

A disciplina de Sistemas Operacionais nos cursos de computação é constituída por uma série de tópicos extensos aos quais caberiam semestres inteiros no estudo de apenas um tópico. Devido à complexidade de tal peça de software, muito dos conceitos são abstraídos de maneira que se torne mais fácil o entendimento pelos alunos. No entanto, tornar as aulas mais dinâmicas e atrativas torna-se um desafio por parte do docente, em virtude da densidade do conteúdo lecionado, mesmo que

resumido.

Para Jones; Newman (2001), muitos dos conceitos introduzidos na disciplina de Sistemas Operacionais são abstrações e não são de fácil entendimento. Junto a isto, muitos dos conceitos são dinâmicos e não podem ser explicados de maneira fácil através de um meio estático como uma folha de papel. Enquanto muitos textos sobre Sistemas Operacionais trazem figuras para ilustrar os conceitos, estas ainda não retratam a natureza dinâmica das abstrações feitas nos sistemas operacionais. Desta forma os alunos acham mais fácil o entendimento deste conceitos quando são explicados por meio de objetos visuais.

1.1 OBJETIVOS

Este trabalho tem por objetivo desenvolver um sistema computacional capaz de simular um sistema operacional nas suas funções de gerência de processos e gerenciamento de memória, para auxiliar professores a ensinarem e alunos a assimilarem os conceitos nas aulas da disciplina de Sistemas Operacionais.

Tem como objetivo específico o desenvolvimento de uma plataforma, onde os alunos poderão experimentar no simulador e desenvolver suas próprias implementações dos algoritmos, bem como criar seus próprios mecanismos de gerência de processos através de pequenos programas escritos em linguagem Python, utilizando uma API disponibilizada pelo simulador. Dessa maneira, o usuário não ficará restrito apenas às simulações disponibilizadas pelo simulador, oferecendo uma gama enorme de possibilidades aos professores na proposta de exercícios e resolução de problemas.

1.2 JUSTIFICATIVAS

O presente projeto visa contribuir na assimilação dos conceitos envolvidos em sistemas operacionais pelos alunos nesta disciplina, bem como auxiliar os professores na explicação desses conceitos de maneira mais prática, construindo uma ferramenta interativa onde tais conceitos possam ser demonstrados. Acredita-se que utilizando uma ferramenta interativa, como a proposta neste trabalho, seja possível fomentar maior interesse pelos tópicos abordados na disciplina.

Enfaticamente Mazieira (2002) diz que entre diversas abordagens para o ensino de Sistemas Operacionais, a maioria não se mostrou efetiva, sendo que das abordagens experimentadas pelo autor, a mais promissora foi a utilização de sistemas operacionais simulados, uma vez que esses sistemas simulados fornecem um ambiente rico em interações dando amplas possibilidades de modificação. Sendo estes sistemas suficientemente realistas, constituem de uma abordagem privilegiada para o ensino da disciplina. Mas o autor ainda ressalva que tal abordagem deve ser complementada em segundo plano com outras abordagens.

1.3 METODOLOGIAS

Para o desenvolvimento desse trabalho será utilizada a linguagem C++ junto do *framework* Qt, possibilitando que o simulador possa ser executado em diversas plataformas. Serão investigados os conceitos de mais difícil assimilação pelos alunos na disciplina de Sistemas Operacionais, possibilitando priorizar esses conceitos e tentar simulá-los da maneira mais intuitiva possível. Essa pesquisa será realizada utilizando consultas na internet, bem como entrevistas a professores que lecionam esta disciplina e bibliografia especializada.

Os conceitos serão demonstrados através de simulações mais técnicas, onde os estados relacionados a uma tarefa serão descritos, e outras simulações mais

intuitivas por meio de animações.

1.4 MOTIVAÇÃO

Embora existam alguns simuladores com propostas parecidas já disponíveis, o que motivou o desenvolvimento deste trabalho foi que nenhum destes simuladores é flexível o suficiente como o proposto, onde o usuário pode alterar as simulações sem renunciar as interações com as mesmas, mantendo o controle total sobre elas. A oportunidade de desenvolver maiores habilidades nas ferramentas, tecnologias e conceitos utilizados neste trabalho também serve de grande motivação para a construção do mesmo.

1.5 PERSPECTIVAS DE CONTRIBUIÇÃO

Busca-se com este trabalho contribuir para área de ensino de Sistemas Operacionais, bem como na área simuladores como objeto de aprendizagem.

1.6 ESTRUTURA DO TRABALHO

No primeiro capítulo são abordadas algumas definições gerais de sistemas operacionais, sua disciplina nos currículos acadêmicos e sua problemática. Também são abordados os objetivos, justificativas e as motivações para a realização do trabalho.

No segundo capítulo são feitas revisões bibliográficas sobre os temas que compõem este trabalho, tais como ferramentas, conceitos e tecnologias utilizadas para o

desenvolvimento, bem como fundamentação teórica da pesquisa.

O terceiro capítulo apresenta os conceitos de gerência de processos, tipos de escalonamento e os algoritmos utilizados para a realização desta tarefa.

O quarto capítulo aborda a gerência de memória virtual, a maneira como um sistema operacional aloca e desaloca a memória virtual para os processos.

O quinto capítulo apresenta o desenvolvimento e arquitetura do simulador, o ambiente interativo onde os conceitos introduzidos no capítulo três e quatro podem ser demonstrados.

O sexto capítulo apresenta o desenvolvimento de um escalonador circular como demonstração do uso da API do simulador.

E por fim, o sétimo capítulo relata a conclusão do trabalho.

2. REVISÃO LITERÁRIA

Este capítulo apresenta a revisão da literatura. Esta revisão foi realizada buscando o estado da arte no campo de simuladores em ambientes de aprendizado, sistemas operacionais e tecnologias utilizadas.

2.1 SIMULADORES EM AMBIENTES DE APRENDIZAGEM

Segundo Flores *et al* (2014), levando em consideração que a aprendizagem representa mudanças de comportamento, procuram-se maneiras de prover experiência aos alunos, e nesta busca pelo conhecimento os simuladores tem-se mostrado grandes aliados, pois além de estarem atraindo estudantes das diversas áreas do conhecimento, ainda estão evoluindo. Destaca ainda que nestas simulações, de maneira geral busca-se disponibilizar informações com as características da vida real, assim permitindo a participação em cenários e situações próximas à realidade vivenciada.

Para Heckler *et al* (2007), ao utilizar simuladores como ferramenta de ensino e aprendizagem, é importante que tanto o professor e o aluno estejam conscientes de que os simuladores são um modelo simplificado da realidade. Desta maneira existe o risco de se assimilar uma ideia errada do fenômeno de estudo. Este trabalho parte desta ideia e dessa maneira é necessário ter um cuidado redobrado ao descrever as simulações, pois elas podem levar os alunos a entenderem e assimilarem de maneira errada o que está acontecendo nestas simulações.

2.2 SISTEMAS OPERACIONAIS

A definição do que é um sistema operacional pode variar de acordo com as concepções de quem está escrevendo sobre ele. Isso pode ser notado na literatura atual sobre sistemas operacionais. Talvez isso ocorra pois, uma vez que o sistema operacional realize as mais diversas tarefas em um sistema computacional, delimitar um escopo sobre essas funções dependerá do que o pesquisador está buscando. Levando em consideração que nos dias atuais existam muitos sistemas operacionais realizando suas tarefas de maneiras distintas, isto torna-se uma tarefa difícil de se fazer, sendo necessário abstrair as ideias. No que segue, temos algumas perspectivas sobre o que é um sistema operacional pela visão de alguns autores na área.

Segundo Tanenbaum (2009) é difícil dizer o que realmente é um sistema operacional de outra maneira que não seja um software que roda em modo *kernel,* e ainda assim isso não é uma verdade. Parte do problema é que ele faz duas coisas distintas: prover uma interface limpa e abstrata dos recursos do hardware para os programadores (e aplicações) e gerenciar esses recursos do hardware. Ele destaca que desta forma o sistema operacional pode ser entendido também como uma extensão da própria máquina.

Para Machado *et al* (2009) o que difere o sistema operacional das aplicações convencionais é a maneira como suas rotinas são executadas em função do tempo. Um sistema operacional não é executado de maneira linear como na maior parte das aplicações. Suas rotinas são executadas concorrentemente em função de eventos que podem ocorrer a qualquer momento. Este conceito pode ser melhor visualizado na Figura 1.

Segundo Silberschatz *et al* (2005), um sistema operacional é programa que gerencia o hardware do computador. Além disso provém uma base para as aplicações executarem e age como um intermediário entre o usuário do computador e o hardware do computador. Um aspecto incrível disso são as inúmeras maneiras que os sistemas operacionais encontraram para realizar estas tarefas.

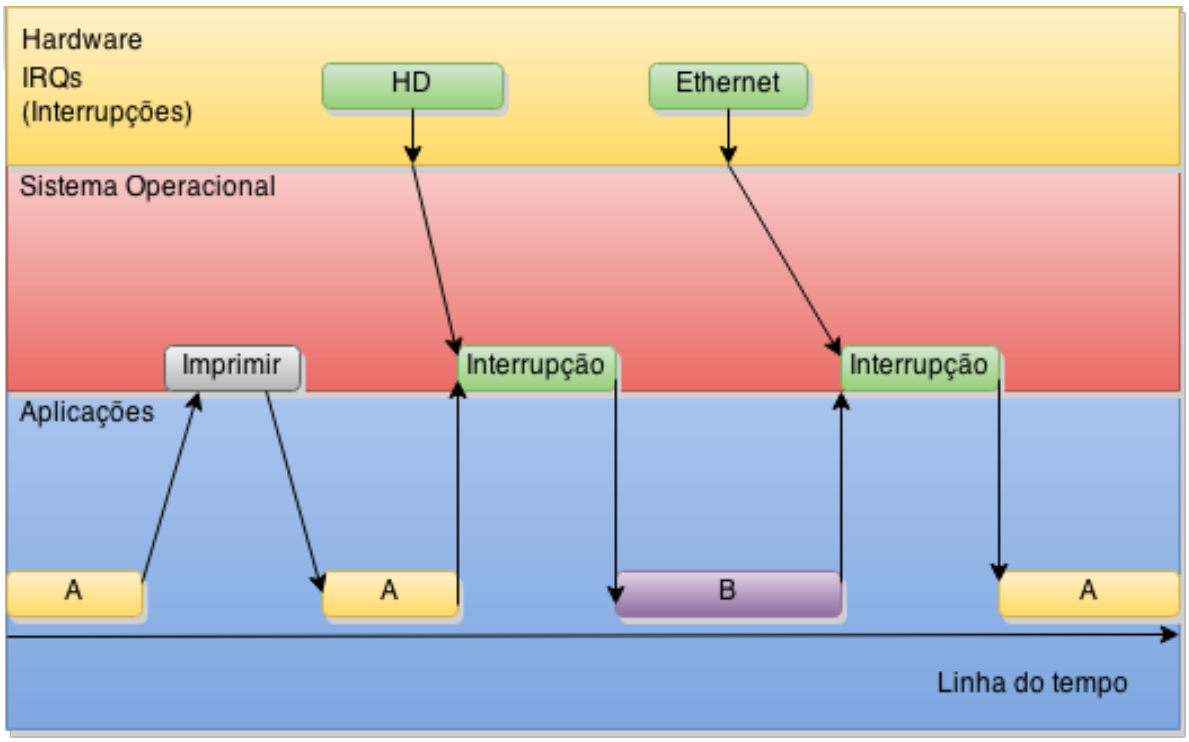

Figura 1: Execução do sistema operacional em função do tempo

As visões apresentadas por Silberschatz et al (2005) e Tanenbaum (2009) convergem no aspecto que diz respeito a prover uma plataforma para que as aplicações sejam executadas. Concordam ainda no fato do sistema operacional ser uma ponte entre o usuário e o hardware. A Figura 2 ilustra esse cenário.

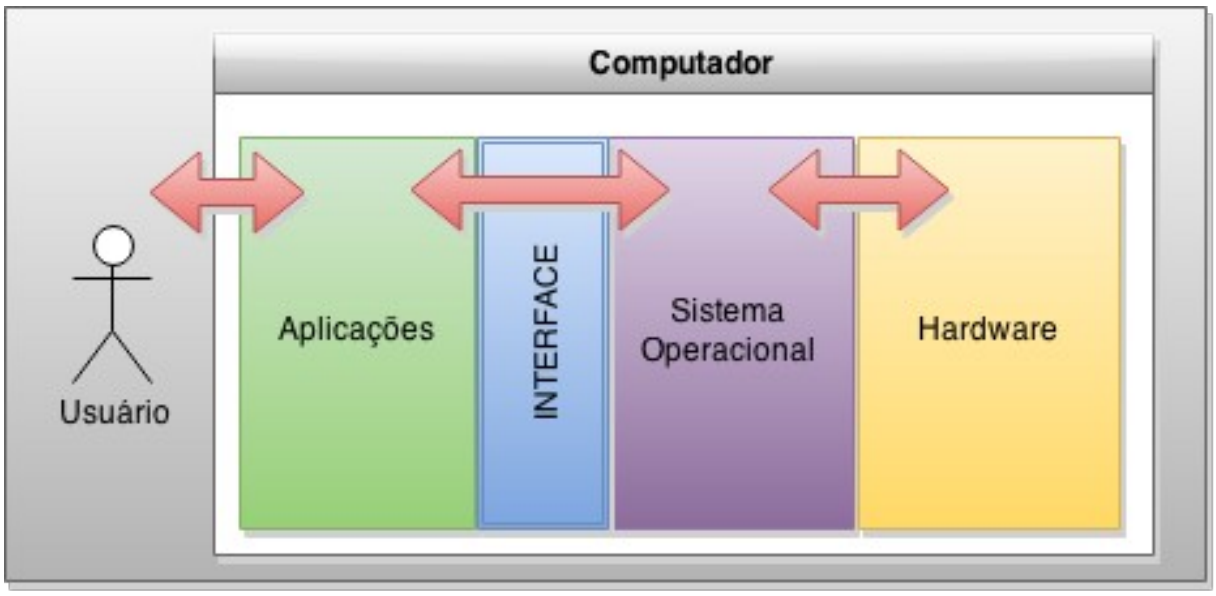

Figura 2: Interação entre usuário e máquina

É importante notar que na Figura 2, a camada de interface, que é a porta de comunicação entre as aplicações do usuário e o sistema operacional, não refere-se apenas à interface gráfica, mas também às chamadas do sistema que o sistema operacional oferece para abstrair o hardware sobre o qual ele está sendo executado. É ali onde se encontram funções tais como abrir, escrever e ler arquivos, bem como enviar e receber pacotes pela rede ou imprimir algo na impressora.

Os sistemas operacionais estão em constante evolução, segundo Tanenbaum e Woodhull (2000). Os sistemas operacionais historicamente estiveram intimamente ligados a arquitetura sobre a qual eles operam. Assim, podemos dizer que evoluções nos sistemas operacionais estão de certo modo atreladas também à evolução do hardware sobre os quais eles operam. Tendo isso em vista, é preciso compreender um pouco sobre a história dos sistemas operacionais, pois os conceitos de multiprogramação e sistemas de tempo compartilhado são as bases deste trabalho.

Os primeiros sistemas operacionais surgiram da necessidade de automatizar muitos processos manuais que existiam ao se operar um computador, que na época utilizavam cartões perfurados. Esses cartões eram chamados de *jobs* (ou tarefas). Os cartões eram lidos por uma unidade de leitura do computador, que realizava o processamento e os escrevia em uma fita magnética de entrada. Esta fita realimentava o computador, que novamente a processava e gerava uma segunda fita magnética de saída com os resultados. Esse procedimento ficou conhecido como processamento em lotes, ou *batch*. À época esse procedimento agilizou a operação de computadores, pois anteriormente os cartões eram submetidos um a um pelo operador do computador, que ficava ocioso entre a execução de um cartão e outro, destaca Machado e Maia (2007), ao contrário do que ocorria no processamento em lote onde os cartões eram submetidos todos de uma só vez.

Segundo ainda Machado e Maia (2007), o primeiro sistema operacional foi denominado monitor, pela sua simplicidade, sendo desenvolvido em 1953 justamente com o propósito de simplificar as tarefas manuais.

Em meados de 1960 foi introduzido o conceito de multiprocessamento nos sistemas operacionais, onde programas podiam compartilhar recursos como a memória de um

computador, que permanecia com sua unidade de processamento ocioso quando operações de entrada e saída (tais como ler e escrever na fita magnética) eram realizadas. Para Tanenbaum (2009), a multiprogramação foi a adição mais importante dentro desta geração de sistemas operacionais.

Embora a multiprogramação tenha resolvido o problema da ociosidade, tentando manter a unidade de processamento a maior parte do tempo ocupada, os computadores ainda faziam processamento em lote. Segundo Tanenbaum (2009), a necessidade dos programadores obterem acesso ao computador para poder depurar seus programas, uma vez que se um erro ocorresse o tempo de resposta seria muito grande, fez com que surgissem os sistemas de tempo compartilhado, onde cada tarefa poderia utilizar a unidade de processamento por pequenos intervalos de tempo. Para acesso a esses sistemas os usuários tinham acesso a um terminal online que era conectado ao computador. Dentre esses sistemas destaca-se o CTSS (*Compatible Time-Sharing System*) desenvolvido no MIT em 1962.

Os sistemas operacionais podem ser divididos em três categorias, segundo Machado e Maia (2007): os sistemas monoprogramáveis (ou monotarefas), os sistemas multiprogramáveis (multitarefas) e sistemas com múltiplos processadores. Os sistemas monoprográveis são aqueles onde apenas uma tarefa pode estar em execução, isto é, ela detém todo o hardware para si, mesmo enquanto aguarda que uma operação de entrada e saída seja realizada, fazendo com que o processador permaneça ocioso durante esse tempo. Para que outra tarefa possa ser iniciada é necessário que a tarefa corrente termine, liberando todos os recursos para a nova tarefa entrar em execução. O MS DOS era um sistema operacional desta categoria.

Os sistemas multiprogramáveis são aqueles onde ao contrário dos sistemas monoprogramáveis, os recursos do computador são compartilhados entre as tarefas. Dessa maneira, quando uma tarefa requisita uma operação de entrada e saída, a execução de outra tarefa será posta para execução enquanto esta outra que estava em execução será suspensa até que a operação de E/S esteja concluída. O sistema operacional é responsável por gerenciar os recursos da máquina para que estes sejam compartilhados entre os programas, recursos esses tais como: memória, processador, disco rígido e quaisquer outros dispositivos. É desejável que esse

compartilhamento de recursos seja justo com todas as tarefas em execução. Nesta categoria temos muitos representantes tais como Linux, Windows, Unix, e BSDs.

Por fim, Machado e Maia (2007), destacam que os sistemas de multiprocessadores são aqueles em que as máquinas possuem mais de um processador, e estes estão trabalhando em conjunto, executando tarefas simultaneamente, ficando a cargo do sistema operacional gerenciar estes recursos. Estes tipos de sistemas tem as mesmas características dos sistemas multiprogramáveis, junto das vantagens de escalabilidade, disponibilidade e balanceamento de carga. Estes sistemas ainda podem ser classificados de duas maneiras: os fortemente acoplados e os fracamente acoplados. Nos sistemas fortemente acoplados existe apenas uma memória principal que é compartilhada pelos processadores do sistema por um barramento de alta velocidade. Nos sistemas fracamente acoplados cada sistema possui sua própria memória, placa-mãe, etc, e as máquinas estão conectadas através de um barramento mais lento, geralmente uma rede local.

2.3 TECNOLOGIAS UTILIZADAS

Nesta subseção serão apresentadas as ferramentas utilizadas para o desenvolvimento deste trabalho. Tais ferramentas foram selecionadas utilizando como principal requisito a portabilidade entre plataformas e gratuidade de uso.

2.3.1 LINGUAGEM C++

Segundo Stroustrup(2015), seu criador, a linguagem C++ pode ser definida como uma linguagem de propósito geral com tendências para a programação de sistemas que é uma versão do C melhorada, oferece suporte a abstração de dados, suporta programação orientada a objetos e suporta programação genérica.

A linguagem C++, apesar de ter um criador, não possui um proprietário, ela é regulamentada pelas normas ISO/IEC 14882 *"The C++ Standard"*. As normas são definidas pelo comitê de gestão ISOCPP que define um padrão a cada 3 anos, visando aprimorar a linguagem conforme as necessidades. No momento de escrita deste trabalho os dois últimos padrões são o C++11 e C++14 respectivamente. Para o desenvolvimento do simulador será utilizado o padrão C++11.

2.3.2 FRAMEWORK Qt

O framework Qt será utilizado para o desenvolvimento de todos os componentes do software relacionados à interface com o usuário. Seu uso ainda implica na portabilidade do software, fazendo com que ele não fique restrito à apenas uma plataforma. Seu desenvolvimento iniciou-se em 1991 pela hoje extinta Trolltech e nos dias atuais é de propriedade da Digia Plc., utilizando algumas modalidades das licenças de código aberto do Projeto GNU.

Segundo Thelin (2007) Qt é um kit de desenvolvimento gráfico multiplataforma para o desenvolvimento de aplicações, que lhe proporciona poder compilar um programa e executá-lo em diversas plataformas como Windows, OSX, Linux e outras vertentes do UNIX. Uma grande parte do Qt está comprometida em prover uma interface neutra para as plataformas suportadas para tudo, desde a representação dos caracteres na memória à criação de aplicações gráficas multitarefa.

2.3.3 LINGUAGEM PYTHON

A documentação oficial da linguagem Python[1] provida pela Python Software Foundation (2015), a descreve como uma linguagem interpretada que utiliza o paradigma de programação orientada a objetos. É também uma linguagem de alto

1 https://docs.python.org/3/faq/general.html#what-is-python

nível com semântica dinâmica, contendo estruturas de dados de alto nível já embutida, combinada com outros aspectos dinâmicos, ela se torna atrativa para o desenvolvimento rápido de aplicações, ou também como linguagem de *scripting* ou como uma linguagem para integrar componentes já existentes.

2.4 TRABALHOS CORRELATOS

Alguns trabalhos já foram realizados nessa área buscando desenvolver uma ferramenta que auxilie os alunos e professores na disciplina de Sistemas Operacionais. Dentre eles, o SOsim, um simulador de sistemas operacionais com propósitos educacionais acerca de gerência de processador e memória. Segundo Maia *et al* (2001) através do emprego do simulador conseguiu-se bons resultados em sala de aula, onde registrou-se um aumento na eficiência de assimilação do conteúdo por parte dos professores e alunos. Melhorou ainda a comunicação entre professores e alunos, reduzindo o tempo gasto em apresentações conceituais, o que permitiu estender o programa da disciplina.

Segundo Gadelha *et al* (2010), no desenvolvimento de um simulador para auxílio de aprendizado sobre conceitos de sistemas de arquivos em um sistema operacional denominado OS Simulator, conseguiu-se resultados positivos na avaliação do mesmo em pesquisa realizada com os alunos, sendo que no processo de ensino e aprendizagem os contemplados apresentaram um ganho de conhecimento prático sobre as teorias ministradas.

Segundo Reis *et al* (2008), no desenvolvimento do TCB-SO/WEB (Treinamento Baseado em Computador para Sistemas Operacionais via Web), que visa auxiliar no processo de aprendizagem de políticas de escalonamento bem como políticas de alocação de memória, constatou-se que a qualidade de ensino é melhorada com o uso de ambientes educativos, pois além de facilitar a aquisição de conhecimento eles ainda estimulam o processo de raciocínio e de abstração, que são frequentemente encontrados em cursos de computação.

O RCOS.java desenvolvido por Jones; Newman (2001) tem uma proposta mais abrangente que a deste trabalho, porém tem objetivos parecidos. Ele conta com simulações das mais diversas partes de um sistema operacional, desde mecanismo de sincronização entre processos até mesmo o de funcionamento do hardware que o simulador controla. Segundo os autores, o RCOS.java é um sistema operacional implementado em Java contendo diversas funções para auxiliarem os alunos no entendimento de algoritmos, conceitos e teorias por trás do design, construção e operação de sistemas operacionais.

3. GERÊNCIA DE PROCESSOS

Por questões de simplicidade, este trabalho parte do conceito de que um processo é um programa em execução contendo três partes: contexto de software, contexto de hardware e espaço de endereçamento.

Segundo Machado e Maia (2007), a gerência de processos é uma das principais funções de um sistema operacional, possibilitando aos programas alocar recursos, compartilhar dados, trocar informações e sincronizar suas execuções. Destaca ainda que nos sistemas multiprogramáveis os processos são executados concorrentemente, compartilhando o uso do processador bem como da memória principal e dispositivos de entrada e saída. Já nos com múltiplos processadores, além de existir as mesmas características dos multiprogramáveis, existe também a possibilidade de execução simultânea de diferentes processos nos processadores disponíveis.

3.1 CONCEITOS DE PROCESSO

Para Tanenbaum (2009), o conceito de processo é fundamental dentro de um sistema operacional, que para ele nada mais é que um programa em execução que está atrelado ao seu espaço de endereçamento onde o processo pode ler e escrever, um conjunto de recursos tais como registradores como o contador do programa e apontador de pilha, lista de arquivos que o processo abriu, alarmes entre muitas outras informações. O autor destaca que a ideia principal é que um processo constitui uma atividade, possuindo programa, entrada, saída e um estado.

Para Machado e Maia (2007), um processo pode ser entendido de duas formas: como um programa em execução inicialmente, porém isto pode ser entendido como

um conjunto necessário de informações para que o sistema operacional implemente a concorrência de programas; ou um processo pode ser definido como o ambiente onde um programa é executado, sendo que neste ambiente, além das informações sobre execução, reside também a quantidade de recursos do sistema que cada programa pode alocar, tais como endereçamento de memória principal, tempo de processador e área em disco.

Segundo Silberschatz e Gagne (2013), nos sistemas operacionais primordiais, uma única tarefa poderia ser executada por vez. Dessa maneira, esta tarefa obtinha acesso exclusivo a todos os recursos. Porém, hoje os sistemas operacionais permitem que diversos programas sejam carregados na memória e executados concorrentemente. Tal evolução demandou maior controle e maior compartimentação dos programas, o que resultou no conceito de processo, que é um programa em execução. Os autores destacam ainda que um processo é a unidade de trabalho de um sistema de tempo compartilhado.

Um processo é composto por três partes, como destaca Machado e Maia (2007). Essas partes são conhecidas como contexto de hardware, contexto de software e espaço de endereçamento. Essas três partes mantêm todas as informações necessárias à execução de um programa. No contexto de hardware encontram-se informações que serão levadas diretamente aos registradores do processador como registradores gerais, contador de programa, apontador de pilha, registrador de estado, para que o processo possa ser iniciado ou resumido. O contexto de software é o conjunto de informações, tais como limites de recursos que o processo pode alocar, lista de arquivos abertos, prioridade de execução, tamanho da área de memória (*buffer*) para operações de entrada e saída, privilégios, identificação, entre outras informações que o sistema operacional pode implementar. O espaço de endereçamento está ligado a memória virtual, sendo a área de memória onde se localizam as instruções e os dados do programa. Este espaço deve ser protegido do acesso de outros processos, desta maneira cada processo possuí seu próprio espaço de endereçamento.

3.1.1 Estados de um processo

Uma vez que um processo pode ser definido como um programa em execução em um sistema operacional, este pode estar em um dos três estados: estado de pronto, em execução ou estado de espera. Machado e Maia (2007) destacam em um sistema multiprogramável onde um processo não pode alocar o processador exclusivamente, para que haja compartilhamento do tempo do mesmo, assim, os processos passam por diferentes estados durante o processamento, essas transições podem ocorrer devido a eventos gerados pelo sistema operacional ou pelo próprio processo.

Segundo Tanenbaum (2009), quatro transições são possíveis entre esses estados: um processo de estado de espera pode apenas ir para o estado de pronto; um processo em estado de pronto pode ir para o estado de execução; no estado de execução um processo pode seguir para espera ou para pronto para ser selecionado novamente pelo escalonador de tarefas (agendador de tarefas – *scheduler*).

As definições abaixo para os estados de um processo são baseadas nas definições dadas por Machado e Maia (2007).

3.1.1.1 Estado de execução

Um processo está em estado de execução quando está sendo executado pelo processador. Somente um processo pode estar sendo executado em um dado momento no processador, nos sistemas com múltiplos processadores é possível executar tantas tarefas simultâneas quanto a quantidade de núcleos.

3.1.1.2 Estado de pronto

Um processo permanece em estado de pronto enquanto aguardar para ser selecionado pelo agendador de tarefas para entrar em estado de execução. O sistema operacional é responsável por determinar a ordem bem como os critérios para que os processos passem ao estado de execução. A isto se dá o nome de escalonador ou agendador de tarefas. Geralmente vários processos estão em estados de pronto em um sistema operacional.

3.1.1.3 Estado de espera

Um processo permanece no estado de espera enquanto aguarda por um evento externo (como uma operação de entrada e saída) ou para conseguir acesso a algum recurso do sistema para prosseguir executando. Durante o tempo em que o processo permanece em estado de espera, ele não é selecionado pelo escalonador para entrar em execução.

3.2 PROCESSOS IO-BOUND e CPU-BOUND

Ao estudarmos sobre gerência de processos é importante compreender que existem geralmente duas categorias de processos sendo executados no sistema operacional: os processos limitados pela CPU (ou *CPU-Bound*) e os processos limitados pela entrada e saída (*IO-Bound*).

Tanto Silberschartz (2013) quanto Tanenbaum (2009) destacam que processos limitados pela entrada e saída são aqueles que passam a maior parte do tempo executando operações de entrada e saída em vez de processamento; já os processos limitados pela CPU são aqueles que passam a maior parte do tempo

executando o processamento. Segundo Tanenbaum (2009) o fator principal de distinção entre essas categorias de processos é o tempo de processamento que um processo toma da fatia de tempo de processamento que o escalonador lhe fornece.

3.3 O ESCALONADOR DE PROCESSOS

Para Machado e Maia (2007), com o surgimento dos sistemas multiprogramáveis, onde múltiplos processos poderiam permanecer na memória compartilhando a CPU, a gerência do processador tornou-se uma das atividades mais importantes em um sistema operacional. Uma vez que mais de um processo estão em estado pronto, é necessário definir critérios para decidir qual será próximo processo a entrar em execução; esses critérios definem a política de escalonamento que, segundo os autores, é a base para a gerência do processador e sistemas multiprogramáveis. Tanenbaum (2009) segue a mesma linha de raciocínio, mas complementa que a parte que realiza a escolha do próximo processo a ser selecionado é denominado escalonador, e o algoritmo utilizado é chamado de algoritmo de escalonamento.

Segundo Silberschartz (2013, p. 59):

> O objetivo da multiprogramação é sempre termos algum processo em execução para a otimização do uso da CPU. O objetivo do compartilhamento de tempo é a alternância da CPU entre os processos com tanta frequência que os usuários possam interagir com cada programa enquanto ele está sendo executado. Para alcançar esses objetivos, o *scheduler* de processos seleciona um processo disponível (possivelmente em um conjunto de vários processos disponíveis) para a execução na CPU. Em um sistema de processador único, nunca haverá mais de um processo em execução. Se houver mais processos, os outros terão que esperar até a CPU estar livre e poder ser reprogramada.

Tanenbaum (2009) afirma que o escalonador, além de decidir qual o processo certo para ser executado, ainda deve se preocupar em fazer uso eficiente da CPU, pois as trocas de contextos entre processos são bem custosas. O estado atual do processo (incluindo contexto de software, hardware e espaço de endereçamento) deve ser salvo na tabela de processos para depois ser recarregado quando o processo for selecionado novamente. Junto a isso, normalmente, a troca de contextos exige que se invalide toda a memória cache, um procedimento que se não for feito da maneira certa pode gastar muito tempo do processamento. Silberschatz (2013, p. 61) exemplifica esse cenário da seguinte maneira: se o escalonador leva 10 milissegundos para decidir qual o próximo processo a ser executado por 100 milissegundos, teremos que 10/(100 + 10) = 9 por cento da CPU será usada para apenas decidir qual o próximo processo.

3.3.1 Escalonamento preemptivo e não preemptivo

Segundo Machado e Maia (2007), a possibilidade de o sistema operacional interromper um processo em execução e substitui-lo por outro, de maneira que seu estado agora seja pronto e aguarde novamente para entrar em estado de execução, é denominada preempção; os sistemas operacionais que implementam a preempção são mais complexos, porém permitem maior flexibilidade nas políticas de escalonamento.

Nos sistemas onde não é implementada a preempção, um processo em execução só será substituído por outro nas seguintes situações, segundo relata Silberschatz (2013): quando um processo passa para o estado de espera em função de uma operação de entrada ou saída, bem como esperar pelo encerramento de um dos processos filhos; ou quando o processo é terminado. Machado e Maia (2007) dizem que este foi o primeiro tipo de escalonamento implementado nos sistemas multiprogramáveis, tipicamente os sistemas em lote, sendo que neste tipo de escalonamento nenhum evento externo poderia ocasionar a perda do processador pelo processo corrente.

3.3.2 Critérios de escalonamento

Segundo Silberschatz (2013, p. 103), muitos critérios têm sido sugeridos para a comparação entre os algoritmos de escalonamento, os critérios utilizados podem fazer muita diferença no momento de dizer qual o melhor algoritmo. Alguns dos critérios de escalonamento descritos por Silberschatz (2013), Machado e Maia (2007) e Tanenbaum (2009) são:

- **Utilização da CPU**: é desejável manter a CPU com o máximo uso possível. Silberschatz (2013) destaca que conceitualmente a utilização da CPU pode variar de 0 a 100 por cento, no entanto em sistemas reais, ela varia de 40 por cento (quando existem poucos processos) à 90 por cento (quando existe alta carga de processos).

- **Throughput**: é o que denominamos vazão de processos, isto é, este valor é o número de processos que são executados durante um certo intervalo de tempo. Segundo Machado e Maia (2007) a maximização desse valor é desejável na maior parte dos sistemas.

- **Tempo de turnaround**: este valor se refere a quantidade de tempo que um processo está em execução, ou seja, do momento de sua criação até o momento de sua finalização. Silberschatz (2013) destaca que esse tempo é compreendido pela soma dos tempos gastos com a espera para ser alocado na memória, a espera na fila de prontos, a execução na CPU e a execução de operações de entrada e saída.

- **Tempo de espera**: Esse valor é o resultado da soma de todos os tempos gastos na fila de prontos, segundo destaca Silberschatz (2013).

- **Tempo de resposta**: Para Tanenbaum (2009), esse é o intervalo de tempo determinado pela diferença entre o tempo em que uma requisição foi feita ao sistema operacional com o tempo em que ela foi servida por ele.

Tanenbaum (2009) destaca que, para diferentes ambientes, são necessários

diferentes tipos de algoritmos de escalonamento, pois diferentes áreas têm objetivos diferentes. Desta maneira o escalonador deve ser otimizado para esses ambientes. Alguns ambientes mencionados são: sistemas de lote, os sistemas interativos onde existe um usuário interagindo com os processos, e sistemas de tempo real nos quais o tempo de resposta a um evento deve ser rigidamente cumprido. Dentro desses requisitos Tanenbaum (2009) destaca ainda que em todos esses ambientes algumas políticas de escalonamento são desejáveis:

- **Justiça**: dar a cada processo uma fatia justa do tempo da CPU.
- **Aplicação da política**: verificar se as políticas estabelecidas estão sendo cumpridas.
- **Equilíbrio**: manter ocupado ao máximo todas as partes do sistema.

3.3.3 Algoritmos não preemptivos

Como já relatado, os algoritmos não preemptivos são geralmente utilizados nos sistemas em lote, sendo que nenhum evento externo pode interromper o sistema e implicar na perda da CPU do processo corrente, a não ser de maneira voluntária. A seguir veremos alguns exemplos de algoritmos não preemptivos.

3.3.3.1 Primeiro a Entrar, Primeiro a ser Atendido

O algoritmo “primeiro a entrar, primeiro a ser atendido” (FCFS – *first come, first served*) é o mais simples dos algoritmos de escalonamento, sendo de fácil entendimento e desenvolvimento, podendo ser facilmente gerenciado por uma fila FIFO (*first in, first out*), destaca Silberschatz (2013).

Neste tipo de escalonamento não preemptivo, a primeira tarefa a ser submetida para

execução será a primeira a ser executada, e as outras tarefas que forem submetidas serão executadas na ordem de chegada. Se voluntariamente o processo corrente decidir liberar a CPU, ele terá de entrar no fim da fila e esperar novamente sua vez para voltar ao estado de execução. A Figura 3 exemplifica esta fila.

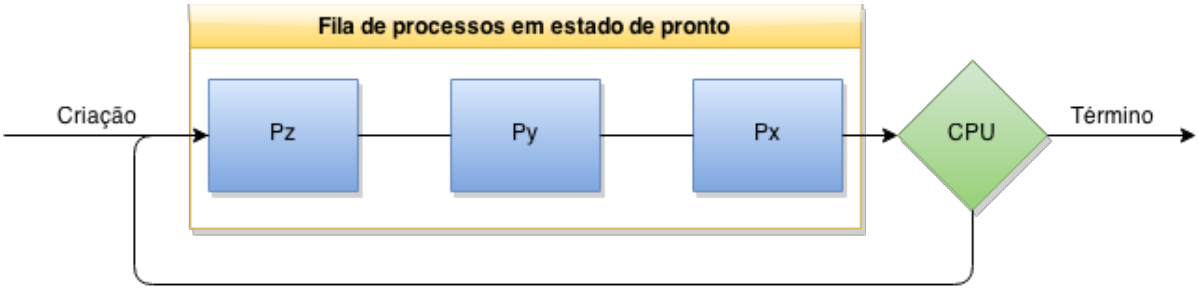

Figura 3: Escalonamento FCFS

Considerando a seguinte lista de processos, contendo os processos que estão em estado de pronto, ilustrada pela Tabela 1.

Entrada	Processo	Tempo de CPU
1	P1	35
2	P2	5
3	P3	10

Tabela 1: FCFS – Lista de pronto

O tempo de espera é o tempo em que um processo espera até entrar em execução na fila de prontos, isto é, se existem três processos A, B e C, e o processo A permanece em execução por X unidades de tempo e o processo B por Y unidades de tempo, dizemos que o tempo de espera de B é X, e o tempo de espera para A é 0, uma vez que ele foi o primeiro processo na fila e não esperou por outros

processos. Para o processo C o tempo de espera é X+Y, pois ele esperou o processo A e B serem executados para entrar em execução.

Para os processos P1, P2, P3 na Tabela 1 os tempos de espera são 0, 35 e 40 respectivamente, tendo como média de tempo de espera um total compreendido por (0+35+40)/3 = 25 unidades de tempo para cada processo na lista de pronto. Se a fila de prontos for reordenada de maneira que contenha os processos na seguinte ordem: P3, P2 e P1, tem-se como tempo médio de resposta (0+10+15)/3 = 8 unidades de tempo. Para Silberschatz (2013), o lado negativo desse algoritmo é justamente o tempo médio de espera, que costuma ser bem longo.

Machado e Maia (2007) destacam que uma das grandes deficiências desse algoritmo é que processos *IO-bound* tem desvantagem em relação aos processo *CPU-bound*, não havendo como tratar essa diferença.

3.3.3.2 Trabalho mais curto primeiro

O escalonamento pelo trabalho mais curto, também conhecido como *Shortest Job First* (SJF), é também um algoritmo não preemptivo, muito parecido com o algoritmo FCFS. Porém para o funcionamento deste algoritmo, é necessário que se saiba previamente o tempo de todos os processos a serem executados. Segundo Tanenbaum (2009), quando várias tarefas estiverem na fila de prontos, o escalonador irá escolher a tarefa com menor tempo de processamento primeiro. Desta maneira, o escalonador irá apenas reordenar as tarefas na fila de pronto para que as com menor duração executem primeiro.

Reconsiderando a lista de prontos da Tabela 1 apresentada no escalonamento FCFS, a ordem em que os processos seriam executados é representado pela Tabela 2.

Entrada	Processo	Tempo de CPU
1	P2	5
2	P3	10
3	P1	35

Tabela 2: Escalonamento SJF

O tempo de resposta médio para este escalonamento neste caso é de (0+5+15)/3 = 8 unidades de tempo. Dessa forma este algoritmo se torna uma opção melhor que o FCFS. No entanto, é pré-requisito que se conheça todos os tempos que os processos levarão até o fim de sua execução quando forem selecionado para entrar em execução, porém na maior parte das situações isso não está definido.

3.3.4 Algoritmos preemptivos

Os algoritmos de escalonamento preemptivos são utilizados nos sistemas interativos, sendo comuns em computadores pessoais, servidores e outros tipos de sistema, destaca Tanenbaum (2009).

3.3.4.1 Escalonamento Circular (*Round-Robin*)

Segundo Tanenbaum (2009), o escalonamento circular (mais conhecido como Round-Robin) é um dos algoritmos mais antigos, simples, justos e amplamente utilizado, sendo que a cada processo é atribuído uma fatia de tempo do processador denominada ***quantum***, no qual o processo tem a permissão de ser executado; se ao final dessa fatia de tempo o processo ainda estiver sendo executado, então ele

sofrerá preempção por tempo, isto é, será substituído pelo próximo processo na lista de pronto e será colocado no fim da fila de prontos para ser executado novamente quando chegar sua vez, continuando o processamento do ponto em que foi interrompido.

Para Silberschatz (2013), este algoritmo foi especialmente projetado para os sistemas de tempo compartilhado, sendo semelhante ao FCFS, porém adicionando a preempção por tempo para permitir que o sistema operacional alterne entre os processos; a fila de prontos é tratada como uma fila circular, sendo que o quantum de cada processo tem entre 10 a 100 milissegundos.

Segundo Machado e Maia (2007) o desempenho deste tipo de escalonamento está diretamente ligado ao tamanho do quantum. Se o quantum for muito grande, então o comportamento deste algoritmo será parecido ao do FCFS; caso o quantum for muito pequeno a performance degradaria pelo alto número de preempções e trocas de contextos, que são custosas. Tanenbaum (2009) exemplifica isso da seguinte maneira: supondo que a troca de contexto entre um processo e outro dure 1 milissegundo e que o quantum seja 4 milissegundos, assim, depois de realizar 4 ms de trabalho a CPU terá de gastar 1 ms para realizar uma troca de contexto; dessa maneira 20 por cento do tempo da CPU é utilizado no escalonamento de processos, um valor muito alto.

Por ser muito parecido com o escalonamento FCFS, a maneira como esse algoritmo funciona é ilustrada na Figura 4.

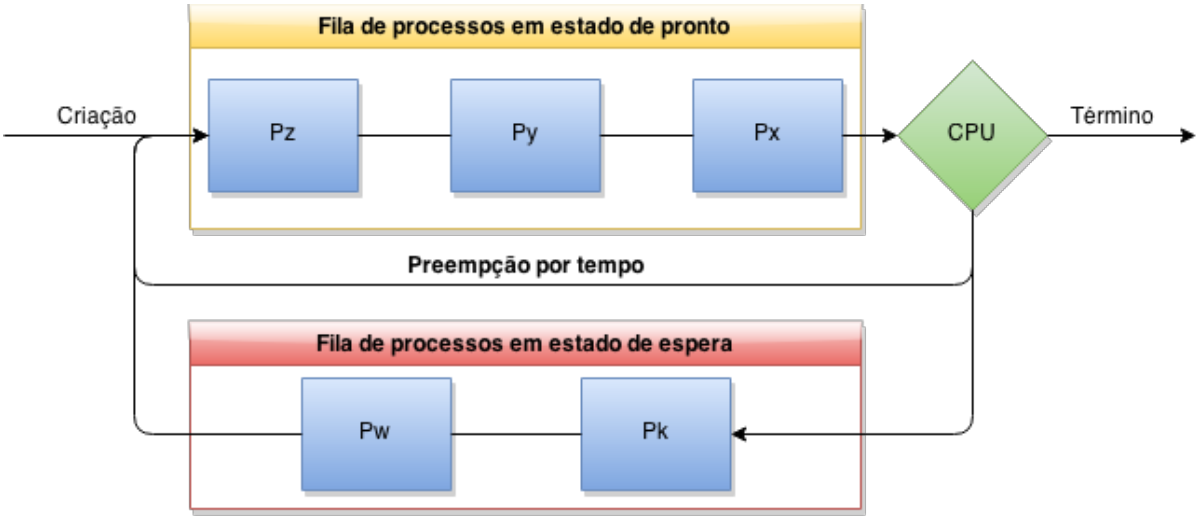

Figura 4: Escalonamento Circular (Round-Robin)

Silberschatz (2013) destaca que os tempos médios de espera para esse escalonador são geralmente longos. Consideremos os processos na lista de pronto e seu quantum na Tabela 3. Supondo que o quantum seja de 4 milissegundos, o primeiro processo P1 processará por 4 ms e sofrerá preempção, necessitando ainda de mais 20 ms de processamento. Assim os próximos processos P2 e P3 não necessitam de todo o quantum e logo liberam a CPU. Dessa maneira P1 receberá mais um quantum e executará o processamento. Quando o quantum expirar ele receberá outra fatia até que tenha sido finalizado, uma vez que só resta ele na lista de prontos. A tabela de escalonamento circular para esse caso está na Tabela 4.

Entrada	Processo	Quantum
1	P1	24
2	P2	3
3	P3	3

Tabela 3: Lista de pronto Round-Robin

Processo	P1	P2	P3	P1	P1	P1	P1	P1	
Instante	0	4	7	10	14	18	22	26	30

Tabela 4: Tabela de escalonamento Round-Robin

O processo P1 espera por 6 ms (10 – 4), P2 espera por 4 ms e P3 espera por 7 ms, logo o tempo médio de espera é de (6+4+7)/3 = 5,66 ms, destaca Silberschatz (2013).

3.3.4.2 Escalonamento por prioridades

Segundo Machado e Maia (2007), o escalonamento por prioridades é um algoritmo que realiza a seleção do próximo processo baseado em um valor de prioridade atribuído ao processo, sendo que o processo com maior prioridade é sempre escolhido para execução, enquanto os com mesma prioridade são escalonados seguindo o critério de fila FIFO; não existe também o conceito de fatia de tempo.

Silberschatz (2013) destaca que geralmente as prioridades são definidas por um intervalo fixo de números, como de 0 a 7, no entanto não existe consenso geral para definir se 0 é a prioridade mais baixa ou mais alta. Para a descrição deste item utilizamos 0 como a prioridade mais alta.

A perda do processador pelo processo corrente somente irá ocorrer em caso de mudança voluntária (em uma operação que coloca o processo em estado de espera), ou em caso de um processo com maior prioridade se tornar disponível, então o processo corrente sofrerá o que é chamado de preempção por prioridade, destaca Machado e Maia (2007).

Um dos problemas com este escalonamento é que os processos com baixas prioridades podem sofrer com o problema de *starvation.* Nesta situação o processo fica esperando indefinidamente para ser executado, no entanto a presença de

processos com maior prioridade impedem que ele seja selecionado.

Tanenbaum (2009) destaca que para evitar a situação de *starvation*, o escalonador pode diminuir a prioridade de execução do processo a cada tique do relógio, uma vez que o processo fique com uma prioridade abaixo de outro processo, ocorrerá então uma troca de contexto para esse novo processo; outra possibilidade seria atribuir um quantum máximo de tempo para o processo ser executado, e quando este tiver acabado, seria dado a oportunidade de outros processos executarem.

O funcionamento geral deste algoritmo é ilustrado pela Figura 5.

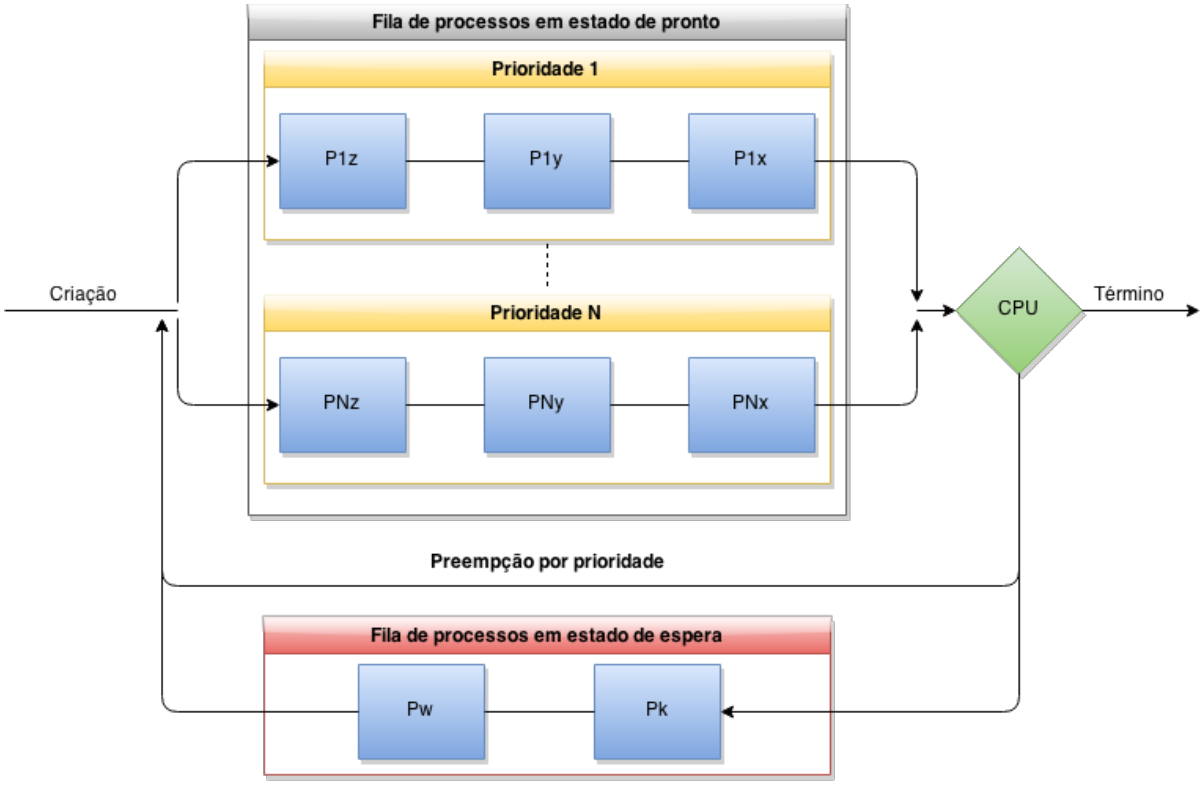

Figura 5: Escalonamento por prioridades

O exemplo abaixo é proposto por Silberchatz (2013, p; 105): considere a Tabela de processos em estado de pronto.

Processo	Tempo de CPU	Prioridade
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

Tabela 5: Lista de pronto – Prioridades

Processo	P2	P5	P1	P3	P4	
Instante	0	1	6	16	18	19

Tabela 6: Tabela de escalonamento Prioridades

Neste exemplo podemos calcular que o tempo médio de espera é (0+1+6+16+18)/5 = 8,2 milissegundos.

4. GERÊNCIA DE MEMÓRIA VIRTUAL

Segundo Silberschatz (2013), memória virtual é uma técnica que permite a execução de processos que não estão completamente residentes na memória principal, sendo uma das maiores vantagens o fato de que os programas podem ser maiores que a memória principal; isto é feito abstraindo a memória em um arranjo enorme e uniforme, separando a memória lógica da memória física.

Machado e Maia (2007) destacam que a técnica de memória virtual é um mecanismo sofisticado, onde as memórias principal e secundária são combinadas, dando ilusão ao usuário da existência de uma memória muito maior que a realmente existente.

Para Tanenbaum (2009), a ideia básica por trás da memória virtual consiste em cada processo ter seu próprio espaço de endereçamento, que é dividido em blocos de tamanho fixo denominados páginas, de maneira que cada página seja uma série contínua de endereços. Estas páginas são mapeadas na memória física, no entanto não é necessário que todas elas estejam mapeadas para a execução do processo. Uma vez que o processo tente acessar uma página de memória o hardware faz a tradução dinamicamente entre o endereço virtual e o endereço físico. Caso a página não esteja mapeada no espaço de endereçamento do processo, o sistema operacional é chamado para que possa mapear as páginas que estão ausentes para que o processo possa prosseguir executando.

4.1 ESPAÇO DE ENDEREÇAMENTO VIRTUAL

O espaço de endereçamento virtual de um processo diz respeito à visão lógica (ou virtual) de como um processo está carregado na memória. Normalmente essa visão

parte de um endereço lógico (0 por exemplo) e permanece em memória contínua, destaca Silberschatz (2013).

Segundo Tanenbaum (2009), o espaço de endereçamento virtual é dividido em unidades denominadas páginas (*pages*), sendo que as unidades correspondentes na memória física são chamadas de molduras de página (*page frames*). Este conceito é exemplificado através da Figura 6, onde existem dois processos A e B, cada um com seu espaço de endereçamento com algumas páginas mapeadas para as molduras da memória principal.

Este mecanismo de mapeamento entre a memória virtual e a memória física é denominado paginação, embora existam outras maneiras de se conseguir um funcionamento parecido com segmentação por exemplo, este trabalho tem o foco em memória virtual baseado na técnica de paginação.

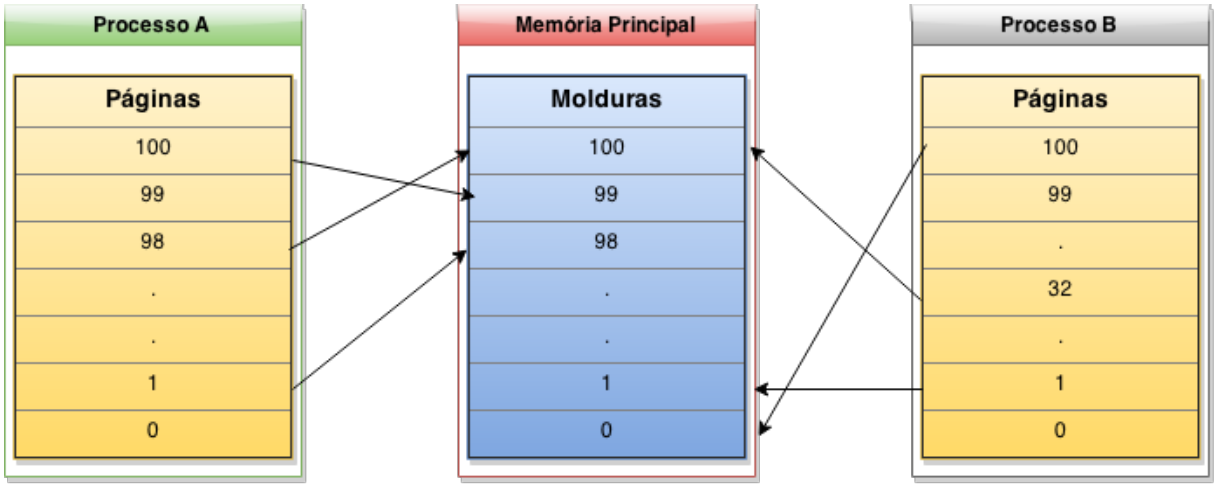

Figura 6: Mapeamento de memória

Machado e Maia (2007), dizem que, por não existir relação direta entre a memória virtual e memória física, um programa pode fazer referências a endereços virtuais que estejam fora dos limites da memória física. Dessa maneira os programas não estão mais limitados a quantidade de memória física disponível; isso é possível, pois o sistema operacional faz uso da memória secundária como extensão da principal.

4.2 POLÍTICAS DE BUSCA DE PÁGINAS

4.2.1 Paginação Antecipada

Na política de busca de páginas antecipada, o sistema operacional carregas várias páginas para a memória principal antes delas serem referenciadas, que podem ou não ser necessárias durante a execução do processo durante seu processamento; se essas páginas se encontrarem armazenadas sequencialmente na memória secundária, existe uma grande economia de tempo se comparado quando é preciso carregar uma página de cada vez, destacam Machado e Maia (2007). Um dos problemas desta técnica é que ela pode consumir muita memória física sem necessidade, uma vez que as páginas carregadas podem não ser referenciadas durante a execução do processo.

4.2.2 Paginação Por Demanda

Segundo Silberschatz (2013, p. 189), “com a memória virtual paginada por demanda, as páginas são apenas carregadas quando necessárias durante a execução do programa, portanto, páginas que jamais serão acessadas, jamais serão carregadas na memória principal”.

Dessa maneira, podemos afirmar que ao contrário da paginação antecipada, utilizando a paginação por demanda o sistema operacional só irá inicialmente carregar as páginas necessárias para o início da execução do programa, e assim que as outras páginas forem referenciadas serão geradas interrupções no sistema operacional, denominadas *page faults*, que notificarão a ausência da página referenciada na memória, cabendo ao sistema operacional buscar essa página e carregá-la na memória principal para que o processo prossiga.

5. O SIMULADOR

Neste capítulo é discutida a proposta do simulador, abrangendo a arquitetura proposta para o seu desenvolvimento e as responsabilidades atribuídas a cada módulo.

5.1 ARQUITETURA GERAL

A arquitetura do simulador, ilustrada na Figura 7, foi pensada com foco na flexibilidade e facilidade de desenvolvimento dos módulos. Para cada módulo foram atribuídas algumas funções que trabalham em conjunto com outros módulos para que o resultado da simulação seja alcançado.

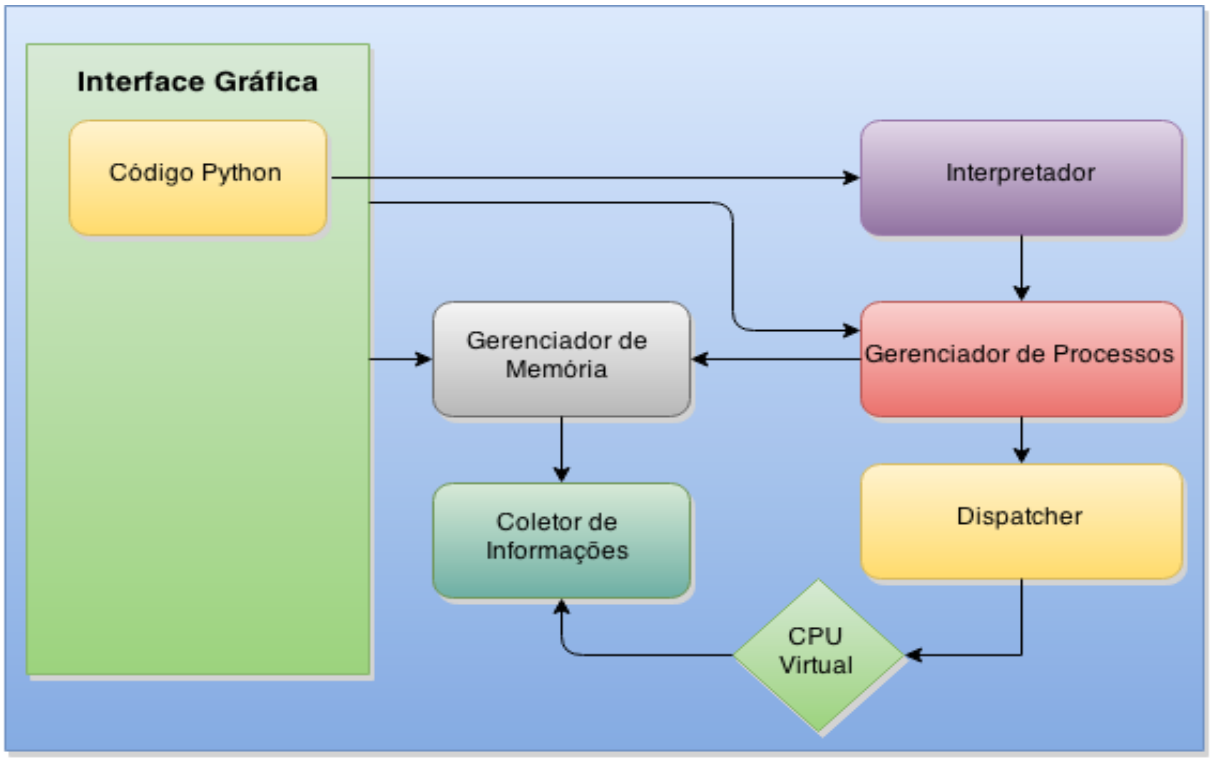

Figura 7: Arquitetura geral do simulador

O funcionamento do simulador se apoia em um programa escrito em linguagem Python, que de agora em diante será denominado programa do usuário. Este programa contém o algoritmo de gerência dos processos, suas funções são dizer como, quando e onde os processos estarão durante a simulação. O simulador, por outro lado, tem a função de prover uma infraestrutura para que os programas do usuário possam ser executados, testados e visualizados a partir de estatísticas, e o usuário ainda pode interagir com essas informações, criando, suspendendo e finalizando processos.

A maior parte da arquitetura é voltada para o gerenciamento de processos. O módulo gerenciador de memória virtual encontra-se mais isolado dentro da arquitetura geral, pois ele apenas servirá requisições feitas pelo gerenciador de processos. Por exemplo, quando um processo for selecionado para execução, o gerenciador de memória virtual, de acordo com suas políticas e o perfil do novo processo, deverá buscar as páginas de memória necessárias para aquele processo.

5.1.1 Interpretador

O interpretador é, na verdade, um módulo contendo um ambiente de execução da linguagem Python junto de uma API disponibilizada pelo simulador para que seja possível criar, alterar e testar algoritmos de escalonamento que serão executados pelo resto do simulador. Este é o módulo mais importante do simulador, pois ele provê uma maneira de o usuário interagir com o simulador de modo muito flexível, não limitando as possibilidades de simulações somente a algoritmos pré-definidos pelo simulador.

Uma das funções mais importantes deste módulo é oferecer uma API eficiente e clara para que os programas escritos em Python possam utilizá-la para desenvolver os algoritmos de escalonamento, por isso se faz necessário definir funções como: criar filas onde processos poderão ser adicionados, funções para adicionar processos às filas, ou funções que são chamadas quando o usuário cria um

processo diretamente da interface do simulador bem como quando são eliminados, funções para recuperar dados do coletor de informações, funções para definir temporalizadores que serão utilizados por algoritmos de escalonamento de tempo compartilhado, entre outras que serão ainda definidas.

5.1.2 Gerenciador de Processos

O gerenciador de processos fica encarregado de implementar as políticas de escalonamento, bem como servir de ponte para o interpretador, fornecendo uma API que o usuário possa utilizar para criar seus próprios algoritmos de escalonamento. O próprio simulador pode se beneficiar desta API para disponibilizar os algoritmos de escalonamento já embutidos.

É atribuição também do gerenciador de tarefas do simulador, controlar os eventos temporais da simulação. Embora o programa que esteja rodando possa controlar o tempo em que as tarefas serão trocadas, fica a cargo do usuário definir o fator de velocidade em que os eventos ocorrerão, isto é, se o programa configurou para um evento temporalizado ocorra em um tempo t, o fator definido pelo usuário faria com que esse tempo seja definido por t multiplicado pelo fator.

Outras funções atribuídas ao gerenciador de processos são: notificar o programa do usuário que o próprio requisitou a criação de um novo processo contendo um certo perfil de tarefa como *CPU-bound* e *IO-bound*, ou até mesmo a prioridade que um determinado processo pode ter. O mesmo ocorre quando o usuário solicita que um determinado processo seja suspenso. Dessa maneira o gerenciador de tarefas notificará o programa do usuário para que o algoritmo definido pelo usuário possa interpretar essa informação da sua maneira, podendo impactar em suas políticas de escalonamento. Este processo também ocorrerá quando o usuário requisitar que uma determinada tarefa seja destruída.

O gerenciador de processos é também essencial na gerência das filas que o programa de usuário pode criar, filas estas que são onde os processos podem ser

adicionados ou removidos, ficando a cargo do programa do usuário definir as funções de cada fila. O programa pode ainda criar um número arbitrário de filas.

A comunicação que ocorre entre o gerenciador de processos e o programa do usuário por meio do interpretador é importante não apenas por fornecer uma maneira do usuário controlar seus processos, mas também para que o estado da simulação seja demonstrado na interface gráfica do usuário em tempo de simulação. Por exemplo, quando um processo é criado e colocado em uma fila previamente criada, é necessário que o simulador exiba esse processo na fila a qual ele está associado e quando o processo é destruído é necessário que ele seja removido desta fila, e estas informações são refletidas na interface gráfica.

5.1.3 Gerenciador de Memória

O gerenciador de memória ficará responsável por implementar espaços de endereçamento de memória virtual para os processos, bem como simular uma memória física onde as páginas utilizadas pelos mecanismos de paginação da memória virtual serão alocadas e liberadas.

Os processos criados no simulador contam com dois tipos de paginação: por demanda e antecipada. Na paginação por demanda, algumas páginas serão carregadas na memória, e quando uma página que estiver ausente for referenciada, causando um *page fault*, o gerenciador de memória ficará responsável por implementar a política de busca de páginas que a buscará na memória secundária e a levará para a memória principal. Na paginação antecipada, todas as páginas do processo são carregadas na memória.

Outra função do gerenciador de memória virtual é implementar a política de alocação de páginas, tais como alocação variável e alocação fixa. Essas políticas dizem ao gerenciador quantas páginas um processo pode ter na memória ao mesmo tempo, na alocação fixa como o nome diz, existe um valor máximo pré-definido. No caso da alocação variável não existe limite estipulado, dependendo das limitações

da memória física.

Caso o limite de páginas residentes de um processo seja atingido, então será chamada a função que tem a responsabilidade de trocar as páginas que estão residentes na memória com as páginas que estão na memória secundária. Essas trocas são feitas utilizando políticas de substituição de páginas, tais como: páginas menos frequentemente utilizadas (LFU – *Least Frequently Used*), página menos utilizadas recentemente (LRU – *Least Recently Used*), entre outras.

5.1.4 Dispatcher

No simulador, o *dispatcher* será encarregado de fazer a troca entre o processo corrente para o próximo processo selecionado em estado de pronto pelo gerenciador de processos para entrar em execução na CPU virtual, realizando a troca virtual do contexto de software e hardware. No *dispatcher* são coletadas informações tais como: o número de vezes que um processo foi selecionado, o seu tempo de *turnaround*, tempo de espera, entre outras, que serão posteriormente utilizadas pelo coletor de informações para disponibilizá-las para as outras partes do sistema.

Segundo Machado e Maia (2007), o *dispatcher* é uma rotina importante na gerência do processador, ficando responsável pela troca de contexto dos processos após o escalonador determinar qual processo ocupará o processador. O período gasto na troca de um processo para outro é denominado como latência do *dispatcher*.

Em um sistema operacional real, o *dispatcher* faz a troca do contexto de software e de hardware de um processo para outro. Essas informações são salvas no bloco de controle de processo (ou PCB – *Process Control Block*) atual, para que mais adiante, quando o processo for novamente selecionado para entrar em estado de execução, elas possam ser restauradas, fazendo com que o processo continue exatamente do ponto em que foi interrompido.

5.1.5 CPU Virtual

A CPU virtual simula algumas ações que o processo corrente poderia solicitar ao sistema operacional, tais como operações de entrada e saída. No simulador essas operações são definidas através do perfil da tarefa no momento de sua criação, onde a tarefa pode ser definida como *CPU-bound*, sendo estes os processos que fazem uso intensivo do processador ou *IO-bound*, nestes os processos realizam diversas operações de entrada e saída, assim fazendo pouco uso do processador e sendo substituída por outro processo na CPU virtual tão logo uma operação de entrada e saída comece.

5.1.6 Coletor de Informações

O coletor de informações é um módulo que fica encarregado de obter informações da maior parte das situações que ocorrem dentro do simulador, tais como novos processos criados, troca de processos, operações de entrada e saída, etc. Essas informações por sua vez serão utilizadas pelo simulador para serem mostradas ao usuário do mesmo, permitindo verificar de forma gráfica esses valores em gráficos e números. Por exemplo, a possibilidade de visualizar os processos que obtiveram acesso a CPU virtual em um gráfico linear em função do tempo ou mesmo visualizar a demanda de páginas de memória virtual que um processo teve durante seu tempo de vida.

Este módulo é um dos elementos mais importantes para que o usuário interaja com o simulador, possibilitando a análise de casos utilizando simulações e colhendo seus resultados, tanto de maneira textual e visual através de gráficos em função do tempo de execução dos processos criados.

5.2 A INTERFACE DO SIMULADOR

Nesta seção apresenta-se as partes que compõem a interface gráfica do simulador.

A janela principal do simulador é demonstrada na Figura 8. Ela é composta de uma barra de ferramentas para que se possa ter acesso rápido as funções do menu, uma barra de menus com todas as opções do simulador, um editor de código Python com suporte a realçamento de sintaxe e o *log* de mensagens geradas pelo interpretador Python. Na barra de ferramentas encontra-se também dois componentes importantes, o relógio do sistema em que todas as operações simuladas são baseadas, e ao seu lado tem-se uma barra onde se pode deslizar uma agulha para determinar o tempo de duração real de uma unidade de tempo dentro do simulador, variando de 50 a 1000 milissegundos.

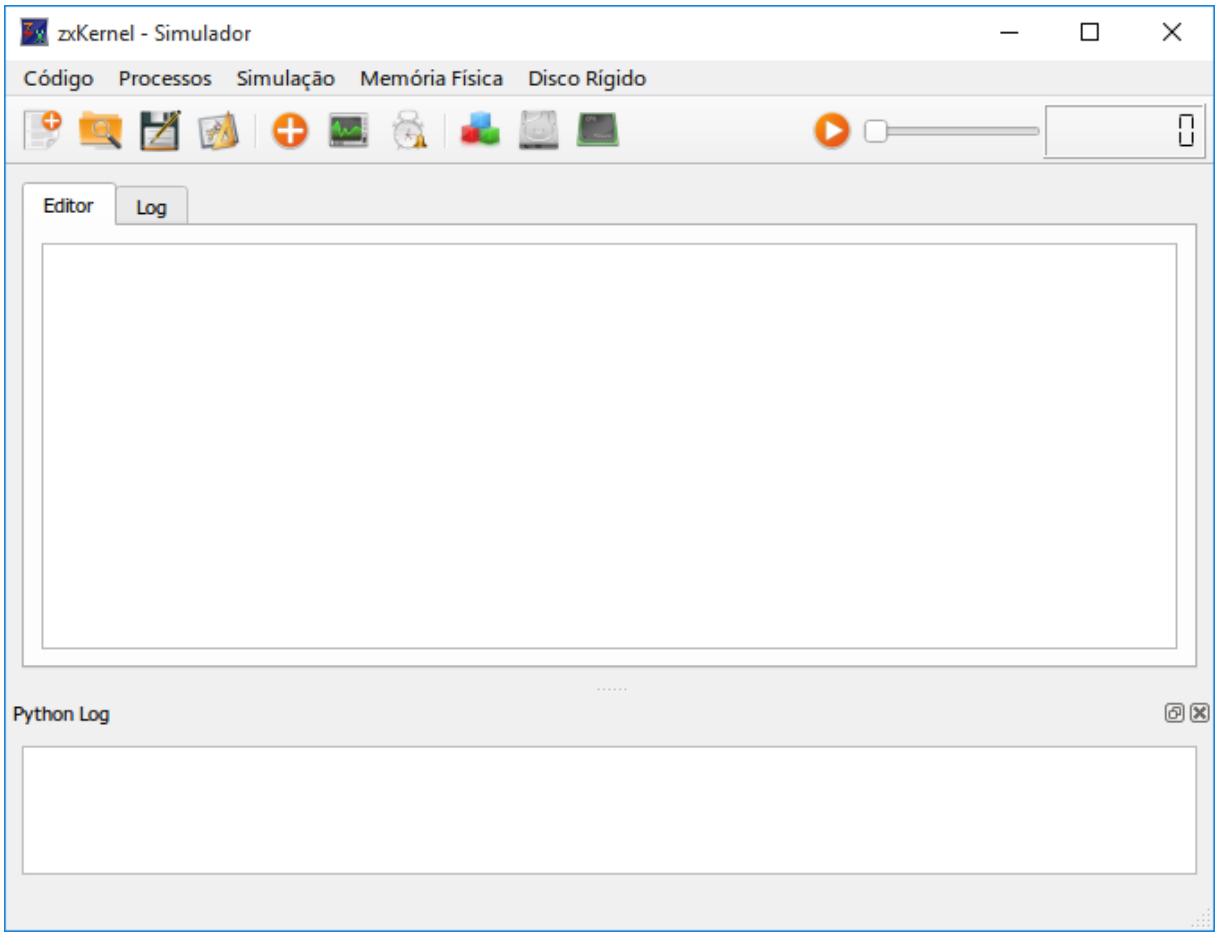

Figura 8: Janela principal do simulador

Na janela principal, além do editor de código Python, temos outra aba chamada Log, que é o lugar onde as informações sobre a simulação são gravadas em forma de texto. Isto é útil pois na maior parte dos componentes tais informações são dadas de maneira visual através de gráficos sobre uma linha do tempo. Desta forma torna-se mais fácil encontrar a informação sobre qual processo estava rodando em um tempo t. As informações textuais e visuais se complementam para formarem uma "grande imagem" sobre o funcionamento de um escalonador, permitindo ao usuário conseguir extrair informações gerais sem maiores análises, apenas interpretando os gráficos.

Na Figura 9 tem-se a janela para a criação de processos. O único meio de se criar um processo na simulação é através desta janela, não sendo possível fazer isso nem mesmo através do código Python pela API exportada.

As opções para a criação de um novo processo são muito simples: o usuário pode escolher o tipo de processo entre CPU-Bound e IO-Bound; dar um nome ao processo (o simulador gera nomes sequenciais automaticamente caso não seja necessário); definir a quantidade de *frames* de memória física que o processo utilizará em sua simulação; o usuário pode também definir a prioridade, no entanto este é apenas um valor que o escalonador do usuário pode ou não utilizar, sendo possível ter acesso a ele na API exportada.

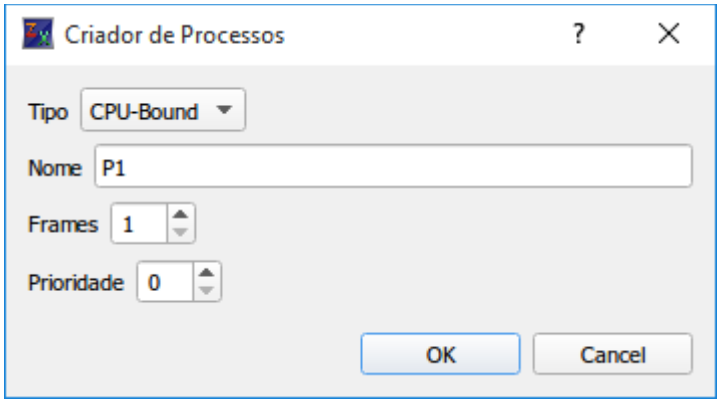

Figura 9: Janela de criação de processos

Para monitorar os processos criados, o usuário necessita utilizar o utilitário

denominado “Monitor de Processos”, a Figura 10 demonstra a interface do utilitário. Ele é composto de uma lista contendo todos os processos criados, mais o processo ocioso do sistema, que é chamado de IDLE. Nesta lista existem apenas informações básicas sobre o processo, como seu número de identificação (PID), seu tipo e nome. O processo será realçado com a cor amarela se estiver suspenso no momento. Para obter mais informações sobre o processo basta que o usuário realize um duplo clique sobre o processo desejado, assim o simulador apresentará mais informações sobre o processo em outra janela representada pela Figura 11.

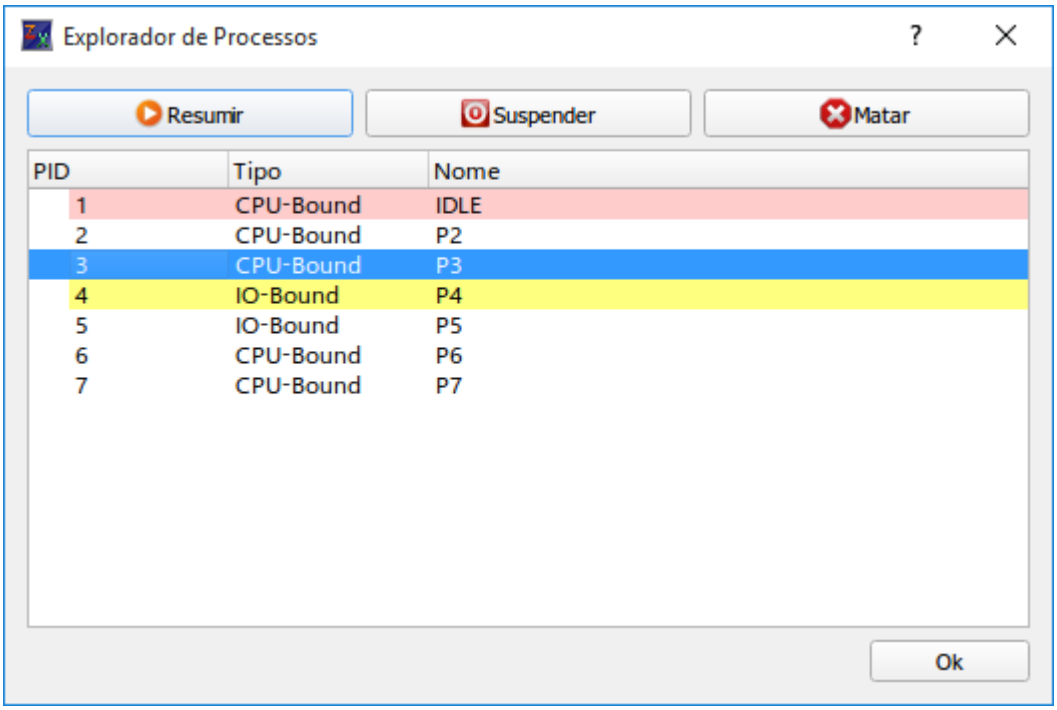

Figura 10: Explorador de processos

Na janela de informações individuais sobre um processo da Figura 11, temos: o número de identificação do processo, este número é único dentro de toda a simulação; o estado atual do processo (executando, pronto, suspenso, zumbi); a cor que representa o processo, esta cor é gerada automaticamente pelo simulador, sendo útil para o utilitário de visualização de memória; nome do processo; tipo do processo; o campo selecionado indica o número de vezes que o processo foi selecionado e entrou em execução ocupando a CPU; o tempo de CPU indica o total

de unidades de tempo que o processo durante seu tempo de vida esteve na CPU; o *turnaround* é o tempo de vida do processo; e por fim o campo tempo de espera é o total de unidades de tempo que o processo permaneceu em estado suspenso – este dado pode ser uma ótima fonte de informação sobre como se comporta um processo permitindo implementar políticas de escalonamento mais dinâmicas.

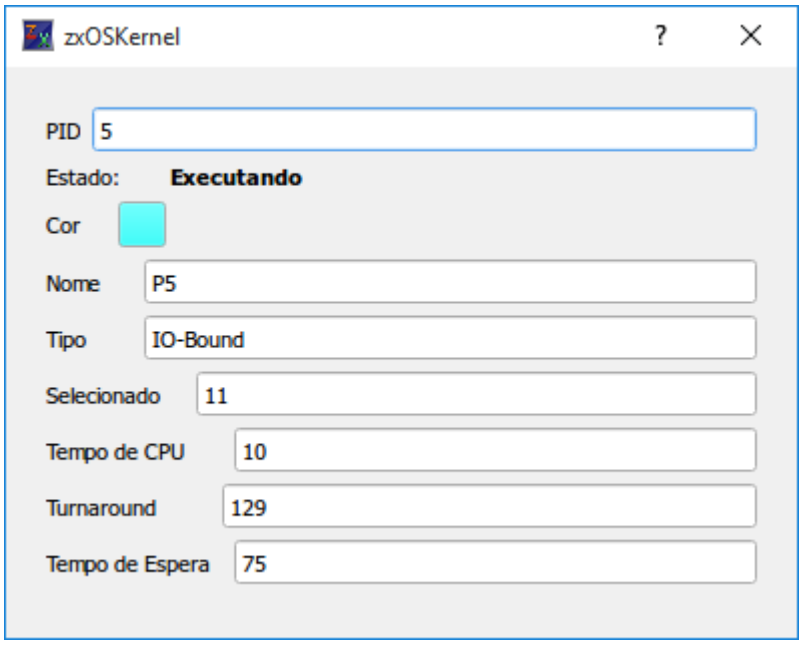

Figura 11: Informações do processo

Para a visualização em tempo real ou do histórico da simulação dos processos que estão ou estiveram em execução, o usuário pode utilizar o utilitário representado pela Figura 12. Neste utilitário existe apenas um gráfico que é gerado em tempo real através das informações obtidas pelo coletor de informações. Este gráfico é dado pelo nome do processo em função do tempo, isto é, no eixo Y temos os nomes dos processos que estiveram presentes na simulação, e no eixo X temos a linha do tempo. Este gráfico permite interatividades como: deslizar pelo histórico desde o tempo 0 da simulação e aumentar ou diminuir a escala de tempo para que se consiga visualizar uma maior fatia de tempo, que por padrão é fixada em 10 unidades de tempo.

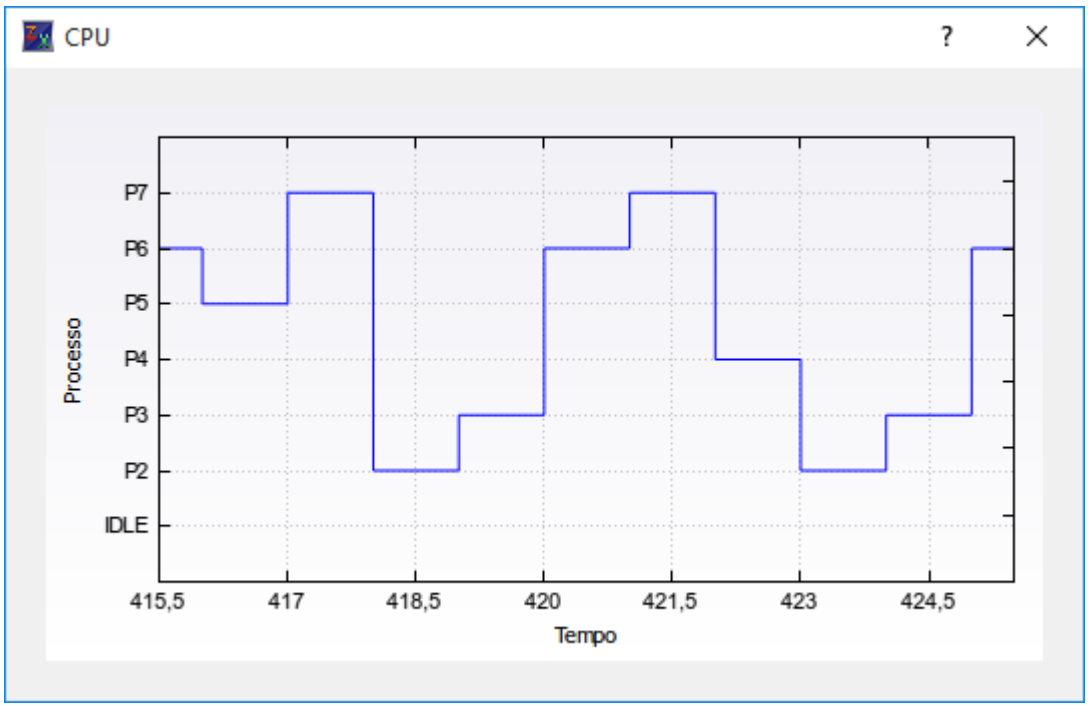

Figura 12: Gráfico de execução de processos na CPU

O visualizador de memória física representado pela Figura 13, exibe de maneira visual quais quadros da memória física estão alocados e quais estão livres. No total são 256 quadros de memória que o simulador disponibiliza. A quantidade de quadros alocadas é encontrada no campo *Frames* da janela.

Um quadro da memória alocado é colorido com a mesma cor atribuída ao processo em seu momento de criação pelo usuário. Para obter o nome do processo que atualmente é dono de um quadro de memória, basta que o usuário pressione qualquer botão do mouse sobre o quadro e assim será exibido o nome do processo no campo Processo na parte inferior da janela.

Junto à visualização da memória física existe um gráfico de carga da memória durante a simulação, este é um gráfico temporal que registra o percentual de uso da memória. Assim como o gráfico da CPU na Figura 12, permite as mesmas opções de interatividade: fazer o deslizamento do gráfico no tempo e aumentar ou diminuir a escala do tempo.

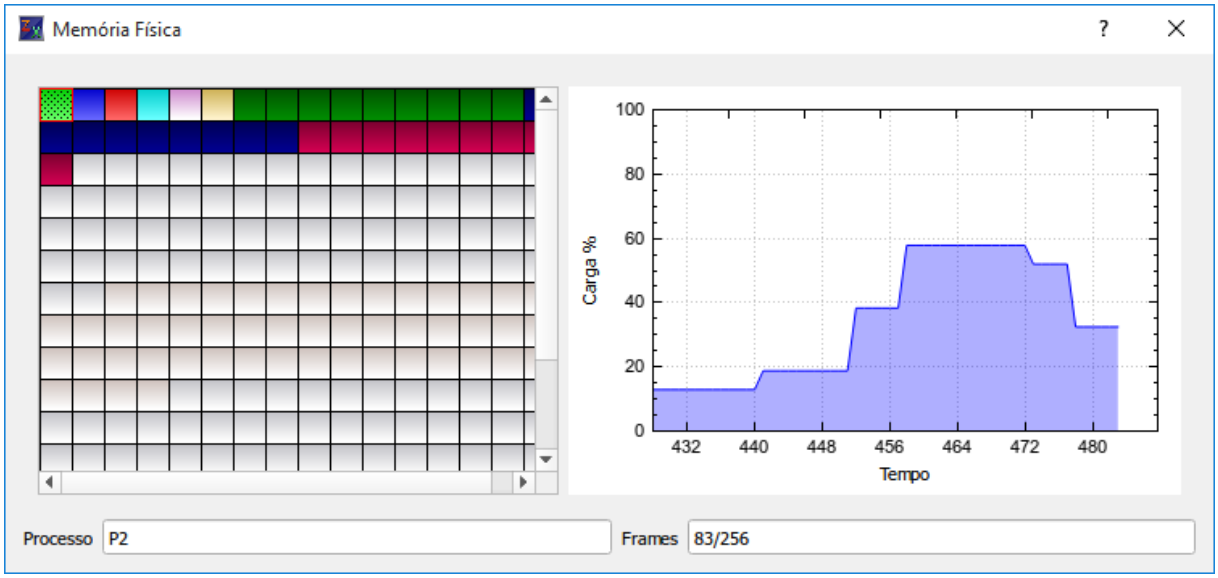

Figura 13: Visualização da memória física

Na janela representada pela Figura 14, é possível monitorar as operações de E/S que os processos do simulador estão realizando ou já realizaram. É possível também visualizar o processo que requisitou a operação atual, bem como a fila de requisições que serão servidas utilizando o critério FIFO.

O simulador apenas suporta duas operações no disco rígido: leitura e escrita, no entanto no mundo real um disco rígido pode realizar inúmeras operações que podem impactar na velocidade em que as chamadas são processadas. Uma vez que o intuito desse simulador está mais focado no campo de gerência de processos e memória, a implementação desse componente é limitada apenas a leitura e a escrita.

No gráfico disposto no monitor de operações do disco rígido, é possível visualizar as operações que os processos requisitaram ao disco de maneira temporal. Este é um gráfico de duas linhas, uma representando as operações de escrita (em vermelho) e outra as operações de leitura (em azul). No eixo Y do gráfico temos os nomes dos processos e no eixo X temos o tempo em que as operações ocorreram. Este gráfico permite as mesmas interatividades já relatadas nas janelas de componentes anteriores.

É possível visualizar a operação atual com os dados de seu processo, o tipo de operação e o tempo até a operação estar completa. O tempo restante de uma operação é representado por uma barra de progresso. Quando a operação atual chegar aos 100% estará terminada, assim o disco rígido virtual começará uma nova operação se ainda existir requisições na fila de espera, caso contrário o disco rígido entrará em modo ocioso.

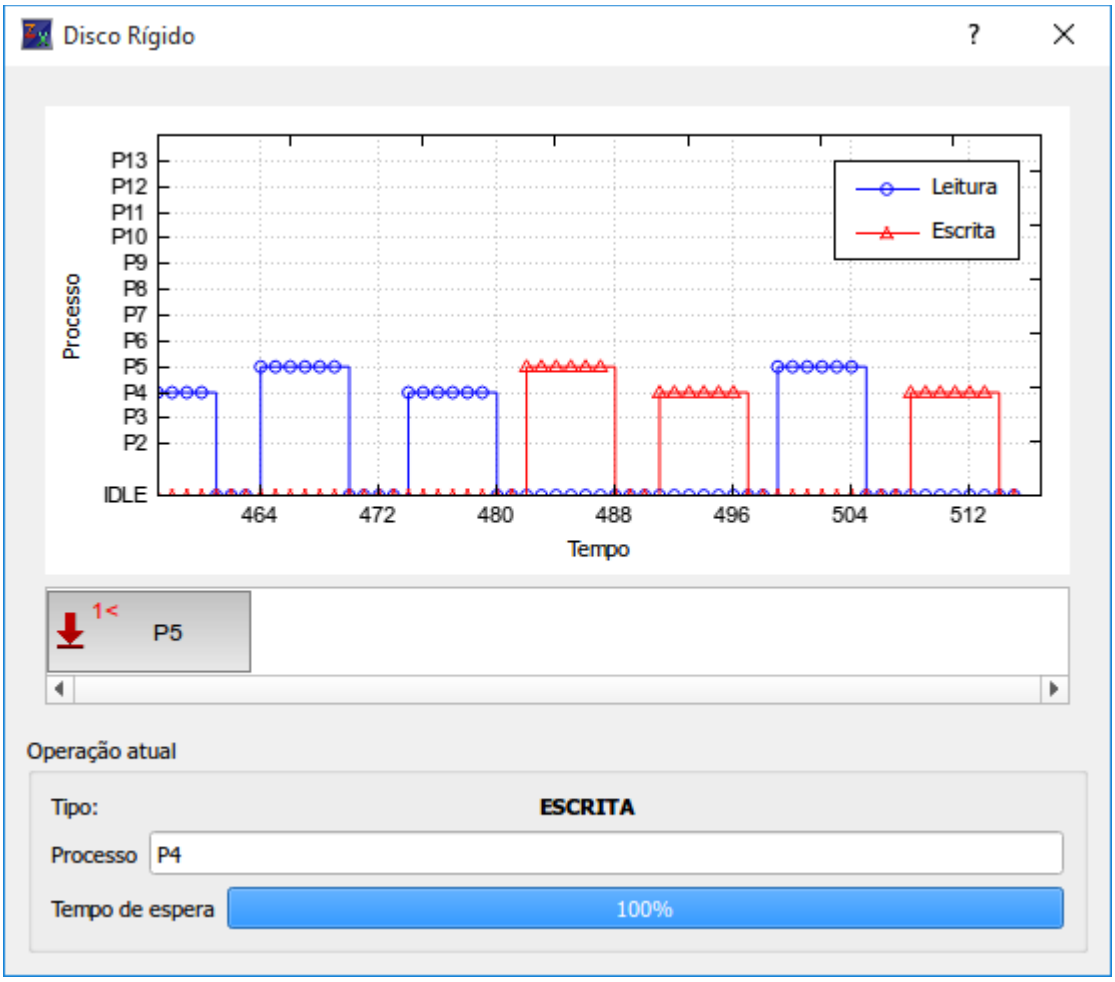

Figura 14: Visualização das operações de E/S no disco

Para monitorar o escalonamento, o usuário utilizará a janela representada pela Figura 15. Nesta janela é possível verificar qual processo está atualmente em execução junto de alguns dados seus que são relevantes para o contexto de

escalonamento.

É possível visualizar as filas de processos registradas pelo escalonador, junto do seu percentual de carga em relação a quantidade de processos atualmente no simulador. As filas e seus itens são desenhados de acordo com a cor que o usuário definiu para a fila no momento de sua criação. As filas são atualizadas em tempo real, bem como o gráfico que é encontrado logo abaixo das filas.

O gráfico de linhas da janela de monitoramento do escalonador é desenhado em tempo real contendo todas as filas que o usuário registrou. Cada linha tem a mesma cor da sua fila correspondente, facilitando a visualização. Neste gráfico é possível verificar como se comportaram as filas durante uma simulação em relação a distribuição da carga nas filas. Este gráfico permite as mesmas interatividades já relatadas para outros gráficos em outras janelas do simulador.

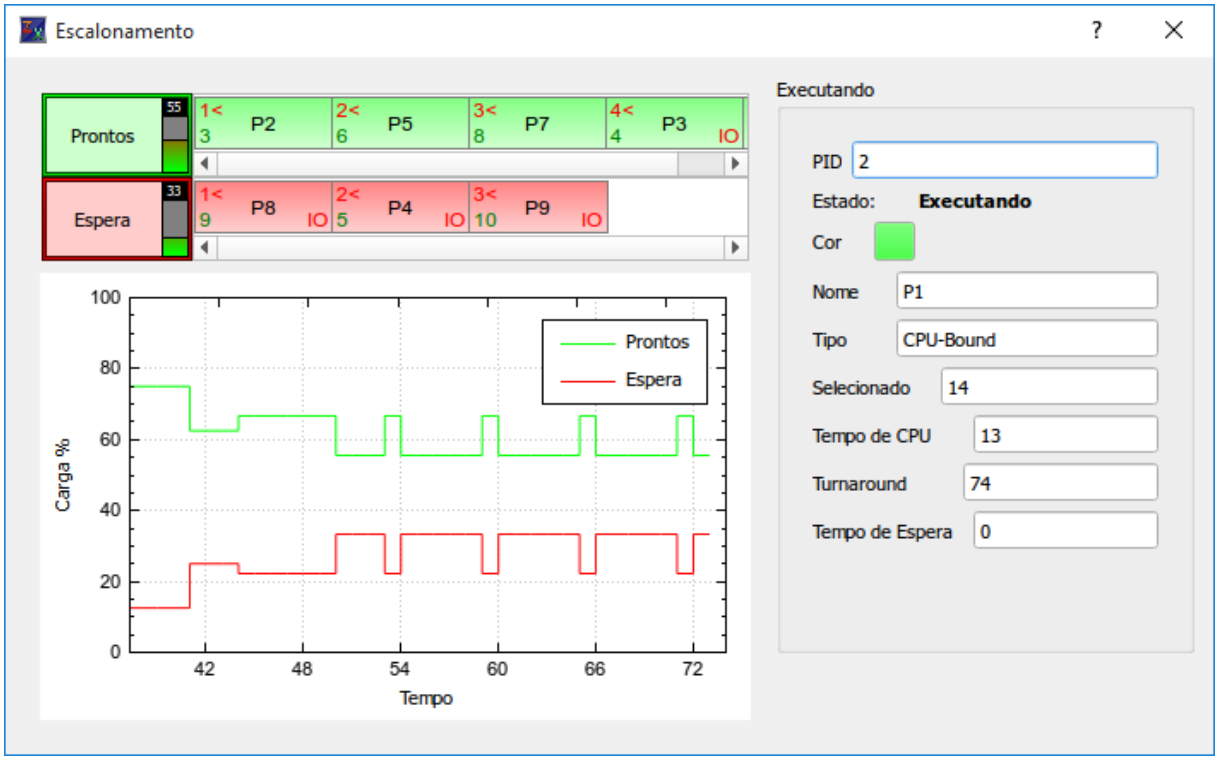

Figura 15: Informações sobre o escalonamento

5.3 A API DO SIMULADOR

Nesta seção é feita a descrição da API que o simulador exporta para o programa do usuário escrito em código Python. A API foi pensada para ser de fácil memorização e baixa complexidade, contando apenas com funções que estejam dentro do domínio da aplicação: escrever escalonadores de processos.

As definições dos métodos são descritas em listagens, começando com o nome do método seguido de seus parâmetros e seu tipo de retorno se houver necessidade. A estrutura geral é como segue:

nome_método(arg0: tipo0, …, argN: tipoN) → tipoRetorno.

5.3.1 Classes exportadas

5.3.1.1 Classe kprocess

A classe kprocess representa a abstração de um processo dentro do simulador, tendo o mesmo conceito discutido na seção 3.1. Embora o objeto se comporte como qualquer outro dentro da linguagem Python, não é possível criar processos diretamente pelo código Python instanciando diretamente esta classe, essa tarefa sempre fica a cargo do usuário através da interface gráfica, onde ele pode definir os parâmetros corretos e o como vai se comportar cada processo. A única maneira de conseguir acesso a esses objetos é através do escalonador que os recebe como parâmetros, assim podendo adicioná-los as suas filas ou realizar outras operações. A seguir temos a listagem de todos os métodos que podem ser chamados em uma instância de um objeto kprocess:

- **state() → stateType**: Este método retorna o estado atual do processo,

podendo ser suspenso, pronto, executando e zumbi, como descrito no item 5.3.2.1.

- **set_state(state: stateType)**: Método responsável por alterar o estado atual do processo. É importante ressaltar que esse método apenas altera a indicação do estado do processo, já o comportamento do mesmo está diretamente ligado a lógica desenvolvida pelo escalonador do usuário, dessa forma utilizar essa função para suspender um processo não irá suspendê-lo, apenas mudar sua indicação de estado. Como descrito no item 5.3.2.1.

- **name() → str**: Método que retorna o nome que o usuário deu ao processo no momento de criação.

- **set_name(name: str)**: Método utilizado para alterar o nome dado pelo usuário ao processo.

- **priority() → int**: Retorna a prioridade atual do processo, se a mesma não for alterada pelo escalonador o valor retornado será o dado pelo usuário no momento de criação.

- **set_priority(prio: int)**: Altera a prioridade do processo. Esse valor apenas é interessante para os escalonadores que operam sobre a prioridade, pois ele não é utilizado por padrão no simulador.

- **data() → scheduler_data**: Retorna um objeto herdado da classe **scheduler_data** contendo os dados privados inseridos pelo escalonador. Este campo é exclusivamente utilizado pelo escalonador para inserir dados adicionais que possam ser resgatados nos processos.

- **set_data(data: scheduler_data)**: Define o atributo data com um objeto que tem como classe base scheduler_data.

- **wait_time() → int**: Retorna o tempo total em espera do processo.

- **turnaround() → int**: Retorna o tempo de *turnaround* do processo.

- **scheduled() → int**: Retorna o número de vezes que o processo foi

selecionado para execução.

- **type() → processType**: Retorna o tipo de processo, se é *IO-Bound* ou *CPU-Bound*. Segundo o item 5.3.2.2.

5.3.1.2 Classe kprocessqueue

A classe kprocessqueue é a representação de uma fila de processos dentro de um sistema operacional no simulador. Seu funcionamento não difere muito de uma fila FIFO, ficando a cargo dos escalonadores realizarem o gerenciamento desta fila.

A única forma de se conseguir um objeto deste tipo é através do método estático definido na classe ***scheduler*** chamado *queue* que retornará uma instância de um kprocessqueue.

- **empty() → bool**: Retorna verdadeiro se a fila estiver vazia.
- **name() → str**: Retorna o nome atual da fila.
- **set_name(name: str)**: Define o nome da fila que será exibido na interface gráfica do simulador na janela de escalonamento. É sempre útil definir um nome para fila uma vez que seu monitoramento se tornará mais simples nos gráficos estatísticos gerados pelo simulador.
- **set_color(r: int, g: int, b: int)**: Define uma cor para a fila. Esta cor será utilizada pelo simulador para desenhar seus elementos.
- **push_back(proc: kprocess)**: Coloca um processo no fim da fila.
- **push_front(proc: kprocess)**: Coloca um processo no início da fila.
- **pop_front() → kprocess:** Retira um processo do início da fila e o retorna como retorno do método.
- **pop_back() → kprocess**: Retira um processo do final da fila e o devolve

como retorno do método.

- **processes() → list**: Retorna uma lista iterável dentro do código python com todos os processos. É possível fazer iteração nesta lista da mesma maneira que se faria em uma coleção do código Python, ex: for proc in fila.processes().

- Operador **in**: É possível utilizar o operador especial ***in*** da linguagem Python para verificar se um dado processo está presente na fila.

5.3.1.3 Classe scheduler

A classe scheduler é utilizada como base para a criação de um novo escalonador para o simulador. Esta classe possui alguns métodos que devem ser implementados nas classes derivadas, pois tais métodos são chamados no momento da simulação em operações como: adicionar novos processos, suspender processos, remover processos e selecionar o próximo processo. Toda a lógica e algoritmo do escalonador tem que ser desenvolvido em torno de cinco métodos descritos abaixo:

- **add(proc: kprocess)**: Este método é chamado quando o usuário, através da interface gráfica do simulador, cria um novo processo. É importante ressaltar que neste momento o escalonador deve salvar o objeto do processo em alguma estrutura de sua preferência, caso contrário ele não mais terá acesso a esse processo.

- **remove(proc: kprocess)**: Método chamado quando o usuário, requisita a remoção de um processo através da interface gráfica. A responsabilidade do escalonador na chamada deste método é apenas remover todas as suas referências a esse processo, pois ele não poderá ser mais visto pelo usuário no simulador.

- **suspend(proc: kprocess)**: Este método é invocado quando o usuário requisita a suspensão de um processo através da interface gráfica. Ocorre

também quando o processo for do tipo IO-Bound e houver uma simulação de operação de E/S, neste caso o processo precisa esperar a operação de E/S terminar sendo este suspenso e resumido quando a operação terminar.

- **resume(proc: kprocess)**: Método chamado quando houver a necessidade de um processo previamente suspenso poder ser executado novamente. Essa chamada pode ter duas origens: o usuário interagindo com o simulador diretamente na interface gráfica, ou quando o processo estiver esperando por uma operação de E/S e ela estiver completa.
- **schedule()**: Método chamado a cada unidade de tempo no simulador. É neste método que o escalonador deve decidir qual será o próximo processo a ser executado e desenvolver todo seu algoritmo.

5.3.1.4 Classe scheduler_data

A classe scheduler_data é uma classe abstrata utilizada para que o escalonador possa inserir seus próprios dados em cada processo, podendo assim gerenciá-los sem a necessidade de criar estruturas adicionais para manter controle sobre os processos de suas listas. Esta classe não possuí nenhum método ou atributo, é apenas utilizada como forma de marcação para indicar que certos dados possam ser inseridos em um processo.

5.3.2 Enumerações exportadas

Algumas enumerações são exportadas para o interpretador Python. Essas enumerações são necessárias para tornar mais fácil a leitura dos códigos de escalonadores e reduzir as chances do usuário errar caso ele tivesse que utilizar números diretamente.

5.3.2.1 Enumeração state

Esta enumeração contém os valores possíveis para o estado de um processo que são suportados pelo simulador. Não é possível adicionar novos valores.

- **Running**: Indica que o processo está atualmente executando.
- **Ready**: Indica que o processo está pronto para execução.
- **Suspended**: Indica que o processo está suspenso, ou esperando por alguma operação ser realizada.
- **Zombie**: Indica que o processo está morto, esperando para ser removido.

5.3.2.2 Enumeração processType

Enumeração que especifica o tipo do processo. Existem apenas dois tipos de processos, os IO-Bound e CPU-Bound. Essas categorias são importantes para que o escalonador possa conhecer melhor os processos.

- **CPUBound**: Categoria de processo que realiza intenso processamento de CPU.
- **IOBound**: Categoria de processo que realiza muitas operações de E/S.

5.3.3 Funções exportadas

- **set_scheduler(sched: scheduler)**: define o escalonador que o simulador utilizará na sua simulação. É importante que o escalonador passado não seja uma objeto temporário, deve ser uma variável com nome como qualquer outra em Python, caso contrário o escalonador será destruído pelo interpretador Python assim que a chamada ao método se encerrar.
- **get_time() → int**: retorna o tempo atual da simulação. Os escalonadores devem se basear neste valor para criar eventos temporais.
- **is_running(proc: kprocess) → bool**: retorna verdadeiro se o processo estiver em execução.
- **is_suspended(proc: kprocess) → bool**: retorna verdadeiro se o processo está atualmente suspenso.
- **is_ready(proc: kprocess) → bool**: retorna verdadeiro se o processo estiver em estado de pronto.
- **is_zombie(proc: kprocess) → bool**: retorna verdadeiro se o processo estiver morto esperando para ser removido pelo simulador.
- **cpu.running_process() → kprocess**: retorna o processo que está atualmente executando na CPU virtual.
- **cpu.switch_to(proc: kprocess):** pede a cpu virtual para executar a troca de contexto para um novo processo.
- **log.info(msg: str)**: grava uma mensagem de informação.
- **log.alert(msg: str)**: grava uma mensagem de alerta.
- **log.error(msg: str)**: grava uma mensagem de erro.
- **log.debug(msg: str)**: grava uma mensagem de depuração.

- **is_idle(proc: kprocess)** → **bool**: retorna se o processo é o tempo ocioso do sistema.

- **idle()** → **kprocess**: retorna o processo de tempo ocioso do sistema. Não é possível suspender, alterar, ou realizar qualquer outra operação neste processo, ele apenas deve ser utilizado quando não houver mais processos para serem executados.

- **scheduler.queue()** → **kprocessqueue**: instância uma nova fila de processos para o escalonador. Esta é a única maneira de se conseguir uma nova fila pelo código escrito pelo usuário.

- **scheduler.register_queue(queue: kprocessqueue)**: função utilizada para registrar a exibição de uma fila na interface gráfica, isto é, quando o usuário quiser exibir uma fila na interface gráfica, ele tem que necessariamente registrá-la através dessa função, caso contrário a fila não será exibida na janela de escalonamento.

6. DESENVOLVIMENTO DO ESCALONADOR CIRCULAR

Neste capítulo é elaborado o desenvolvimento de um escalonador circular utilizando a API do simulador como demonstração do seu uso.

O escalonador circular desenvolvido neste capítulo segue os mesmos critérios do algoritmo de escalonamento demonstrado no seção 3.3.4.1. Um escalonador preemptivo, onde um processo recebe uma fatia de tempo do processador na qual pode permanecer em execução denominada *quantum* e quando esta fatia expira o processo sofre preempção por tempo. Se o processo for IO-Bound ele provavelmente não utilizará todo seu quantum, deixando o processador livre para o próximo processo ser executado.

Este escalonador utiliza apenas duas filas: fila de processos prontos e fila de processos em espera. Os processos da fila de prontos são selecionados com o critério FIFO, isto é, o primeiro a entrar é o primeiro a ser executado. Do ponto de vista de justiça com os processos pode-se dizer que este algoritmo é totalmente justo, uma vez que todos os processos recebem as mesmas oportunidades de execução e tempo de CPU. No entanto, este escalonador não é muito eficiente, uma vez que os processos IO-Bound por utilizarem pouco tempo de CPU terão menos tempo de execução na CPU e passarão a maior parte do tempo de sua vida ou na fila de espera ou na fila de prontos.

Nesta implementação do escalonador serão utilizadas duas variáveis de controle em cada processo: o *quantum* e quando o processo expirará seu tempo de execução. É muito importante definir eventos temporais em termos do tempo global. Poderíamos utilizar apenas o *quantum* e a cada vez que o simulador chamasse a função *schedule* do nosso escalonador iriamos decrementando este valor até chegar a zero. Porém, esta não é uma boa estratégia, pois não é garantido que o método *schedule* será chamado apenas uma vez. As vezes o próprio escalonador necessita chamar

schedule para forçar a troca para outro processo. Este cenário é bem comum quando queremos suspender um processo que está em execução.

É necessário criar uma classe que herda de *scheduler_data* para as informações privadas do escalonador que serão inseridos em cada processo. A Figura 16 mostra a implementação da classe round_robin_data, sendo que nesta classe tem-se dois atributos: o quantum e o tempo de expiração.

```
 7  # Dados utilizados pelo escalonador em cada processo
 8  class round_robin_data(scheduler_data):
 9      def __init__(self):
10          scheduler_data.__init__(self)
11          self.quantum = int()
12          self.expireAt = int()
13
```

Figura 16: Dados privados do escalonador circular

O próximo passo é criar a classe do escalonador que herda da classe exportada *scheduler*. Na Figura 17 tem-se a implementação do construtor do escalonador. No construtor são criadas as duas filas necessárias (processos espera e processos prontos), uma variável para indicar que o reescalonamento é necessário, isto é, forçar a escolha de um novo processo. Utilizamos o método *register_queue* para registrar essas filas na janela de escalonamento.

```
15  class round_robin(scheduler):
16      def __init__(self):
17          scheduler.__init__(self)
18          # Cria as filas de pronto e espera
19          self.wait = scheduler.queue()
20          self.ready = scheduler.queue()
21          # Flag para forçar o reescalonamento
22          self.need_resched = False
23          # Exibe as filas na interface gráfica
24          self.register_queue(self.ready)
25          self.register_queue(self.wait)
26
```

Figura 17: Construtor do escalonador circular

É necessário implementar os métodos: *add, resume, suspend, remove* e *schedule*; em qualquer classe que derive de *scheduler*. Na Figura 18 tem-se a implementação do método *add* que será invocado quando um novo processo for criado pelo usuário.

Como o escalonamento circular é baseado no critério FIFO, o novo processo será apenas colocado no fim da fila de processos prontos por meio do método *push_back* da fila. É importante também instanciar um objeto contendo as informações privadas utilizadas pelo escalonador. Neste caso essas informações estão na classe *round_robin_data* na Figura 16. Utilizando o método *set_data* da classe kprocess se insere este objeto diretamente no processo.

```
27      # Adiciona um novo processo
28      def add(self, process):
29          data = round_robin_data()
30          process.set_data(data)
31          self.ready.push_back(process)
32
```

Figura 18: Método add para o escalonador circular

O método responsável por suspender um processo é representado pela Figura 19. Este método mudará o estado do processo para *Suspended* e o colocará na fila de processos em espera.

```
33      # Suspende um processo
34      def suspend(self, process):
35          if process in self.wait:
36              return
37
38          process.set_state(state.Suspended)
39          self.wait.push_back(process)
40          # Se estiver rodando é necessário reescalonar
41          if is_running(process):
42              self.need_resched = True
43              self.schedule()
44          self.ready.remove(process)
45
```

Figura 19: Método suspend para o escalonador circular

Uma condição para que essa operação de suspender um processo transcorra normalmente é que o processo não esteja em execução no momento da chamada. Para esses casos o método verifica se o processo está atualmente em execução e se isto for verdade, ele primeiro forçará o reagendamento de um novo processo, marcada a variável *need_resched* como verdadeira, e logo em seguida chamando o método *schedule* que, ao verificar que *need_resched* é verdadeira, selecionará outro processo para execução. A última operação realizada no método é remover o processo da fila de prontos.

O método *resume* representado pela Figura 20 é invocado quando, o usuário através do explorador de processos do simulador, requisitou que o processo continue sua execução ou quando uma operação de E/S que foi requisitada anteriormente pelo processo esteja completa. Este método apenas troca o estado do processo para *Ready*, remove o processo da fila de espera e o recoloca no fim da fila de processos prontos.

```
47      # Resume um processo que estava em estado de espera
48      def resume(self, process):
49          if process in self.ready:
50              return
51          process.set_state(state.Ready)
52          self.wait.remove(process)
53          self.ready.push_back(process)
54
```

Figura 20: Método resume para o escalonador circular

O usuário pode também remover processos em execução no simulador. Neste caso o método *remove* será chamado. Este método para o escalonador circular é exibido na Figura 21.

O código listado apenas elimina as referências ao processo o removendo das filas de prontos e espera. Em seguida muda seu estado para *Zombie*. Assim como na ação de suspender um processo, é necessário verificar se o processo a ser removido é o processo atualmente em execução e é necessário forçar a seleção de um novo processo logo após a remoção do processo.

```
55      # Remove um processo
56      def remove(self, process):
57          self.wait.remove(process)
58          self.ready.remove(process)
59          process.set_state(state.Zombie)
60          # Se estiver rodando é necessário reescalonar
61          if is_running(process):
62              self.need_resched = True
63              self.schedule()
64
```

Figura 21: Método remove para o escalonador circular

O método principal do escalonador chamado *schedule* é apresentado na Figura 22. Nesta figura temos apenas a primeira parte do método para o escalonador circular. A responsabilidade desta parte do código é apenas determinar se é realmente necessário selecionar um novo processo para execução. Se a variável *need_resched* for verdadeira, então o reescalonamento será feito, caso contrário serão feitas as seguintes verificações:

- Se o processo atual não for o processo ocioso do sistema (IDLE), então, obtemos os dados privados que foram inseridos no processo no método *add*, e verificamos se o tempo atual do simulador ultrapassou o tempo limite de execução dado ao processo, ou se, o estado do processo atual não for *Running*. Se uma dessas condições forem verdadeiras, então é necessário reescalonar e o processo atual sofrerá preempção por tempo.

- Se o processo atual for o processo ocioso do sistema (IDLE) e existir processos na fila de prontos, então é necessário reescalonar. O processo ocioso somente deve ser executado quando não mais houver processos para serem selecionados. Uma vez que exista um processo para ser executado e o processo atual for o ocioso é necessário selecionar o processo em espera na fila de prontos para a execução.

```
65      # Reescalona os processos
66      def schedule(self):
67          current = cpu.running_process()
68          if current == None:
69              log.error('Processo atual é inválido')
70              return
71
72          resched = self.need_resched
73
74          if not is_idle(current):
75              data = current.data()
76              if data.expireAt >= get_time() or not is_running(current):
77                  resched = True
78          else:
79              if not self.ready.empty():
80                  resched = True
81
82          self.need_resched = False
83
```

Figura 22: Método schedule do escalonador circular, parte 1

A Figura 23 representa a seleção de um novo processo para execução. O critério para seleção de processos na fila de pronto do escalonamento circular é o FIFO, deste modo o próximo processo a ser executado será o primeiro da fila.

Nesta parte do código é importante ressaltar que é necessário selecionar outro processo. Primeiramente o escalonador retira o primeiro processo da fila de prontos utilizando o método *pop_front* da classe *kprocessqueue*, se este for um processo válido, isto é, não nulo, então será este processo que entrará em execução.

Caso o processo atual não esteja com o estado marcado como em execução, então não será preciso realizar nenhuma outra ação para este processo. No entanto, se o processo estiver em execução, o processo será colocado no fim da fila de processos prontos.

Se existir um próximo processo para ser executado, o escalonador o dará mais 2 unidades de tempo de quantum e calculará o tempo em que o processo sofrerá preempção por tempo para o próximo processo e o colocará em execução mudando seu estado para *Running* e utilizando a função *cpu.switch_to* para realizar a troca de contexto para o novo processo.

Quando não existir um próximo processo na fila de prontos, o escalonador selecionará o processo ocioso do sistema para execução.

```
84          if resched:
85              next = self.ready.pop_front()
86
87              # Apenas mudamos para o estado de pronto se estivermos rodando
88              if is_running(current):
89                  current.set_state(state.Ready)
90                  if not is_idle(current):
91                      self.ready.push_back(current)
92
93              if next != None:
94                  # Se não for IDLE, então damos um novo quantum
95                  if not is_idle(next):
96                      ndata = next.data()
97                      ndata.quantum = 2
98                      ndata.expireAt = get_time() + ndata.quantum;
99
100                 # Muda o estado para rodando e faz a troca de contexto
101                 next.set_state(state.Running)
102                 cpu.switch_to(next)
103                 return
104
105             # Não existem processos para escalonar, troca para IDLE
106             cpu.switch_to(idle())
107
```

Figura 23: Método schedule do escalonador circular, parte 2

Para informar ao simulador que é preciso utilizar o escalonador definido na classe *round_robin*, é necessário primeiramente criar um objeto não anônimo com uma instância da classe *round_robin* e então utilizar a função *set_scheduler* para defini-lo como o escalonador utilizado pelo simulador. Estes passos são representados pela Figura 24.

```
108 # Cria um escalonador e o coloca em uso
109 my_sched = round_robin()
110 set_scheduler(my_sched)
111
```

Figura 24: Criação e definição do escalonador circular como padrão

7. CONCLUSÃO

À medida que o simulador aqui desenvolvido for evoluindo poderá contribuir para tornar as aulas de Sistemas Operacionais nos tópicos de gerência de processos e memória virtual uma tarefa mais dinâmica e mais interativa, uma vez que oferece uma ferramenta onde é possível escrever algoritmos de escalonamento e visualizar informações sobre as simulações feitas sobre estes algoritmos através de gráficos interativos em tempo real, como fora demonstrado no desenvolvimento do presente trabalho. A partir disto, um professor pode desenvolver uma explicação sobre um certo algoritmo de escalonamento para seus alunos e os mesmos seriam desafiados a escrever tal algoritmo para o simulador. Dentro deste cenário, o trabalho de desenvolvimento deste algoritmo pelo aluno o faria compreender melhor o funcionamento dos algoritmos e suas principais deficiências.

O autor acredita que um professor deve sempre buscar tornar suas aulas mais atrativas e dinâmicas, e a utilização da tecnologia da informação pode ser um ótimo meio de se conseguir isto.

O autor considera também que os objetivos propostos pelo presente trabalho foram alcançados, pois as funções inicialmente propostas foram todas desenvolvidas.

Para trabalhos futuros pretende-se implementar o suporte a criação de algoritmos de escalonamento para operações de E/S no disco rígido virtual, tornando o simulador mais abrangente. Boa parte da infraestrutura necessária para que isto ocorra já foi desenvolvida, porém como esse não era o foco deste trabalho não foi possível seguir adiante.

Dificuldades foram encontradas no desenvolvimento deste trabalho, mas elas apenas tornaram mais gratificante conclui-lo.

REFERÊNCIAS BIBLIOGRÁFICAS

FLORES, Cecilia Dias. BEZ, Marta Rosecler. BRUNO, Rosana Mussoi. **O Uso de Simuladores no Ensino da Medicina**. Revista Brasileira de Informática na Educação, vol. 22, num. 2, ano: 2014. Disponível em <http://br-ie.org/pub/index.php/rbie/article/view/2422/2731>. Acesso em 18, março, 2015.

GADELHA, Renê N. S., AZEVEDO, Ryan Ribeiro de, OLIVEIRA, Hilário T. A. de, NEVES, Tiago D., SOUZA, Cleyton C., SILVA, Edilson Leite da. **OS Simulator: Um Simulador de Sistema de Arquivos para Apoiar o Ensino/Aprendizagem de Sistemas Operacionais**. Disponível em <http://www.br-ie.org/pub/index.php/sbie/article/view/1470>. Acesso em 16, junho, 2014.

HECKLER, Valmir. SARAIVA, Maria de Fátima Oliveira. FILHO, Kepler de Souza Oliveira. **Uso de simuladores, imagens e animações como ferramentas auxiliares no ensino/aprendizagem de óptica**. Revista Brasileira de Ensino de Física, vol. 29, num. 2, pág. 267-273, Ano: 2007. Disponível em: <http://www.scielo.br/pdf/rbef/v29n2/a11v29n2.pdf>. Acesso em 18, março, 2015.

JONES, David; NEWMAN, Andrew. **RCOS.JAVA: A SIMULATED OPERATING SYSTEM WITH ANIMATIONS**. Proceedings of the Computer-Based Learning in Science Conference. Rep. Tcheca, 2001.

MACHADO, Francis B. MAIA, Paulo. **Arquitetura de Sistemas Operacionais**, 4 ed. Rio de Janeiro. Editora LTC, 2007.

MAIA, Luiz Paulo. **SOsim: Simulador para ensino de sistemas operacionais**. 2001. 85 p. Dissertação – Universidade Federal do Rio de Janeiro, IM/NCE.

MAZIEIRA, Carlos. **Reflexões sobre o ensino prático de Sistemas Operacionais**. Workshop de Ensino de Informática da SBC, 2002. Disponível em <http://dainf.ct.utfpr.edu.br/~maziero/doku.php/research:publications>. Acesso em 26

de fevereiro, 2015.

REIS, Fabricio Pereira. **TBC-SO/WEB: SOFTWARE EDUCATIVO PARA APRENDIZAGEM DE GERÊNCIA DE PROCESSOS E DE GERÊNCIA DE MEMÓRIA EM SISTEMAS OPERACIONAIS.** Disponível em <http://algol.dcc.ufla.br/~heitor/Projetos/TBC_SO_WEB/tbc_so_web.html>. Acesso em 16, junho, 2014.

SILBERSCHATZ, Abraham. GALVIN, Peter B. GAGNE, Greg. **Operating Systems Concepts.** 7ed. Estados Unidos, JOHN WILEY & SONS. INC, 2005.

SILBERSCHATZ, Abraham. GAGNE, Greg. **Fundamentos de sistemas operacionais**. 8 edição. Brasil, Rio de Janeiro, LTC, 2013.

STROUSTRUP, Bjarne. **The C++ Programming Language**. Disponível em <http://www.stroustrup.com/C++.html>. Acesso em 26 de fevereiro, 2015.

TANENBAUM, Andrew S. WOODHUUL, Albert S. **Sistemas Operacionais: Projeto e Implementação**, 2ed. Tradução de Edson Furmankiewicz. Porto Alegre. Editora Bookman, 2000.

TANENBAUM, Andrew S. **Modern Operating Systems.** 3ed. New Jersey, Editora Pearson, 2009.

THELIN, Johan. **Foundations of Qt Development**, 1ed. Berkeley. Editora APRESS, 2007.

APÊNDICE

Código fonte do escalonador circular desenvolvido no Capítulo 6.

```
1   #
2   # Escalonador RoundRobin (circular)
3   #
4
5
6   # Dados utilizados pelo escalonador em cada processo
7   class round_robin_data(scheduler_data):
8       def __init__(self):
9           scheduler_data.__init__(self)
10          self.quantum = int()
11          self.expireAt = int()
12
13
14  class round_robin(scheduler):
15      def __init__(self):
16          scheduler.__init__(self)
17          # Cria as filas de pronto e espera
18          self.wait = scheduler.queue()
19          self.ready = scheduler.queue()
20          # Flag para forçar o reescalonamento
21          self.need_resched = False
22          # Exibe as filas na interface gráfica
23          self.register_queue(self.ready)
24          self.register_queue(self.wait)
25
26      # Adiciona um novo processo
27      def add(self, process):
28          data = round_robin_data()
29          process.set_data(data)
30          self.ready.push_back(process)
31
```

```
32      # Suspende um processo
33      def suspend(self, process):
34          if process in self.wait:
35              return
36
37          process.set_state(state.Suspended)
38          self.wait.push_back(process)
39          # Se estiver rodando é necessário reescalonar
40          if is_running(process):
41              self.need_resched = True
42              self.schedule()
43          self.ready.remove(process)
44
45
46      # Resume um processo que estava em estado de espera
47      def resume(self, process):
48          if process in self.ready:
49              return
50          process.set_state(state.Ready)
51          self.wait.remove(process)
52          self.ready.push_back(process)
53
54      # Remove um processo
55      def remove(self, process):
56          self.wait.remove(process)
57          self.ready.remove(process)
58          process.set_state(state.Zombie)
59          # Se estiver rodando é necessário reescalonar
60          if is_running(process):
61              self.need_resched = True
62              self.schedule()
63
64      # Reescalona os processos
65      def schedule(self):
66          current = cpu.running_process()
67          if current == None:
68              log.error('Processo atual é inválido')
69              return
```

```
70
71          resched = self.need_resched
72
73          if not is_idle(current):
74              data = current.data()
75              if data.expireAt >= get_time() or not is_running(current):
76                  resched = True
77          else:
78              if not self.ready.empty():
79                  resched = True
80
81          self.need_resched = False
82
83          if resched:
84              next = self.ready.pop_front()
85
86              # Apenas mudamos para o estado de pronto se estivermos
    rodando
87              if is_running(current):
88                  current.set_state(state.Ready)
89                  if not is_idle(current):
90                      self.ready.push_back(current)
91
92              if next != None:
93                  # Se não for IDLE, então damos um novo quantum
94                  if not is_idle(next):
95                      ndata = next.data()
96                      ndata.quantum = 2
97                      ndata.expireAt = get_time() + ndata.quantum;
98
99                  # Muda o estado para rodando e faz a troca de contexto
100                 next.set_state(state.Running)
101                 cpu.switch_to(next)
102                 return
103
104             # Não existem processos para escalonar, troca para IDLE
105             cpu.switch_to(idle())
106
107 # Cria um escalonador e o coloca em uso
```

```
my_sched = round_robin()
set_scheduler(my_sched)
```