MEC-SETEC

INSTITUTO FEDERAL MINAS GERAIS - *Campus* Formiga
Curso de Ciência da Computação

ESTUDO COMPUTACIONAL DE MÉTODOS PARA ESCALONAMENTO DE PROCESSOS COM ABORDAGEM CLÁSSICA E *FUZZY* VIA SIMULAÇÃO

Danilo da Silva Alves

Orientador: Prof. Me. Diego Mello da Silva

Formiga - MG
2018

Danilo da Silva Alves

ESTUDO COMPUTACIONAL DE MÉTODOS PARA ESCALONAMENTO DE PROCESSOS COM ABORDAGEM CLÁSSICA E *FUZZY* VIA SIMULAÇÃO

Trabalho de Conclusão de Curso apresentado ao Instituto Federal Minas Gerais - Campus Formiga, como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Me. Diego Mello da Silva

Formiga - MG
2018

Alves, Danilo da Silva.
004 Estudo Computacional de Métodos para Escalonamento de Processos com Abordagem Clássica e *Fuzzy* via Simulação / Danilo da Silva Alves. -- Formiga : IFMG, 2018.
156p. : il.

Orientador: Prof. Me. Diego Mello da Silva
Trabalho de Conclusão de Curso – Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – *Campus* Formiga.

1. Escalonamento de Processos. 2. Gerenciamento de CPU. 3. Sistemas Operacionais. 4. Simulação. 5. *Fuzzy*. I. Título.

CDD 004

Ficha catalográfica elaborada pela Bibliotecária Me. Naliana Dias Leandro CRB6-1347

DANILO DA SILVA ALVES

ESTUDO COMPUTACIONAL DE MÉTODOS PARA ESCALONAMENTO DE PROCESSOS COM ABORDAGEM CLÁSSICA E FUZZY VIA SIMULAÇÃO

Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Minas Gerais-Campus Formiga, como Requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Aprovado em: 24 de Novembro de 2018.

BANCA EXAMINADORA

Prof.º Diego Mello da Silva

Prof.º Everthon Valadão dos Santos

Prof.º Walace de Almeida Rodrigues

Dedico este trabalho a minha mãe, Elza,
por todos os ensinamentos que fizeram de mim quem eu sou.

Agradecimentos

Agradeço acima de tudo a minha mãe, Elza, que em momento algum poupou esforços para me prover apoio durante esta jornada, por sempre me apoiar em minhas decisões e por me ensinar que devemos sempre buscar ser pessoas melhores e nunca desistir das nossas ambições.

Agradeço ao Prof. Diego, que, mais do que um professor, se tornou um grande amigo e um modelo a ser seguido, compartilhando suas experiências, oferecendo conselhos e aplicando a face cruel da realidade quando necessário. Que me apoiou desde o início, aceitando orientar este projeto e o desempenhando com maestria.

Deixo aqui também os meu agradecimentos ao Prof. Everthon, por quem tenho um grande apreço, que me inspira como modelo de professor a ser seguido e que foi de grande importância na realização deste projeto, cedendo seus conhecimentos na área e direcionando este trabalho para o caminho correto.

Agradeço aos amigos que conquistei durante este percurso, que se tornaram minha família ao longos destes anos. Com eles pude compartilhar minhas alegrias e minhas tristezas. Com eles aprendi que, por mais que as as coisas estejam ruins, na companhia de amigos a carga é dividida e a angústia é aliviada.

Por fim, agradeço a todas as pessoas que não compatilhavam da mesma experiência mas entenderam os motivos da minha ausência em suas vidas.

"O corpo não elimina os venenos ao conhecer seus nomes.
Tentar controlar o medo, a depressão ou o tédio chamando-os por nomes
é recorrer a superstição de confiança em maldições e invocações.
É tão fácil ver por que isso não funciona.
Obviamente, nós tentamos conhecer, nomear e definir o medo
para torná-lo objetivo, isto é, separá-lo do eu."
Alan Watts

Resumo

A proposta deste trabalho é efetuar um estudo acerca de métodos de escalonamento de processos em sistemas operacionais que utilizem em sua construção abordagens clássicas e de inteligência computacional, esta última baseada em lógica *fuzzy*.O estudo realizou um comparativo entre o desempenho de métodos de escalonamento clássicos FCFS, SRT, SJF, RR e Prioridades presentes na literatura especializada com os métodos *fuzzy* PFCS de Ajmani (2013), FPCS de Bashir et al. (2011) e IFCS de Behera et al. (2012) considerando quatro indicadores de desempenho, a saber: "Tempo de Espera", "*Throughput*", "*Turnaround*" e "Utilização da CPU". Para realizar a comparação, os métodos de escalonamento propostos foram aplicados via simulação de eventos discretos em seis diferentes cenários com temporização estocástica. Na simulação considerou-se um sistema computacional com diferentes variações na disponibilidade de CPUs e dispositivos de I/O, usadas para descrever cenários com escassez ou alta disponibilidade destes recursos. Cada método foi experimentado nestes diferentes cenários por meio de 100 replicações independentes do modelo de simulação, comparando-se a mediana dos referidos resultados para cada um dos indicadores de desempenho. Por fim, analisou-se quais dos métodos tiveram melhor desempenho por cenário, concluindo ao final que os métodos SRT, SJF e PFCS/IFCS foram robustos diante da variação de recursos nos diferentes cenários, sendo que os dois primeiros métodos podem causar postergação indefinida enquanto que os métodos inspirados em lógica *fuzzy* tendem a atender processos de qualquer tamanho e natureza.

Palavras-chave: Escalonamento de Processos, Gerenciamento de CPU, Sistemas Operacionais, Simulação, *Fuzzy*.

Abstract

The goal of this work is to study different methods of process scheduling in Computer System based both in classic and computational intelligence approaches. In this work we make a performance comparison of the methods FCFS, SRT, SJF, RR and Priority, described by Tanembaum (2010) and Silberzchats (2013), with three new methods based on fuzzy logic: PFCS (Ajmani, 2013), FPCS (Bashier et al, 2011) and IFCS (Behera et al, 2012). Four perfomance indicators were used to make this comparison: Waiting time, Throughput, Turnaroud and CPU Utilization. A discrete event simulator were used to reproduce the life cycle of processes in a computer system, with CPU and I/O management. Six different scenarios were used to test the methods in the simulator, varying the availability of CPUs and I/O devices. Each method ran over 100 independent replications of the model, considering stochastic time in arriving of new processes (exponential) and burst time of I/O-CPU operations (triangular). After that, we analyze all the outcomes comparing the median of each performance indicators among all the six scenarios. Our results suggest that, considering the variations of computer resources, the classic methods SRT and SJF, and the fuzzy-based methods PFCS/IFCS had good performance, but the PFCS/IFCS priority methods can avoid starvation while SRT and SJF cannot lead with this because the policy they are based.

Keywords: Process Scheduling, CPU Management, Operating Systems, Simulation, Fuzzy.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

CPU	*Central Process Unit*
FCFS	*First Come, First Served*
FEL	*Future Event List*
FPCS	*Fuzzy Priority CPU Scheduling*
I/O	*Input/Output*
IFCS	*Improved Fuzzy-Based CPU Scheduling*
PFCS	*Proposed Fuzzy CPU Scheduling*
PRTY	*Priority*
RR	*Round-Robin*
SJF	*Shortest Job First*
SRT	*Shortest Remaining Time First*

Lista de símbolos

μ_p	Associação para variável Prioridade
μ_b	Associação para variável Tempo de *Burst*
μ_h	Assosciação para variável Taxa de Resposta

Sumário

1 INTRODUÇÃO

O escalonador de processos é a ferramenta base dos sistemas operacionais. Ele é responsável por definir qual dos processos será executado na CPU. Alternando a CPU entre os processos, o sistema operacional pode elevar a produtividade do computador (SILBERSCHATZ, 2013). Tanenbaum (2010) afirma que, como o tempo de CPU é um recurso escasso, um bom escalonador pode fazer uma grande diferença no desempenho observado e na satisfação do usuário. Diante do exposto, é possível compreender a importância de efetuar uma boa distribuição do uso de CPU no tempo entre os processos concorrentes. Para que essa situação seja alcançada, é necessário que se tenha um bom algoritmo de escalonamento, que forneça cada fatia de tempo de forma justa com o ojetivo de se oter um melhor desempenho do sistema.

Neste contexto Iyoda (2000) ressalta que o interesse na busca por sistemas que explorem os paradigmas da inteligência computacional tem crescido de forma expressiva nos últimos tempos. O principal objetivo dessas pesquisas é desenvolver sistemas computacionais que sejam eficientes, robustos, fáceis de operar e capazes de fornecer soluções de qualidade para problemas complexos. Este documento relata o estudo e desenvolvimento de um protótipo de simulador de escalonador de processos e o seu uso na simulação e análise dos principais métodos de escalonamento clássicos listados na literatura, e de métodos de escalonamento baseados no paradigma de inteligência computacional, mais especificamente, lógica *fuzzy*. Será efetuada uma análise do comportamento destes métodos quando aplicados a diferentes cenários com quantidade variada de recursos a fim de observar em qual ambiente cada método de escalonamento pode melhor ser aplicado.

A realização deste trabalho tem por intenção (i) contribuir no âmbito didático, uma vez que deixará como legado uma ferramenta de código aberto para estudo e compreensão sobre o funcionamento do escalonador de processos de um sistema computacional hipotético e das políticas de escalonamento mais comuns; (ii) exercitar o aspecto investigativo sobre políticas de escalonamento em virtude da existência de análise experimental dentro de um escopo reduzido de experimentos realizados sob cenários com diferentes recursos; e (iii) o aspecto de aplicação dado que pretende apresentar e analisar métodos que possuem algum nível de inteligência computacional embutida.

1.1 Trabalhos Relacionados

Devido a complexidade em se abstrair os conceitos computacionais do funcionamento dos sistemas operacionais, existem estudos que buscam como objetivo tornar o entendimento

desses eventos uma etapa menos obscura, provendo inclusive recursos didáticos para este fim. Nesta seção serão apresentados simuladores que buscam facilitar o aprendizado em disciplinas de sistemas operacionais.

1.1.1 PSSAV

O *Process Scheduling Simulation, Analyzer, and Visualization*[1] (PSSAV) é um protótipo de simulador de escalonamento de processos determinístico desenvolvido em 2010 por Firmansyah Adiputra e Hari Toha Hidayat. Fornece suporte aos algoritmos de escalonamento *First-Come, First-Served* (FCFS), *Round-Robin* (RR), Prioridades e *Shortest Job First* (SJF) preemptivo e não preemptivo (ADIPUTRA; HIDAYAT, 2010). O simulador possui uma interface gráfica por onde é possível informar um conjunto de processos como entrada, definidos pelo identificador, tempo de chegada, tempo de *burst* e prioridade do processo. Para o algoritmo de escalonamento RR é possível definir um *quantum*. O diferencial que o simulador apresenta está na exibição dos dados de saída, em forma de gráficos e animações. Também é possível armazenar os resultados para efetuar futuros comparativos entre os métodos. Desenvolvido em linguagem Java, o PSSAV é uma ferramenta de código aberto multiplataforma.

1.1.2 SOsim

O SOsim[2] foi desenvolvido pelo professor Luiz Paulo Maia como parte de sua dissertação de mestrado. O simulador foi desenvolvido com o objetivo de criar uma ferramenta gratuita que facilitasse e melhorasse o ensino das aulas de sistemas operacionais para alunos e professores (MAIA, 2001). De acordo com Maia (2001), O SOsim permite que o professor apresente os conceitos e princípios de funcionamento de um sistema operacional multiprogramável e/ou multitarefa, como Unix, OpenVMS e Windows, de uma maneira simples. O simulador permite visualizar os conceitos de multiprogramação, processo e suas mudanças de estado, gerência do processador (escalonamento) e a gerência memória virtual. A partir das opções de configuração, é possível selecionar diferentes políticas e alterar o funcionamento do simulador. Desta forma, o aluno tem a oportunidade de visualizar os conceitos teóricos apresentados em aula de forma simples e animada. O simulador implementa as políticas *round robin* (RR) e escalonamento por Prioridades. Desenvolvido em Borland Delphi 7.0, executa em ambiente Windows e, até a data deste documento, a ferramenta não possui o código fonte disponibilizado.

[1] Disponível em: https://code.google.com/archive/p/pssav/
[2] Disponível em: http://www.training.com.br/sosim/

1.1.3 TBC-SO/WEB

O TBC-SO/WEB[3] é um software voltado para ensino/aprendizagem via web. O projeto tem como objetivo a contribuição no ensino presencial ou no ensino a distância dos tópicos (i) políticas de escalonamento de processos e (ii) políticas de alocação de memória principal, presentes em uma disciplina introdutória de sistemas operacionais. Essa contribuição é fomentada pela sua atuação como facilitador do processo de aprendizagem desses tópicos uma vez que conceitos abstratos são apresentados de forma intuitiva (REIS et al., 2009). O TBC-SO/WEB foi desenvolvido em Java, e pode ser executado em plataforma web e possui código fonte disponível à comunidade. Segundo Reis et al. (2009), as principais características deste simulador são:

- Interface gráfica.
- Conteúdo teórico embutido à ferramenta.
- Telas e legendas auto explicativas.
- Animação do passo a passo de cada tópico.
- Implementação das políticas de alocação de memória *First-Fit*, *Next-Fit*, *Best-Fit* e *Worst-Fit*.
- Implementação das políticas de escalonamento de processos *First-Come, First Served* (FCFS), *Shortest Job First*(SJF), *Shortest Remaining Time* (SRT), escalonamento por prioridades e *Highest Response Rate Next*.

1.1.4 Simuladores de Gerência de Memória e Processador para Auxílio às Aulas Teóricas de Sistemas Operacionais

Ribeiro et al. (2014) explicam que, com o objetivo de diminuir o impacto negativo causado pela difícil caracterização do real dinamismo dos eventos computacionais estudados nas disciplinas de sistemas operacionais, foram desenvolvidos em seu trabalho dois simuladores, sendo eles: (i) simulador das rotinas de gerenciamento de memória real e virtual e (ii) simulador de gerenciamento de CPU em sistemas operacionais com arquitetura *multicore*.

Os simuladores desenvolvidos pelos autores possuem interface visual e interatividade com o usuário, implementando as estratégias de *Page-Fault* e alocação *First-Fit*, *Best-Fit* e *Worst-Fit* para o simulador de gerenciamento de memória e, políticas de escalonamento de bando e de tempo compartilhado assim como o gerenciamento de *threads* no simulador de gerenciamento de CPU. Os simuladores são multiplataforma e foram desenvolvidos em

[3] Disponível em: http://professores.dcc.ufla.br/ heitor/Projetos/TBC_SO_WEB/tbc_so_web.html

linguagem Java. Até a data deste documento o código fonte não foi disponibilizado pelos autores.

1.1.5 Uma Ferramenta de Apoio à Aprendizagem de Sistemas Operacionais

Isotani et al. (2009) apresenta o desenvolvimento de uma ferramenta para auxiliar alunos que cursam a disciplina de sistemas operacionais a compreender melhor o comportamento de processos no sistema. A ferramenta desenvolvida permite a captura de processos de usuários do sistema em tempo real, a fim de aplicar o mesmo processo ao simulador de escalonador com a política desejada. Após a captura do processo ou simulação, é exibido um gráfico visual como saída permitindo um comparativo entre o comportamento do processo no sistema operacional e o comportamento do processo no simulador ao se aplicar a política desejada.

O software foi desenvolvido em linguagem C e opera em ambiente Windows. Implementa os métodos *Round-Robin* (RR), *First-Come, First-Served* (FCFS) e *Shortest-Job-First* (SJF). Como parâmetros da simulação é possível alterar o tempo de *clock*, o tempo de troca de contexto e o *quantum* (somente para RR). Até a data deste documento não foi disponibilizada nenhuma forma de acesso ao código do software.

1.2 Objetivos

1.2.1 Objetivo Geral

Objetivo geral deste projeto é efetuar um comparativo entre os algoritmos de Bashir et al. (2011), Behera et al. (2012) e Ajmani (2013) com algoritmos clássicos da literatura que utilizem métodos de escalonamento preemptivo, não preemptivo e de tempo compartilhado via simulação de eventos discretos considerando as métricas tempo de espera, *throughput*, *turnaround* e utilização da CPU.

1.2.2 Objetivos Específicos

- Elaboração de um simulador capaz de reproduzir o escalonamento de processos utilizando suas características gerais, e ciclo de vida clássico;

- Implementação dos métodos de escalonamento de processos tradicionais, assim como os métodos baseado em inteligência computacional propostos pelos autores citados;

- Simular a chegada de processos em um sistema computacional, assim como o gerenciamento de processos ao longo de seu ciclo de vida;

- Efetuar a análise dos resultados obtidos e efetuar um comparativo através de gráficos que apresentam o comportamento de cada método quando considerando seu desempenho segundo as métricas tempo de espera, *throughput*, *turnaround* e utilização da CPU.

1.3 Estrutura do Documento

Este documento é dividido como segue. No **Capítulo 2** serão apresentados os conceitos sobre sistemas operacionais, processos, escalonamento de processos e lógica *fuzzy*, necessários para que o leitor tenha compreensão dos tópicos substanciais que estruturam a realização deste trabalho; o **Capítulo 3** apresenta a metodologia de desenvolvimento empregada para este trabalho, indicando as decisões tomadas e os recursos utilizados para sua concretização; no **Capítulo 4** são apresentados os resultados em forma de comparativo, obtidos através da experimentação realizada em diferentes cenários com variações de recursos; no **Capítulo 5** acontece o fechamento do trabalho, onde é feito o resumo do que foi realizado, os resultados encontrados e as propostas de continuidade do projeto.

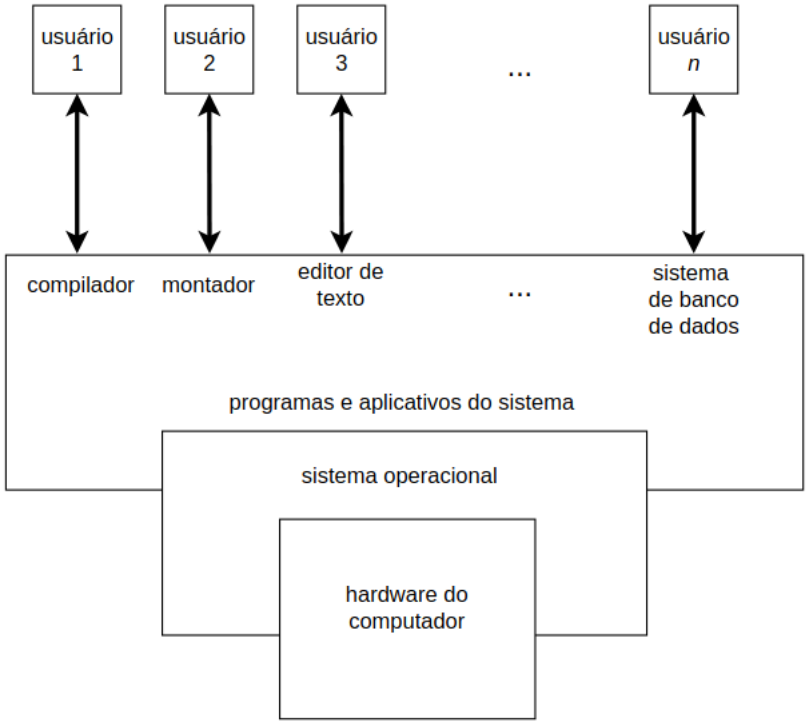
2 FUNDAMENTAÇÃO TEÓRICA

Neste ponto, serão apresentados os conceitos básicos da composição de um sistema computacional e de como acontece o gerenciamento de processos em uma Unidade Central de Processamento (CPU). Também serão descritos e exemplificados os métodos de escalonamento clássicos mais encontrados na literatura, assim como os métodos baseados em lógica *fuzzy* propostos pelos referidos autores.

2.1 Sistemas Operacionais

Silberschatz (2013) afirma que um sistema de computação pode ser dividido em quatro componentes: o hardware, o sistema operacional, os programas aplicativos e os usuários, como mostra a Figura 1. Essa visão em camadas permite tratar didaticamente os componentes típicos de um sistema computacional ao mesmo tempo que apresenta as diferentes responsabilidades de cada componente para o melhor uso dos recursos computacionais envolvidos.

Figura 1 – Visão abstrata dos componentes de um sistema computacional.



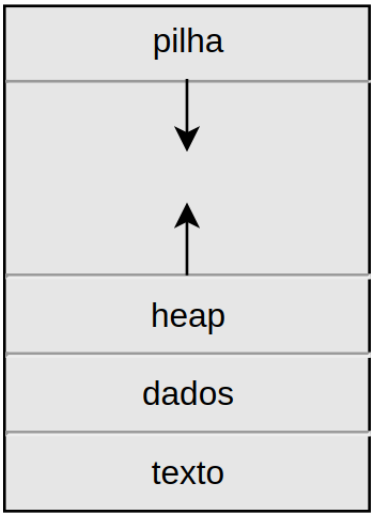
Fonte: (SILBERSCHATZ, 2013)

Cada um dos elementos apresentados será um pouco mais esclarecido. O **hardware** (i.e., a unidade central de processamento (CPU — *Central Process Unit*), a memória e os dispositivos de entrada/saída) fornece os recursos básicos de computação do sistema, porém em baixo nível. Os **programas aplicativos** (como processadores de texto, planilhas, compiladores e navegadores da Web) definem as formas pelas quais esses recursos são utilizados para resolver os problemas computacionais dos usuários. Estes elementos estão localizados na camada mais alta do sistema. Entre o hardware e os programas aplicativos encontra-se o **sistema operacional**. O sistema operacional é o programa mais intimamente envolvido com o hardware. Nesse contexto, podemos considerar um sistema operacional com um alocador de recursos. Ao lidar com solicitações de recursos numerosas e possivelmente concorrentes, o sistema operacional precisa decidir como alocá-los a programas e usuários específicos para poder operar o sistema de computação de maneira eficiente e justa (SILBERSCHATZ, 2013). Dentre os recursos mais disputados de um sistema computacional está o uso da Unidade Central de Processamento (CPU). As próximas seções apresentam detalhes sobre como este recurso é gerenciado, e as políticas clássicas e inspiradas em técnicas de inteligência computacional.

2.2 Caracterização de Processos

Silberschatz (2013) afirma que um processo é mais do que o código do programa, que também é conhecido como **seção de texto**. Ele também inclui a instrução corrente, representada pelo valor do contador de programa (PC) e pelo conteúdo dos registradores do processador. Geralmente, um processo também inclui a **pilha do processo**, que contém dados temporários (como parâmetros de funções, endereços de retorno e variáveis locais), e uma **seção de dados**, que contém as variáveis globais. Um processo também pode incluir um ***heap***, que é uma região de memória alocada dinamicamente durante o tempo de execução do processo. A estrutura de um processo hipotético caracterizado em memória é mostrada na Figura 2.

Figura 2 – Processos em memória.



Fonte: (SILBERSCHATZ, 2013)

Tanenbaum (2010) explica que, informalmente um processo é apenas um programa em execução, acompanhado dos valores atuais do contador de programa, dos registradores e das variáveis. Conceitualmente cada processo tem sua própria CPU virtual. É claro que, na realidade, a CPU troca, a todo momento, de um processo para outro. Esse mecanismo de trocas rápidas é chamado de **multiprogramação**.

2.2.1 Estados de Um Processo

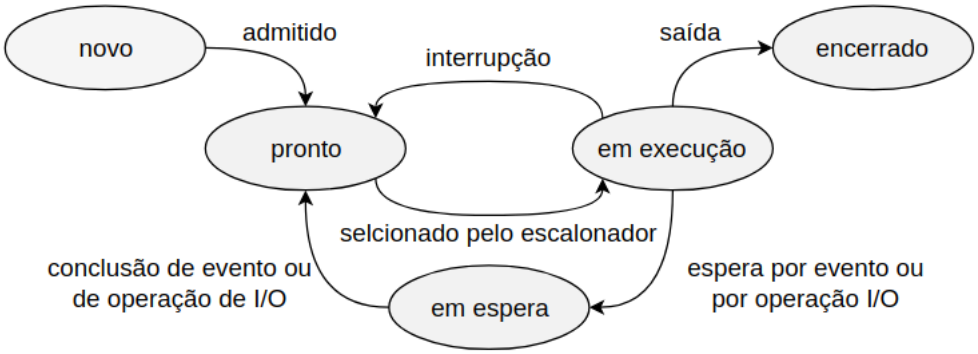
Segundo Silberschatz (2013), quando um processo é executado, este muda de estado. O estado de um processo é definido em parte pela atividade corrente desse processo. Cada processo por estar em um dos estados a seguir:

- **Novo**: O processo está sendo criado.
- **Em execução**: Suas instruções estão sendo executadas.
- **Em espera**: O processo está aguardando a ocorrência de algum evento (como a conclusão de uma operação I/O).
- **Pronto**: O processo está aguardando ser atribuído a uma CPU.
- **Concluído**: O processo terminou sua execução.

Silberschatz (2013) salienta que esses nomes são arbitrários e variam entre os sistemas operacionais. No entanto, os estados que eles representam são encontrados em

todos os sistemas. É importante reforçar que só um processo pode estar em execução em alguma CPU a cada instante. Porém, muitos processos podem estar prontos e em espera. O diagrama de estado que corresponde ao ciclo de vida de um processo pode ser observado na Figura 3.

Figura 3 – Diagrama de estados do processo.



Fonte: (SILBERSCHATZ, 2013)

O ciclo de vida de um processo ocorre segundo a explicação que segue. Após a criação do processo (**novo**), este é admitido pelo sistema e aguarda (**pronto**) até que ele seja selecionado para executar em uma CPU. Após ser selecionado para iniciar execução a CPU é reservada para o processo, que executa (**em execução**) até que seja interrompido para dar lugar a outro processo julgado pelo sistema como mais urgente ou para aguardar a execução de um evento de I/O (**em espera**). Caso seja interrompido sem terminar de fato sua execução, o processo regressa ao estado pronto para aguardar novamente por sua vez. Após o término da ocorrência de operação I/O, o processo regressa ao estado pronto novamente. Após terminar de executar na CPU, o processo deixa o sistema (**encerrado**), dando lugar para outro processo que esteja pronto executar na CPU (TANENBAUM; 2010).

2.3 Gerenciamento de CPU

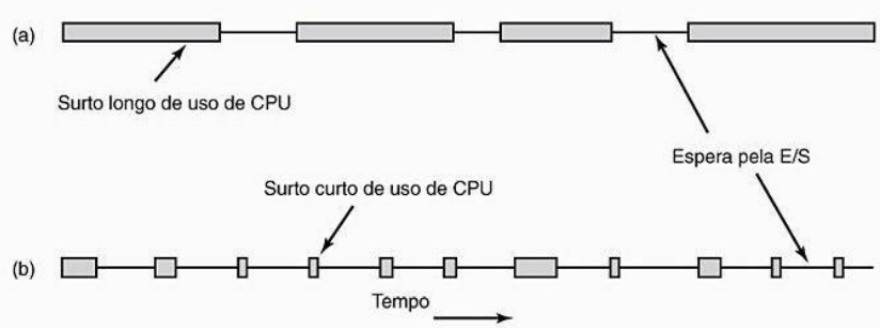
Em um sistema com um único processador, só um processo pode ser executado de cada vez. Qualquer outro processo deve esperar até que a CPU esteja livre e possa ser realocada. Silberschatz (2013) explica que a idéia é relativamente simples. Um processo é executado até que este tenha que esperar, normalmente após uma solicitação entrada/saída (I/O, do inglês *Input/Output*). Em um sistema de computação simples, sem uso de multiprogramação, a CPU permanece ociosa durante uma operação de I/O de forma que todo esse tempo de espera é desperdiçado com nenhum trabalho útil realizado. Em sistemas

computacionais que utilizam a multiprogramação, o objetivo é fazer um uso produtivo desse tempo ocupando a CPU com outro processo. Vários processos são mantidos na memória ao mesmo tempo. Quando um processo entra em espera, o sistema operacional desvincula a CPU deste processo e a atribui a outro processo que está aguardando execução. Esse padrão continua. Sempre que um processo tem de esperar, outro processo pode assumir o uso da CPU, aumentando a eficiência do sistema computacional.

Silberschatz (2013) salienta que o objetivo da multiprogramação é otimizar o uso da CPU aumentando sua utilização sempre que existir processos aptos para execução. A percepção do compartilhamento de tempo é feito mediante a alternância da CPU entre os processos com frequência alta suficiente para que os usuários possam interagir com cada programa enquanto ele está sendo executado. Para alcançar esses objetivos, o escalonador de processos seleciona um processo disponível 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 de esperar até que a CPU esteja disponível. Os processos que estão armazenados na memória principal e estão prontos e esperando para serem executados são mantidos em uma **Fila de Prontos**.

De acordo com Tanenbaum (2010), quase todos os processos alternam entre *bursts* (surtos) de computação e requisições de I/O, sendo o *busrt* de I/O mais frequente a escrita em disco. Este fato é ilustrado a seguir, na Figura 4. Em geral, a CPU executa indefinidamente até que seja feita uma chamada de sistema para ler um arquivo ou escrever nele. Quando a chamada de sistema encerra, a CPU computa novamente até que haja nova requisição para escrita de dados, e assim o ciclo se repete, alternando entre CPU *bursts* e I/O *bursts*. Neste sentido, um I/O *burst* ocorre quando um processo entra no estado bloqueado esperando que um dispositivo externo termine a operação de I/O que está realizando. Quando um processo gasta mais tempo em operações na CPU do que em esperas por I/O, este é chamado de "orientado à CPU" ou ainda "CPU *bound*". Em contrapartida, processos que gastam mais tempo em operações de I/O do que em processamento na CPU são chamados de "orientados à I/O" ou "I/O *bound*".

Figura 4 – Usos de CPU se alternam com períodos de espera por I/O. (a) Um processo orientado à CPU. (b) Um processo orientado à I/O.



Fonte: (TANENBAUM, 2010)

Tanenbaum (2010) também explica que, quando um sistema computacional é multiprogramado, pode ocorrer de múltiplos processos competirem pelo uso da CPU ao mesmo tempo. Essa situação ocorre sempre que dois ou mais processos estão simultaneamente na Fila de Prontos. Se somente uma CPU se encontrar disponível, deverá ser feita uma seleção de qual processo será executado em seguida. A parte do sistema operacional que realiza essa seleção é chamada de **escalonador**, e o algoritmo utilizado por ele é o **algoritmo de escalonamento**.

Silberschatz (2013) apresenta que as decisões de escalonamento podem ser tomadas nas quatro situações a seguir:

1. Quando um processo passa do estado de execução para o estado de espera (por exemplo, como resultado de uma solicitação de I/O).
2. Quando um processo passa do estado de execução para o estado de pronto (por exemplo, quando ocorre uma interrupção).
3. Quando um processo passa do estado de espera para o estado de pronto (por exemplo, conclusão da operação de I/O).
4. Quando o processo termina.

Nas situações 1 e 4, não há alternativa. Um novo processo deve ser selecionado para execução caso a Fila de Prontos não esteja vazia. No entanto, nas situações 2 e 3 há alternativa. Quando o escalonamento só ocorre nas situações 1 e 4, dizemos que o esquema de escalonamento é **não-preemptivo**. Caso o escalonamento ocorra também nas situações 2 e 3, o esquema de escalonamento é **preemptivo**. Diferentes algoritmos de escalonamento possuem propriedades distintas, e a escolha do algoritmo pode favorecer

uma classe de processos em vez de outra. Na escolha do algoritmo a ser usado em uma situação específica, devemos considerar as propriedades dos diversos algoritmos.

Muitos critérios têm sido propostos para a comparação de algoritmos de escalonamento. As características usadas na comparação podem fazer uma grande diferença no algoritmo avaliado como melhor. Os critérios são os descritos a seguir:

- **Utilização da CPU.** Nosso objetivo é manter a CPU tão ocupada quanto for possível. Conceitualmente, a utilização da CPU varia entre 0 e 100 por cento. Em um sistema real, ela deve se manter entre 40 por cento (para um sistema pouco carregado) e 90 por cento (para um sistema muito carregado). O cálculo deste indicador é apresentado pela Equação 1.

$$\frac{\sum_{i=1}^{n} u_i}{T * n}. \tag{1}$$

  Onde T representa o tempo de simulação, n a quantidade de núcleos envolvidos na simulação e u a utilização total de cada núcleo i.

- ***Throughput***. Quando a CPU se encontra ocupada com a execução de processos, trabalho está sendo realizado. Uma medida de trabalho é a quantidade de processos que são concluídos por unidade de tempo, chamada de *throughput*. Se tratando de processos longos, essa taxa pode ser de um processo por hora; já para transações curtas, ela pode ser de dez processos por segundo. O cálculo deste indicador pode ser observado na fórmula 2.

$$\sum_{i=1}^{n} 1 \tag{2}$$

  Onde n representa o número total de processos que finalizaram execução.

- **Tempo de *Turnaround*.** Ao analisarmos um processo específico, o critério importante é quanto tempo dura sua execução. O tempo decorrido entre o momento em que o processo é submetido e o momento de sua conclusão é o tempo de *turnaround*. O tempo de *turnaround* consiste na soma dos períodos gastos com a espera para ser alocado à memória, a espera na Fila de Prontos, a execução na CPU e a execução de operações de I/O. O cálculo do tempo médio de *turnaround* pode ser expresso Equação 3.

$$\frac{\sum_{i=1}^{n} (Tf_i - Tc_i)}{n} \tag{3}$$

  Onde n indica a quantidade de processos que chegaram ao fim de sua execução e Tf_i e Tc_i os tempos de finalização e chegada do i-ésimo processo, respectivamente.

- **Tempo de Espera.** O algoritmo de escalonamento não afeta o período de tempo durante o qual um processo é executado ou realiza operações de I/O; o algoritmo apenas afeta o tempo que um processo gasta esperando na Fila de Prontos. O tempo de espera é a soma dos períodos gastos em espera na Fila de Prontos. O cálculo do tempo médio de espera para um conjunto de processos é apresentado na Equação 4.

$$\frac{\sum_{i=1}^{n} Te_i}{n} \tag{4}$$

Onde n representa a quantidade de processos que chegaram ao sistema e Te_i o tempo de espera do i-ésimo processo.

É desejável a maximização da utilização da CPU e do *throughput* e a minimização do tempo de *turnaround* e do tempo de espera. Para cada política de escalonamento usada, o desempenho de cada um dos indicadores pode variar segundo a estratégia empregada, que pode valorizar algum indicador em detrimento dos demais. Cabe conhecer qual política melhor responde para cada critério considerado.

2.4 Inferência com Lógica *Fuzzy*

Butt e Akram (2016) explicam que os problemas do mundo real em sua maioria são complexos e as informações obtidas através deles nem sempre são precisas e suficientes. Nos cenários de tomada de decisão, muitas vezes temos disponíveis vários caminhos a serem tomados e muitas vezes todas as informações destes caminhos não são conhecidas. Estes casos não podem ser modelados com a utilização de teoria de conjunto simples e lógica booleana. Para lidar com tais situações, normalmente são utilizadas a teoria de conjuntos *fuzzy* (ZADEH et al., 1965) e a lógica *fuzzy* (ZADEH, 1975).

2.4.1 Conjuntos *Fuzzy*

Transcheit (2004) explica que na teoria clássica de conjuntos, o conceito de pertinência de um elemento a um determinado conjunto é bem definido. Dado o conjunto A em um universo U, os elementos deste universo simplesmente pertencem ou não pertencem àquele conjunto. Isso pode ser expresso pela função de pertinência 5.

$$f_A(x) = \begin{cases} 1, & \text{se } x \in \text{A} \\ 0, & \text{se } x \notin \text{A} \end{cases} \tag{5}$$

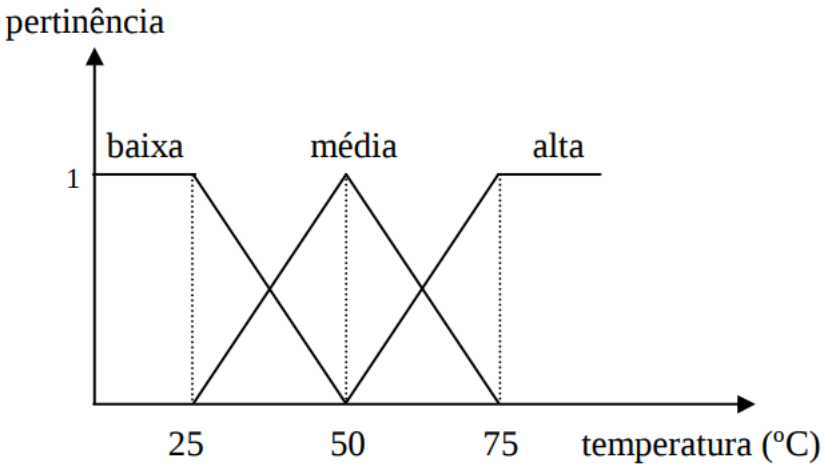
Como alternativa à esta forma de tratar incerteza Zadeh (1973) propôs uma caracterização mais ampla, generalizando a função de pertinência de modo que ela pudesse assumir infinitos valores no intervalo [0,1]. Um conjunto *fuzzy* A em um universo X é

definido por uma função de pertinência $\mu_A(x) : X \rightarrow [0, 1]$, representado por um conjunto de pares ordenados $A = \{\mu_A(x)/x\}, x \in X$ onde $\mu_A(x)$ indica o quanto x é compatível com o conjunto A. Isso faz com que um determinado elemento possa pertencer a mais de um conjunto *fuzzy* ao mesmo tempo, porém com diferentes graus de pertinência.

2.4.2 Variáveis Liguísticas

Transcheit (2004) define uma variável linguística como sendo uma variável cujos valores são nomes de conjuntos *fuzzy*. Por exemplo, a temperatura de um determinado processo pode ser uma variável linguística que pode assumir os valores baixa, média e alta. Estes valores são descritos através de conjuntos *fuzzy*, representados por funções de pertinência, como observado na Figura 5.

Figura 5 – Funções de pertinência para a variável temperatura.



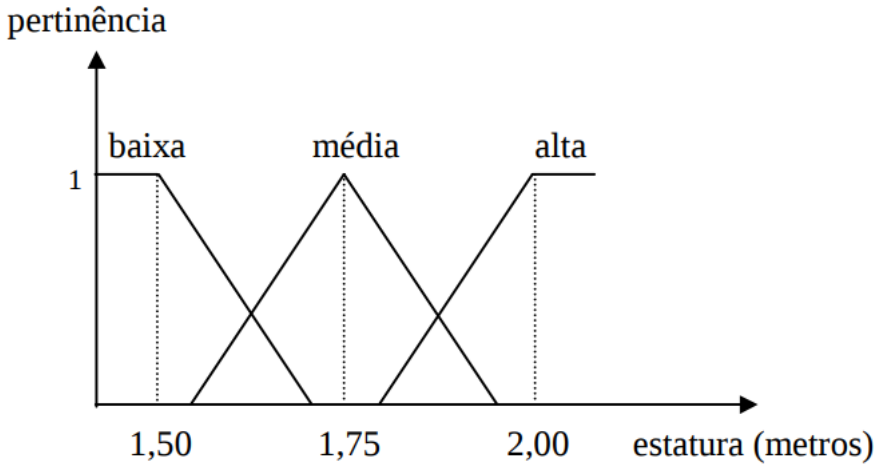
Fonte: (TRANSCHEIT, 2004)

Os valores de uma variável linguística podem ser sentenças lógicas construídas a partir de termos primários (alto, médio, baixo, longo, curto), conectivos lógicos (negação, conjunção e disjunção), modificadores linguísticos (muito, pouco, extremamente) e delimitadores (parênteses). A principal função das variáveis linguísticas é fornecer sistemática para a caracterização aproximada de fenômenos complexos ou mal definidos. Em essência, a utilização do tipo de descrição linguística empregada por seres humanos ao invés de variáveis quantificadas permite o tratamento de sistemas que seriam muito complexos quando analisados através de termos matemáticos convencionais (TRANSCHEIT, 2004).

2.4.3 Funções de Pertinência

Transcheit (2004) discorre em seu trabalho sobre o tema funções de pertinência. Segundo o autor, elas podem ser apresentadas em diferentes formas, sujeitando-se ao conceito que se deseja representar e do contexto em que serão utilizadas. Para exemplificar, seja a variável linguística estatura (de pessoas), representada pelos termos: T(estatura) = {baixa, média, alta}, definidos por suas funções de pertinência. Estaturas de até 1,50 metros apresentam grau de pertinência 1 no conjunto baixa, de forma que e o grau de pertinência a este conjunto decresce à medida que a estatura aumenta, por exemplo, considerando que a pessoa em questão deixa de ser baixa quando atinge estatura de 1,75 m. Estaturas de 1,80 metros apresentam grau de pertinência diferente de zero tanto quando analisados sob o olhar de uma estatura média quanto de uma estatura alta, segundo o mapeamento de pertinência, e pela mesma lógica terá pertinência definida em nível zero se confrontarmos o valor de 1,80 m em relação ao que se considera uma estatura baixa. A Figura 6, adaptada de Transcheit (2004) mostra um possível mapeamento para o caso, utilizando funções de pertinência trapezóide e triangular.

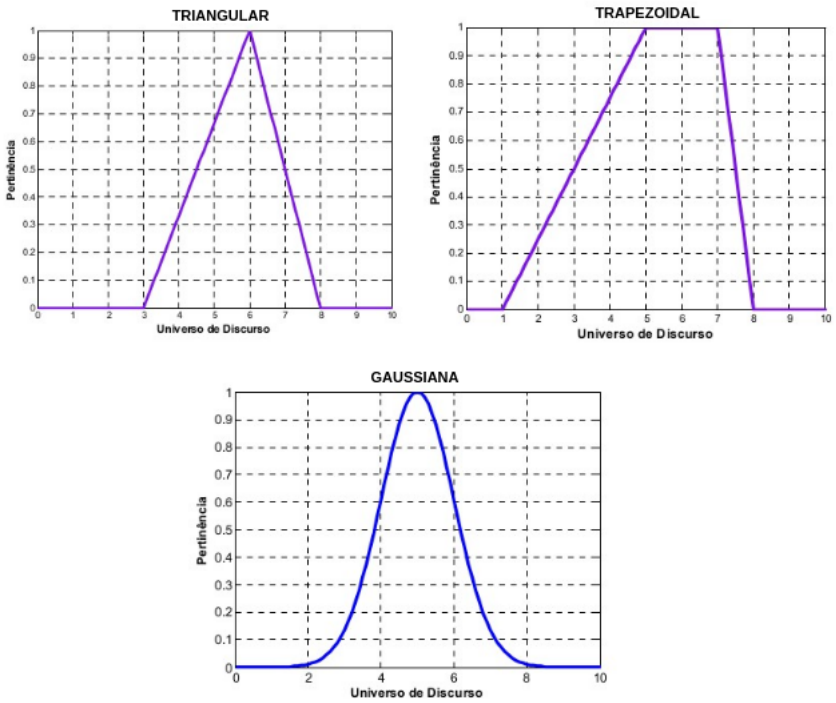
Figura 6 – Funções de pertinência para a variável estatura.



Fonte: (TRANSCHEIT, 2004)

Funções de pertinência podem ser definidas a partir das experiências e perspectivas do usuário, mas é comum fazer o uso de funções de pertinência padrão, como, por exemplo, as de forma triangular, trapezoidal e Gaussian, apresentadas na Figura 7.

Figura 7 – Principais formas de funções de pertinência.



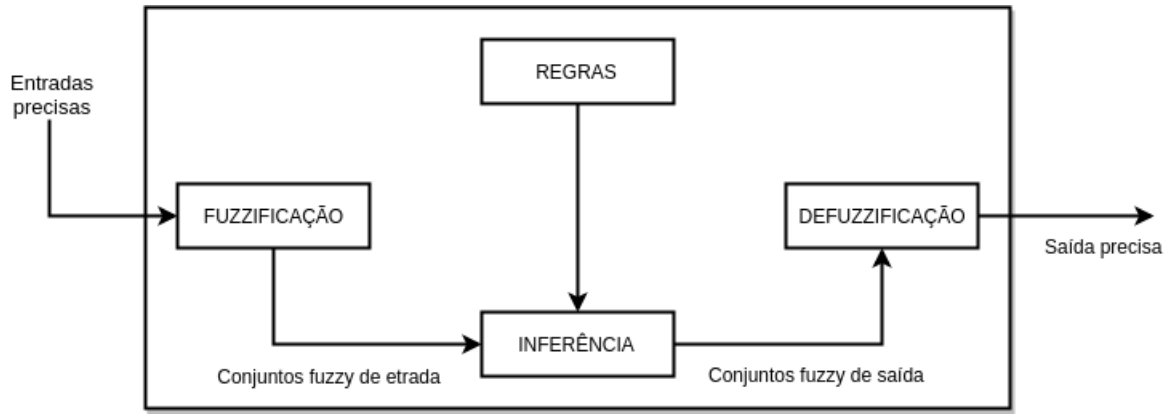
Fonte: Adaptado de (NETO et al., 2006)

2.4.4 Sistemas de Inferência *Fuzzy*

Bashir et al. (2011) nos mostra que um sistema de inferências *fuzzy* tenta derivar respostas de uma base de conhecimento usando um motor de inferência *fuzzy*. O motor de inferências, que é considerado o cérebro dos sistemas especialistas, fornece as metodologias utilizadas para que se possa efetuar o raciocínio sobre as informações da base de dados para que então sejam calculados os resultados. O motor de inferências é o conjunto de regras lógicas na forma **SE-ENTÃO** armazenadas na base de conhecimento do sistema, onde o **SE** representa a parte dos **Antecedentes** (entrada) e o **ENTÃO** a parte dos **Consequentes** (saída).

Seja o diagrama apresentado na Figura 8, obtido de Transcheit (2004). Explicando de maneira alto nível, em um sistema de inferências *fuzzy* são recebidas entradas não-*fuzzy*, ou "**entradas precisas**" (às vezes denominadas de nítidas ou *crisp*), resultantes de medições ou observações. Após isso, é necessário realizar o mapeamento destes dados para os conjuntos *fuzzy*, em uma etapa denominada de **fuzzificação** ou nebulização. O estágio de inferência determina como as regras de inferência são ativadas e combinadas para produzir o conjunto de saídas. No estágio de **defuzzificação** é efetuada a interpretação do conjunto obtido no estágio anterior, produzindo então a saída não-*fuzzy*, ou "**saída precisa**" em um formato específico (numérico, por exemplo).

Figura 8 – Diagrama de funcionamento de um sistema *fuzzy*.

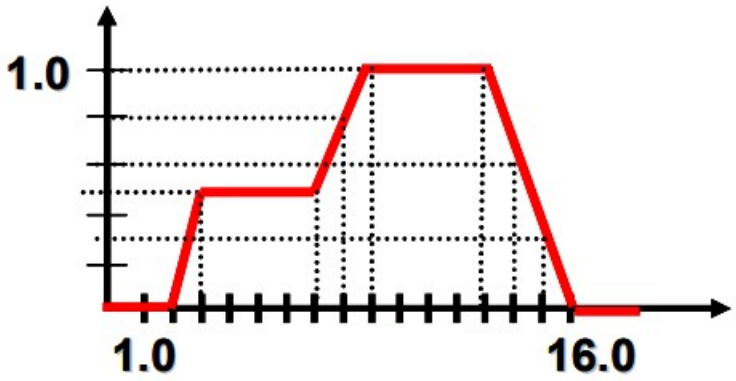


Fonte: Adaptado de (TRANSCHEIT, 2004)

Neto et al. (2006) explica que o método COG (center of gravity), também chamado de centro de gravidade ou ainda centróide, é o método de defuzzificação mais utilizado atualmente. Vamos supor um universo de discurso discreto, onde a saída nítida $\boldsymbol{S}$ é dada pelo centro de gravidade do conjunto de consequentes obtido através da composição das regras. Neste caso, n indica a amplitude (quantização) da saída, d_i representa o valor de saída para o intervalo i e $\mu_A\ (d_i)$ o grau de pertinência. Assim é possível encontrar o ponto de equilíbrio da região *fuzzy* calculando a média ponderada das regiões através da *equação* 6.

$$\boldsymbol{S} \leftarrow \frac{\sum_{i=0}^{n} d_i \mu_A(d_i)}{\sum_{i=0}^{n} \mu_A(d_i)} \tag{6}$$

Figura 9 – Variável de saída.



Fonte: Adaptado de (NETO at al., 2006)

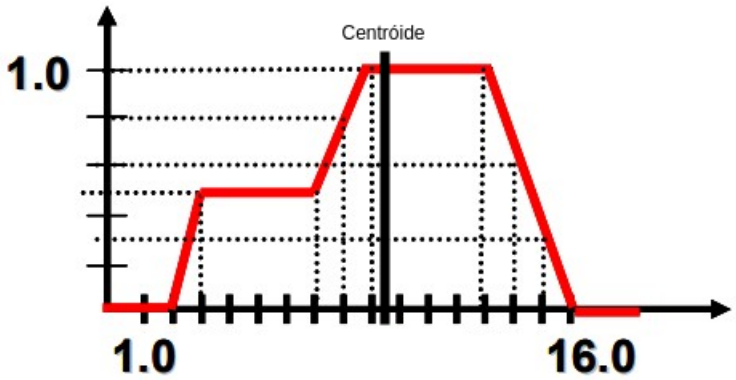
Para exemplificar, seja a seguinte variável de saída apresentada na Figura 9. Deseja-se encontrar o valor preciso pelo método de defuzzificação do centróide. Logo, o resultado da composição das saídas deverá ser discretizado, tomando-se medidas do produto dos valores da composição final pela pertinência da agregação. Isso resultará em:

$$\boldsymbol{S} = (0*0+1*0+2*0+3*0.5+4*0.5+5*0.5+6*0.5+7*0.5+8*0.8+9*1+10*1+11*1+12*1+13*1+14*0.6+15*0.3+16*0)/(0+0+0+0.5+0.5+0.5+0.5+0.5+0.8+1.0+1.0+1.0+1.0+1.0+0.6+0.3+0.0)$$

$$\boldsymbol{S} = 86.8/9.2$$

Temos que o centróide $\boldsymbol{S}$ é 9.4348, conforme exibe a Figura 10.

Figura 10 – Localização do centróide.



Fonte: Adaptado de (NETO at al., 2006)

Coppin (2012) afirma que, na verdade, o centro de gravidade deveria ser calculado como uma integral contínua; porém se usarmos uma soma discreta com uma seleção suficiente de valores, será possível obter uma resposta suficientemente aproximada.

Caso o leitor tenha interesse em se aprofundar nestes tópicos, é recomendada a leitura das obras Bittencourt (1995), Russel (2002), Russel (2004), Artero (2009) e Coppin (2012) presentes na literatura especializada.

2.5 Políticas de Escalonamento Clássicas

O escalonador da CPU lida com a decisão de para qual dos processos da Fila de Prontos a CPU deve ser alocada e que, para isso, existem muitos algoritmos de escalonamento diferentes (SILBERSCHATZ, 2013). Isto posto, a presente seção irá apresentar algumas políticas clássicas de escalonamento. Os detalhes de cada política são dados nas subseções que seguem.

2.5.1 Primeiro a Entrar, Primeiro a Ser Atendido

Segundo Silberschatz (2013), sem dúvida o algoritmo de escalonamento mais simples é o **"primeiro a entrar, primeiro a ser atendido" (FCFS, *first-come, first served*)**. Nesse algoritmo, o processo que solicita a CPU primeiro é o primeiro a usá-la. A implementação da política FCFS é facilmente controlado com uma fila FIFO (*first in, first out*). Quando um processo passa para o estado pronto, este é inserido no final da Fila de Prontos. Quando a CPU está livre, ela é alocada para o processo na cabeça da fila. O processo que está agora em execução é então removido da fila.

Considere o conjunto de processos a seguir, com tempo de pico fornecido em milissegundos:

Processo	Chegada	Tempo de Pico
P_1	0	24
P_2	1	3
P_3	2	3

Se os processos forem atendidos na ordem FCFS, será obtida a execução exibida no gráfico de Gantt a seguir, que ilustra o início e o fim da execução de cada processo participante:

Observe que o algoritmo de escalonamento FCFS não utiliza preempção. Uma vez que a CPU é alocada para um processo, este a ocupa até liberá-la. Silerschatz (2013) ressalta que lado negativo deste método é que o tempo médio de espera na política de escalonamento FCFS geralmente é bem longo.

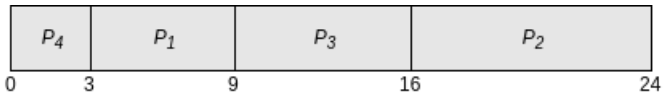
2.5.2 Trabalho Mais Curto Primeiro

Silberschatz (2013) apresenta uma abordagem diferente para escalonamento da CPU com o algoritmo **"trabalho mais curto primeiro" (SJF, *shortest job first*)**, onde é associado a cada processo seu intervalo do próximo pico de CPU. Quando a CPU está disponível, ela é designada ao processo que tem o próximo pico de CPU mais curto.

Como exemplo, considere o conjunto de processos a seguir, com tempo de pico fornecido em milissegundos:

Processo	Chegada	Tempo de Pico
P_1	0	6
P_2	0	8
P_3	0	7
P_4	0	3

Usando o escalonamento SJF, o conjunto de processos seria alocado conforme exibe o seguinte gráfico de Gantt:



Silberschatz (2013) declara que o algoritmo de escalonamento SJF é comprovadamente ótimo quando se trata de fornecer o menor tempo de espera médio para um conjuntos de processos. Executar um processo curto antes de um longo reduz mais o tempo de espera do processo curto do que aumenta o tempo de espera do processo longo. Consequentemente, o tempo de espera e o *turnaround* são reduzidos. Embora seja ótimo, o algoritmo de escalonamento SJF pode causar inanição quando ocorre um grande fluxo de processos curtos, fazendo com que processos longos esperem por tempo indefinido.

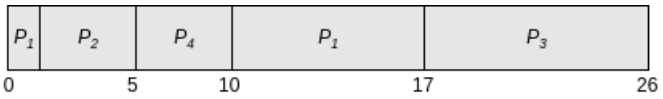
2.5.3 Próximo de Menor Tempo Restante

Tanenbaum (2010) apresenta que, com o algoritmo **"próximo de menor tempo restante" (SRT, *shortest remaining time*)**, o escalonador sempre escolhe o processo cujo tempo de execução restante seja menor. Para isso o tempo de execução deve ser previamente conhecido. Quando um novo processo chega à Fila de Prontos, é feita uma comparação entre o seu tempo total e o tempo restante do processo que está em execução. Se, para terminar, o novo processo precisar de menos tempo que o processo em execução, então esse será suspenso e o novo processo será iniciado. Esse algoritmo permite que processos curtos tenham um melhor desempenho.

Como exemplo, considere os processos a seguir, onde o tempo de pico é dado em milissegundos:

Processo	Chegada	Tempo de Pico
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

O escalonamento dos processos utilizando o método SRT é representado pelo gráfico de Gantt a seguir:



O processo P_1 é iniciado no momento 0, já que nesse instante ele é o único processo aguardando na Fila de Prontos. O processo P_2 chega no momento 1. O tempo restante do processo P_1 (7 milissegundos) é maior do que o tempo solicitado pelo processo P_2 (4 milissegundos); portanto o processo P_1 é interrompido e o processo P_2 é escalonado. Silberschatz (2013) ressalta que, por ser uma variante do SJF, o algoritmo de escalonamento SRT é ótimo quando se observa o tempo de espera. Diminui o tempo de *turnaround* e também pode ocasionar inanição de processos longos quando se tem uma grande demanda de processos curtos.

2.5.4 Chaveamento Circular

De acordo com Araújo (2011) como os sistemas operacionais podem oferecer serviços a múltiplos usuários ou até mesmo diversos serviços a um único usuário, é necessário que existam políticas de escalonamento para determinar as regras de como, quando e qual processo será executado na sua vez com a finalidade de satisfazer alguns objetivos conflitantes, conforme (TANENBAUM, 2001):

- Garantir que cada processo receba uma parte justa da CPU.
- Minimizar o tempo de resposta para os usuários interativos.
- Minimizar o tempo que os usuários devem esperar pela saída.
- Maximizar o número de tarefas processadas por hora.
- Conciliar processos de alta prioridade com baixa prioridade.

Para tentar satisfazer esses objetivos, geralmente os sistemas operacionais baseiam-se em tempo compartilhado, onde o tempo de resposta é dividido em intervalos de tempo (*quantum*) para o processo de acordo com seu algoritmo de escalonamento. Se durante a execução de um processo o *quantum* é esgotado, um novo processo é selecionado para execução, provocando então uma troca de contexto.

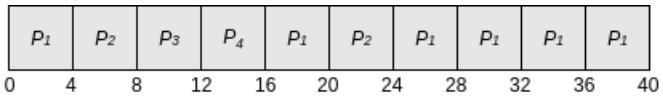
Tanenbaum (2010) afirma que o algoritmo de escalonamento por **chaveamento circular (RR, *round-robin*)** é um dos algoritmos mais simples, justos e utilizados. A cada processo é atribuído um intervalo de tempo, o seu ***quantum***, no qual ele é permitido

executar. Se, ao final do *quantum*, o processo ainda estiver executando, a CPU sofrerá preempção e será atribuída a outro processo. O mesmo ocorre se o processo for bloqueado ou terminar antes que o *quantum* tenha decorrido. O escalonador só precisa manter uma fila de processos que estejam no estado de pronto. Quando um processo utiliza todo o seu *quantum*, ele é colocado no final da fila.

Para exemplificar, vamos considerar o seguinte conjunto de processos, com tempo de duração de pico dado em milissegundos:

Processo	Chegada	Tempo de Pico
P_1	0	24
P_2	1	8
P_3	2	4
P_4	3	4

Os agendamentos resultantes pelo escalonamento RR, com *quantum* de 4 milissegundos, podem ser observados no gráfico de Gantt a seguir:



Tanenbaum (2010) complementa que o RR é um algoritmo de escalonamento justo, permitindo que cada processo execute um pouco na CPU. Porém, em um sistema com um fluxo constante de processos, o tempo de espera cresce. Após um processo ser executado, ele só receberá uma fatia de tempo da CPU após todos os outros processos receberem a mesma fatia, e, com um alto fluxo de chegada de processos, a Fila de Prontos irá crescer e consequentemente o tempo de espera irá aumentar.

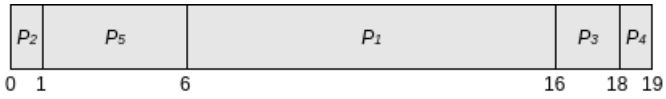
2.5.5 Escalonamento por Prioridades

Segundo Silberschatz (2013), no algoritmo de **escalonamento por prioridades (*Priority*)** uma prioridade é associada a cada processo, e a CPU é alocado ao processo em espera que possuir a prioridade mais alta. Processos com prioridades iguais são agendados de acordo com a sua chegada (FCFS). É importante observar que discutimos o escalonamento em termos de alta e baixa prioridade. Geralmente as prioridades são indicadas por algum intervalo de números fixo, como 0 a 7 ou 0 a 4.095. No entanto não existe um consenso geral que defina de 0 é a prioridade mais alta ou mais baixa. Alguns sistemas utilizam números baixos para representar baixa prioridade; outros usam números baixos para representar altas prioridades.

Para ilustrar o método de escalonamento, considere os cinco processos a seguir, com duração do pico de CPU dado em milissegundos:

Processo	Chegada	Tempo de Pico	Prioridade
P_1	0	10	3
P_2	0	1	5
P_3	0	2	2
P_4	0	1	1
P_5	0	5	4

Usando o método de escalonamento por prioridades, considerando mais prioritário o processo com prioridade representada pelo maior número, os processos seriam agendados de acordo com o gráfico de Gantt a seguir:



Silberschatz (2013) complementa que um grande problema dos algoritmos de escalonamento por prioridades é o bloqueio indefinido, ou inanição, que é quando o algoritmo de escalonamento mantém processos de baixa prioridade em espera por tempo indefinido, aumentando o tempo de espera médio do sistema. Em um sistema de computação muito carregado, um fluxo constante de processos de prioridade mais alta pode impedir que um processo de prioridade baixa consiga usar a CPU. A vantagem deste método está em diferenciar os processos de acordo com a sua importância.

2.6 Políticas de Escalonamento com Abordagem *Fuzzy*

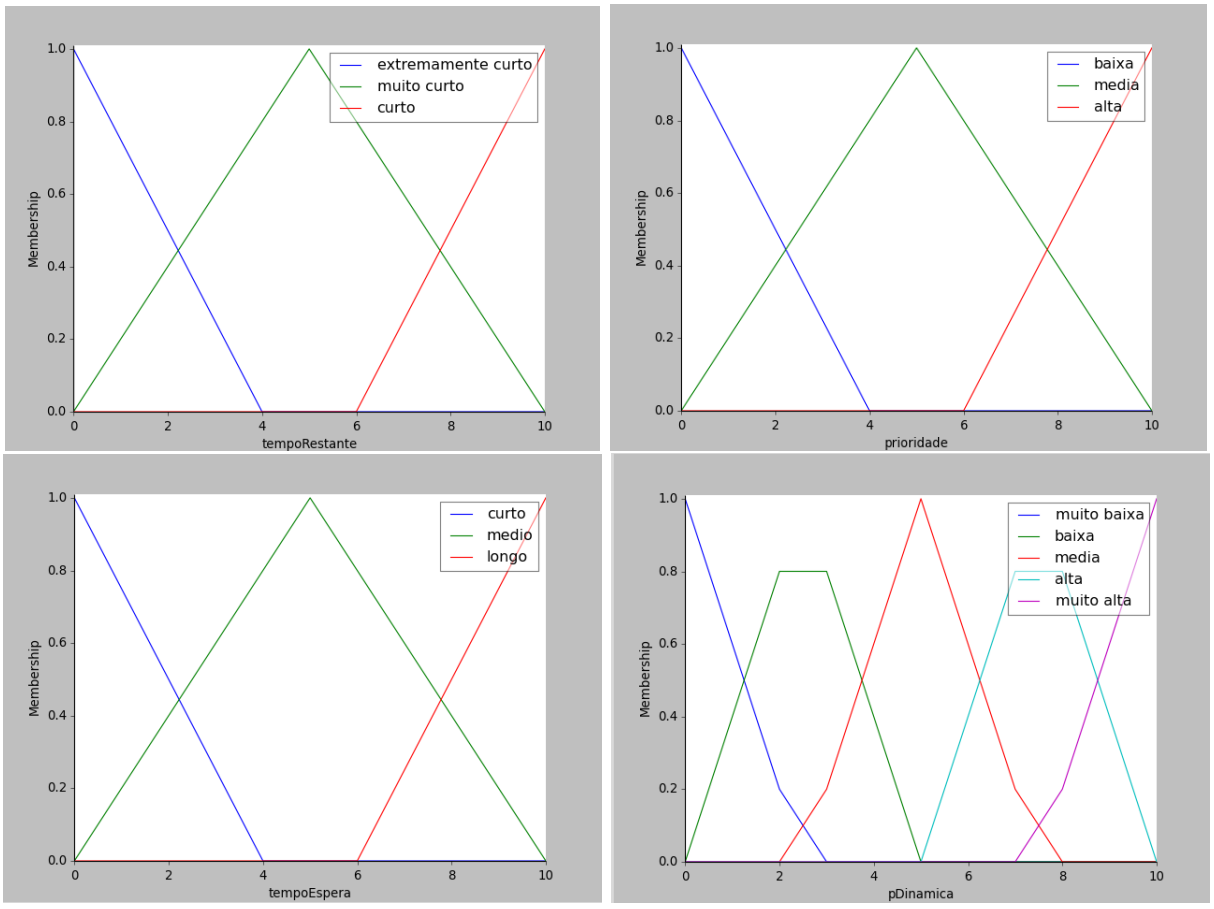
No contexto específico de escalonamento de processos, técnicas de inteligência computacional, como a lógica *fuzzy*, têm sido utilizadas em trabalhos reportados na literatura. Durante nossa investigação foram identificados alguns trabalhos, dos quais destacam-se os métodos que serão apresentados nas próximas subseções.

2.6.1 FPCS (*Fuzzy Priority CPU Scheduling Algorithm*)

Bashir et al. (2011) propôs um algoritmo que utilizasse um sistema de inferências *fuzzy* para escalonar processos na CPU. No modelo proposto, os dados de entrada para o motor de inferências consistem em três variáveis, sendo elas (i) a prioridade estática (designada antes da execução), que pode ser classificada em “baixa”, ”média” ou ”alta”,

(ii) o tempo restante de execução do processo, que é classificado em “extremamente curto”, “muito curto” ou “curto” e (iii) o tempo pelo qual o processo já aguardou por execução, podendo ser classificado em “curto”, “médio” ou “longo”. Após analisar as variáveis, o sistema atribui ao processo uma prioridade dinâmica, que indica sua ordem de escalonamento e é classificada em “muito baixa”, “baixa”, “média”, “alta” e “muito alta”. As funções de pertinência para cada variável de entrada podem ser observadas nas Figuras 11 a **??**.

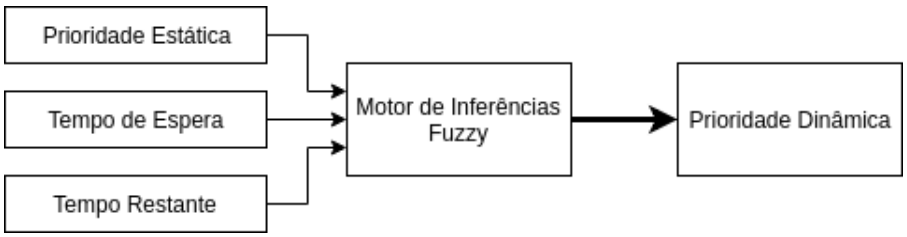
Figura 11 – Funções de pertinência para o tempo de execução restante, prioridade estática, tempo de espera e prioridade dinâmica do processo.



Fonte: Adaptado de (BASHIR at al., 2011)

O motor de inferências consiste em 27 regras, conforme mostra a Tabela 1, onde, fornecidas as três métricas de entrada, se obtém a prioridade dinâmica como resultado. As regras envolvidas na inferência são dadas a seguir, no diagrama apresentado na Figura 12.

Figura 12 – Diagrama de funcionamento do sistema de inferências *fuzzy* para o algoritmo FPCS.



Fonte: Adaptado de (BASHIR at al., 2011)

Tabela 1 – Regras de inferência para o algoritmo FPCS

Regra	Prioridade estática	Tempo restante	Tempo de espera	Prioridade dinâmica
1	Baixa	Extremamente curto	Curto	Muito alta
2	Baixa	Extremamente curto	Médio	Muito alta
3	Baixa	Extremamente curto	Longo	Muito alta
4	Baixa	Muito curto	Curto	Muito baixa
5	Baixa	Muito curto	Médio	Baixa
6	Baixa	Muito curto	Longo	Alta
7	Baixa	Curto	Curto	Muito baixa
8	Baixa	Curto	Médio	Baixa
9	Baixa	Curto	Longo	Alta
10	Média	Extremamente curto	Curto	Muito alta
11	Média	Extremamente curto	Médio	Muito alta
12	Média	Extremamente curto	Longo	Muito alta
13	Média	Muito curto	Curto	Média
14	Média	Muito curto	Médio	Média
15	Média	Muito curto	Longo	Muito alta
16	Média	Curto	Curto	Média
17	Média	Curto	Médio	Média
18	Média	Curto	Longo	Alta
19	Alta	Extremamente curto	Curto	Muito alta
20	Alta	Extremamente curto	Médio	Muito alta
21	Alta	Extremamente curto	Longo	Muito alta
22	Alta	Muito curto	Curto	Alta
23	Alta	Muito curto	Médio	Alta
24	Alta	Muito curto	Longo	Muito alta
25	Alta	Curto	Curto	Alta
26	Alta	Curto	Médio	Alta
27	Alta	Curto	Longo	Muito alta

Fonte: (BASHIR at al., 2011)

Alguns exemplos de regras podem ser observados a seguir:

SE a prioridade estática é baixa E o tempo restante é extremamente curto E o tempo de espera é curto ENTÃO a prioridade dinâmica é muito alta.

SE a prioridade estática é baixa E o tempo restante é muito curto E o tempo de espera é curto ENTÃO a prioridade dinâmica é muito baixa.

A proposta do algoritmo é ter uma média de tempo de espera dos processos menor e *throughput* maior ao comparar seu desempenho com o algoritmo de escalonamento por prioridades.

2.6.2 IFCS (*Improved Fuzzy-Based CPU Scheduling*)

Behera et al (2012) propôs o **IFCS**, um algoritmo de escalonamento para sistemas operacionais de tempo real que utiliza em sua estrutura um motor de inferências *fuzzy*. Na arquitetura proposta na Figura 13, o tempo de *burst* (BT), o tempo de chegada (AT) e a prioridade (P) dos processos são utilizados como entrada para a unidade computacional, que é responsável por calcular a associação do tempo de *burst* (μ_b), prioridade (μ_p) e taxa de resposta (μ_h), que são os dados de entrada para a base de regras *fuzzy*, onde as novas prioridades dos processos são avaliadas individualmente para escalonamento. A taxa de resposta, necessária para calcular μ_h, é obtida através do cálculo 7. As associações são calculadas conforme as Equações 8, 9 e 10, e a prioridade dinâmica é calculada pela Equação 11. A proposta do algoritmo é reduzir o tempo de *turnaround* e o tempo de espera dos processos.

$$Tx_{resp} = \frac{T_{espera} + T_{execução}}{T_{execução}} \tag{7}$$

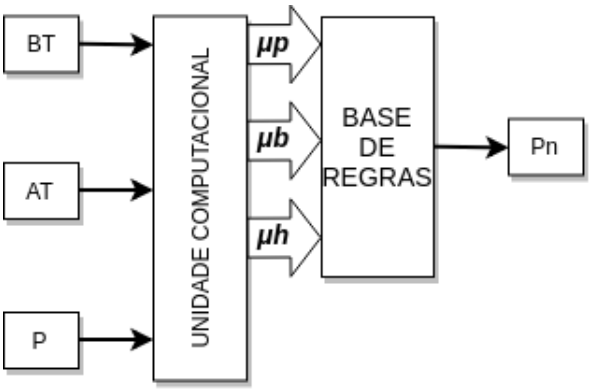
$$\mu_p = \frac{Prioridade}{max(Prioridade) + 1} \tag{8}$$

$$\mu_b = 1 - \frac{T_{burst}}{max(T_{burst}) + 1} \tag{9}$$

$$\mu_h = \frac{Tx_{resp}}{max(Tx_{resp}) + 1} \tag{10}$$

$$P_n = max\left\{\mu_p, \mu_b, \mu_h\right\} \tag{11}$$

Figura 13 – Arquitetura proposta para o algoritmo IFCS.



Fonte: Adaptado de (BEHERA et al., 2012)

2.6.3 PFCS (*Proposed Fuzzy CPU Scheduling Algorithm*)

Ajmani (2013) propôs um algoritmo utilizando abordagem de sistemas *fuzzy* para escalonar processos em um sistema operacional de tempo real. O algoritmo calcula a prioridade dinâmica do processo baseando-se apenas na sua prioridade estática e no tempo de *burst*. Para calcular a prioridade dinâmica de cada processo, são utilizadas as Equações a seguir:

$$\mu_p = \frac{pt_i}{max(pt_1)} + 1 \tag{12}$$

$$\mu_b = 1 - \left(\frac{bt_i}{max(bt_i)} + 1\right) \tag{13}$$

$$dp_i = (\mu_p + \mu_b) \tag{14}$$

$$dp_i = max\left\{\mu_p, \mu_b\right\} \tag{15}$$

Onde bt_i é o tempo do *burst* atual, pt_i representa a prioridade estática e dp_i a prioridade dinâmica que será atribuída ao processo. Após a obtenção dos valores μ_p e μ_b através das Equações 12 e 13 respectivamente, é então calculada a prioridade dinâmica do processo, que consiste em encontrar o processo na Fila de Prontos que contenha a menor prioridade estática. Este processo recebe como prioridade dinâmica o resultado da Equação 14. Aos demais processos, a prioridade dinâmica é definida pela Equação 15. Em termos de tempo de *turnaround* e tempo de espera o PFCS obtém os mesmos resultados que o IFCS, citado anteriormente. Porém, a proposta deste algoritmo é utilizar como vantagem o fato de eliminar a dependência do cálculo da taxa de resposta.

2.7 Simulação

Chwif e Medina (2015) afirmam que os sistemas reais comumente apresentam uma complexidade elevada, fato que se deve, principalmente, à sua natureza dinâmica (mudança de estado ao longo do tempo) e à sua natureza aleatória (que é regida por variáveis aleatórias). Um modelo de simulação tem a capacidade de capturar com mais fidelidade essas características, buscando repetir em um computador o mesmo comportamento que o sistema apresentaria quando exposto às mesmas condições. Um modelo de simulação pode ser capaz de analisar todos aspectos do sistema de interesse. A simulação é uma ferramenta de análise de cenários que pode prever, com certa confiança, o comportamento de um sistema baseado em dados.

Banks (1998) define a simulação como sendo uma imitação do funcionamento de um sistema do mundo real. Simulação é uma poderosa metodologia para solução de problemas do mundo real, inclusive problemas de escalonamento. Diante desta afirmação, Kiran (1998) agrega que modelos de simulação são totalmente capazes de representar os detalhes de situações de escalonamento e a abordagem de simulação para estes problemas é útil para transmitir os detalhes aos usuários devido a possibilidade de inclusão de conteúdo visual oferecida pela simulação.

2.7.1 Simulação de Eventos Discretos

Banks (2005) afirma que sistemas em que as mudanças são predominantemente descontínuas (não obedecem uma sequência temporal) são chamados de **sistemas discretos**. A simulação de eventos discretos pode ser aplicada em sistemas onde ocorrem mudanças de estados apenas em um pontos discretos no tempo. Estes modelos de simulação são analisados por métodos numéricos, que, como explica Banks (2005), são métodos computacionais que são construídos através das proposições do modelo. Após análise os resultados são coletados para estimar as medidas de desempenho do sistema real que foi simulado. O sistema é modelado de acordo com seu estado a cada momento no tempo e este estado é alterado conforme as entidades são modificadas através dos eventos. Os principais conceitos utilizados para definir a simulação de eventos discretos são descritos a seguir:

- **Sistema**: Coleção de entidades que operam em conjunto ao longo do tempo para alcançar um ou mais objetivos. Exemplo: Pessoas e Máquinas.

- **Modelo**: Representação abstrata do sistema. Normalmente contém os relacionamentos estruturais, lógicos ou matemáticos que definem o sistema em termos de estado, entidades e seus atributos, processos, atividades e atrasos que compõem o sistema.

- **Estado do sistema**: Conjunto de variáveis que armazenam todas as informações necessárias para descrever sistema em cada instante de tempo.
- **Entidade**: Qualquer componente ou objeto do sistema que requer uma representação distinta e detalhada no modelo.
- **Atributo**: Propriedades que integram as entidades.
- **Evento**: Ocorrência que altera o estado do sistema onde são conhecidos o tempo de início e o tempo de duração, que pode ser definido em termos de distribuição estatística.
- **FEL**: Lista de eventos futuros (do inglês, *Future Event List*), armazena em ordem cronológica a ocorrência de eventos no sistema.
- ***Clock***: Variável que representa o tempo simulado. É alterada conforme a ocorrência de eventos no sistema.
- **Coleção**: Conjunto de entidades de uma mesma classe que compõem o sistema.

Chwif e Medina (2015) explicam que a simulação de eventos discretos leva em consideração as mudanças de estado do sistema ao longo do tempo e é utilizada para modelar sistemas que mudam de estado em momentos específicos no tempo, a partir da ocorrência de eventos. O *clock* sempre indica o instante em que um evento acontece.

2.7.2 Diagrama de Ciclo de Atividades

Sistemas de eventos discretos acabam envolvendo sistemas de filas de uma forma ou de outra e podem ser representados por Diagramas de Ciclos de Atividades (DCA), um dispositivo adequado para explicitar as relações lógicas destes sistemas. Por meio do DCA são explicitadas as interações entre as principais entidades do modelo (PIDD, 2004).

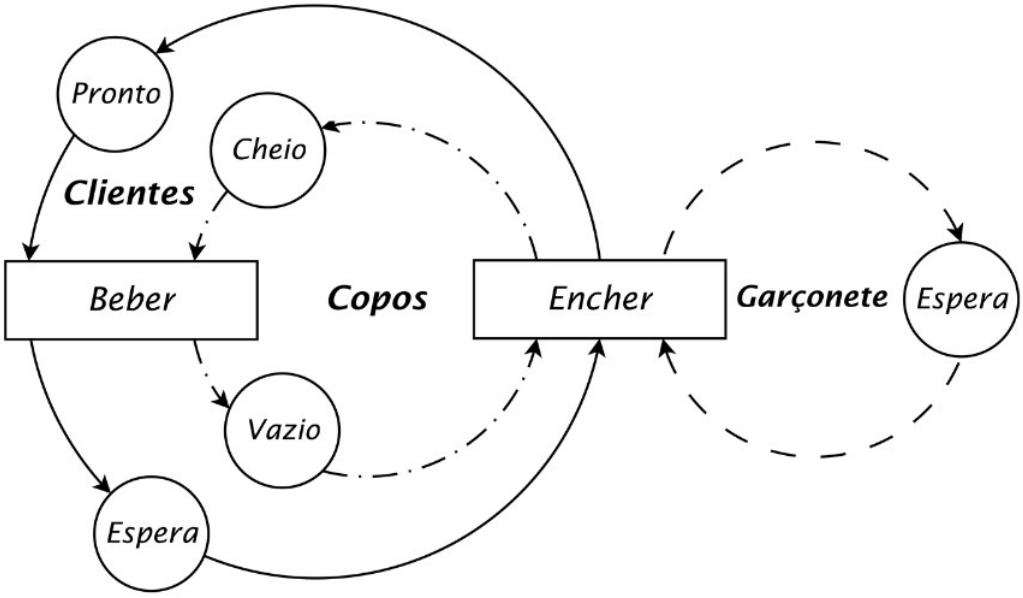
De acordo com Chwif e Medina (2015), o DCA é uma forma de modelar as interações entre os objetos pertencentes a um sistema. Os componentes que integram o DCA são descritos a seguir:

- **Entidade**: No DCA, uma entidade é qualquer componente do sistema que possa manter sua identidade ao longo do tempo. São elementos do sistema que possuem o comportamento constantemente rastreado.As entidades podem estar participando de alguma atividade ou aguardando em filas.
- **Atividade**: Normalmente, uma atividade é um evento que envolve a cooperação de diferentes entidades e através de uma atividade as entidades mudam seus estados no sistema.

- **Fila**: Um estado de fila, chamado de estado passivo, não envolve a participação de diferentes atividades, sendo, geralmente, um estado onde uma entidade deve aguardar até o acontecimento de um evento.

Para exemplificar, considere uma versão simplificada de um *pub*, onde existam três entidades: "o cliente", "a garçonete" e "o copo". Ao entrar no *pub*, o cliente pede uma cerveja, e então a garçonete enche um copo para servi-lo. A seguir, o cliente bebe a cerveja. O cliente participa, portanto, das atividades "encher e beber". Os copos e a garçonete participam da atividade "encher".

Figura 14 – DCA do *pub*.



Fonte: (CHWIF; MEDINA, 2015)

O DCA exibido na Figura 14 ilustra logicamente que a atividade "Beber" não pode ser iniciada até que um cliente esteja na fila "Pronto" e um copo esteja na fila "Cheio". Similarmente, a atividade "Encher" não pode dar início até que uma garçonete esteja na fila "Espera", um como esteja na fila "Vazio" e um cliente esteja na fila "Espera". Ao término de cada atividade, a transição das entidades é fixo. Depois de "Encher", a garçonete vai para a fila "Espera", e o copo vai para a fila "Cheio". Depois de "Beber", o copo vai para a fila "Vazio", e o cliente vai para a fila "Espera".

A convenção de que atividades e filas devem estar dispostas alternadamente num DCA faz com que a modelagem se torne mais robusta na circunstância de uma mudança inevitável no modelo. É essencial, ainda, que todo o ciclo de vida de uma entidade seja

fechado. Chwif e Medina (2015) concluem que, as principais vantagens de se representar um modelo através da técnica de DCA são:

- Simplicidade de construção, pois torna-se possível desenvolver um modelo conceitual de simulação a partir de, apenas, dois símbolos: atividade e fila;
- Habilidade de mostrar, explicitamente, as interações entre os objetos do sistema e seus fluxos;
- Facilidade de entendimento e utilização;

Por outro lado, o DCA apresenta algumas desvantagens:

- Os diagramas se tornam muito incompreensíveis à medida que se aumenta a complexidade do sistema a ser modelado;
- Se torna complicado capturar toda a lógica do modelo, especialmente ao se modelar um sistema com lógica complexa. Por exemplo, o DCA não mostra claramente disciplinas de fila ou atribuições condicionais de valores para atributos de entidades, Segundo Paul (1993), o DCA mostra o fluxo lógico, mas não a lógica em profundidade.

3 MATERIAIS E MÉTODOS

Serão apresentados nas seções subsequentes a configuração de hardware utilizada na construção do projeto assim como os recursos computacionais envolvidos nas etapas de formulação, implementação, execução e validação do modelo. Também será explicada a metodologia envolvida para que as fases do projeto fossem bem definidas, com o objetivo de garantir a organização necessária para sua condução do momento inicial até a finalização.

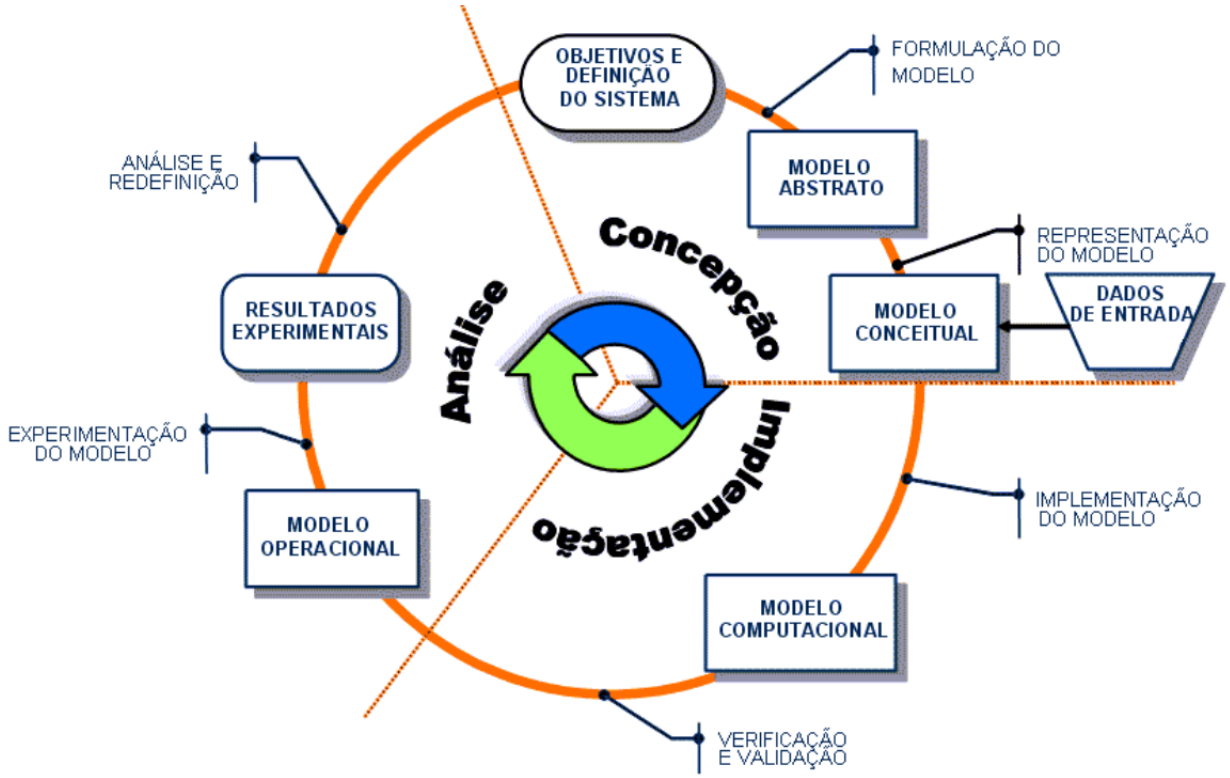
3.1 Materiais

O simulador e os algoritmos de escalonamento foram implementados em linguagem Python 2.7. A linguagem de *script* Bash foi utilizada para a replicação dos experimentos enquanto a linguagem R foi útil na geração de gráficos com objetivo de análise dos resultados. A implementação dos algoritmos de escalonamento e do simulador, tal como a análise de resultados, foram realizadas em um computador DELL XPS L502X, com processador Intel CoreTM i7-2670QM 2.20GHz, 6GB de memória RAM, placa de vídeo NVIDIA GeForce GT 525M 1GB e sistema operacional Linux Mint 17.1 64 bits.

3.2 Metodologia

A construção de um programa de computador que, para muitos, é a própria simulação, é apenas uma dentre as inúmeras atividades de um estudo de simulação (CHWIF; MEDINA, 2015). Neste trabalho foram seguidas as etapas propostas na metodologia descrita por Chwif e Medina (2015), que afirmam que, o desenvolvimento de um modelo de simulação compõe-se de três etapas, conforme apresenta a Figura 15: (i) concepção ou formulação do modelo; (ii) implementação do modelo; e (iii) análise dos resultados.

Figura 15 – Metodologia da simulação.



Fonte: (CHWIF; MEDINA, 2015)

Na primeira etapa, "**Concepção**", foi efetuada uma investigação com o objetivo de entender claramente o sistema de escalonamento de processos a ser simulado e os seus objetivos assim como as métricas de desempenho. Os dados de entrada também foram modelados nesta fase. Posteriormente foi decidido com clareza qual seria o escopo do modelo, suas abstrações e seu nível de detalhamento. Finalizada a etapa de concepção, o modelo que anteriormente estava apenas em mente, denominado modelo abstrato, foi representado com a utilização de um diagrama de ciclo de atividade a fim de torná-lo um modelo conceitual.

Na segunda etapa, "**Implementação**", o modelo conceitual foi transformado em um modelo computacional com a utilização de recursos computacionais como linguagens e suas bibliotecas. Essa foi a etapa que mais consumiu tempo no projeto. Após a implementação do modelo computacional, foi feita uma comparação minuciosa com o modelo conceitual, com a finalidade de avaliar se o seu funcionamento atende aos requisitos estabelecidos na etapa de concepção. Em seguida foi feita a validação do modelo, que será discutida na seção 3.2.6.

Na terceira etapa, "**Análise**", com o modelo computacional pronto para realização

dos experimentos, temos o modelo experimental ou modelo operacional. Nesta etapa, o modelo foi executado diversas vezes, com cenários de execução variados a fim de efetuar uma análise do comportamento do simulador. Robinson (2004) enfatiza que o estudo de simulação não é linear. A melhor maneira é imaginar que o projeto foi desenvolvido de forma cíclica, onde as etapas foram repetidas na sequência apresentada na Figura 15 até que, entre uma iteração e outra, não existissem mais diferenças nos resultados de cada etapa, tornando o modelo sólido.

3.2.1 Revisão Bibliográfica

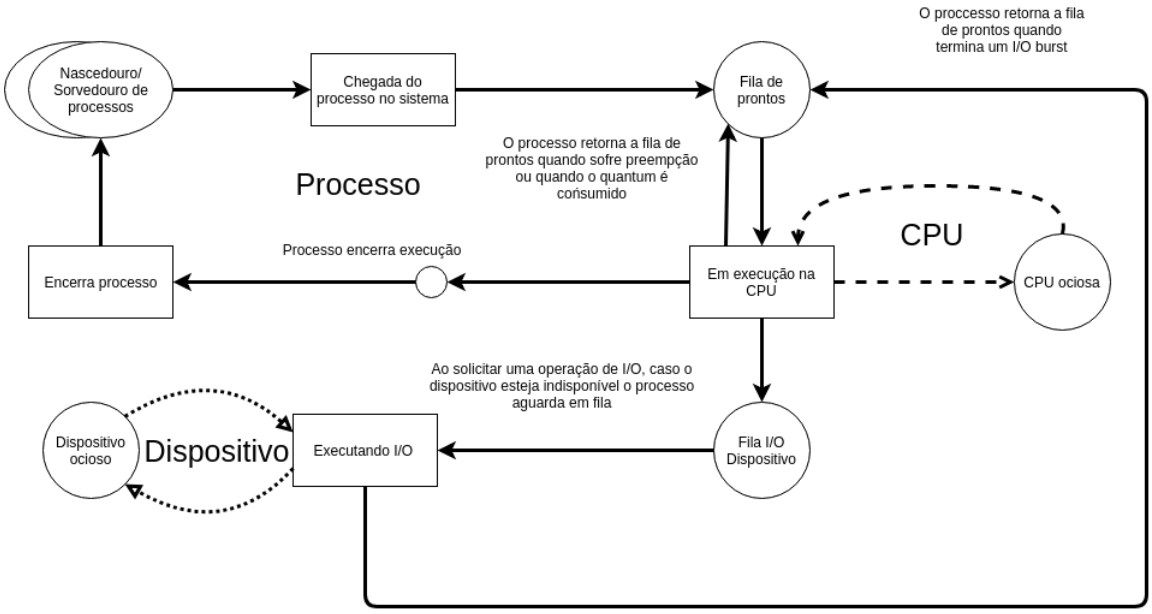
A princípio, houve um maior foco em investigar os componentes integrantes de um escalonador de processos, seu funcionamento e as dependências entre si para que fosse definida a lógica do modelo de simulação, tal como suas entidades e filas. Ainda nesta etapa, foram estudados em detalhes os métodos clássicos mais listados na literatura envolvida.

Posteriormente foi efetuada uma busca na literatura por documentos que tratavam de métodos computacionais para escalonamento de processos no sistema operacional com a utilização de técnicas de inteligência computacional. Dos artigos encontrados durante a investigação, foram selecionados os artigos de Ajmani (2013), Bashir et al. (2011) e Behera et al. (2012) pela facilidade de compreensão e de reprodução com o objetivo de que se tornasse menos exaustivo o entendimento do leitor a respeito dos métodos desenvolvidos pelos autores.

3.2.2 Modelagem do Processo de Escalonamento de CPUs

O DCA do simulador foi modelado conforme apresenta a Figura 16, contendo três entidades distintas, sendo elas Processo, CPU e Dispositivo de I/O, que interagem entre si durante todo o período de simulação. Os eventos são representadas por retângulos, neles uma ou mais entidades estão executando um determinado evento. Os eventos que integram o diagrama são a chegada de processo no sistema, a execução do processo na CPU, a execução de operação I/O de um processo e o encerramento do processo. As filas e atrasos são representadas por círculos. Um processo pode estar na Fila de Prontos, aguardando execução na CPU ou na fila de um determinado dispositivo aguardando execução de operação I/O. Os dispositivos e CPUs podem estar ociosos, aguardando para serem utilizados por um processo. O nascedouro/sorvedouro de processos é representado por círculos sobrepostos, e é onde os processos “nascem e morrem”.

Figura 16 – DCA do simulador de escalonamento de processos.



Fonte: Elaborado pelo autor.

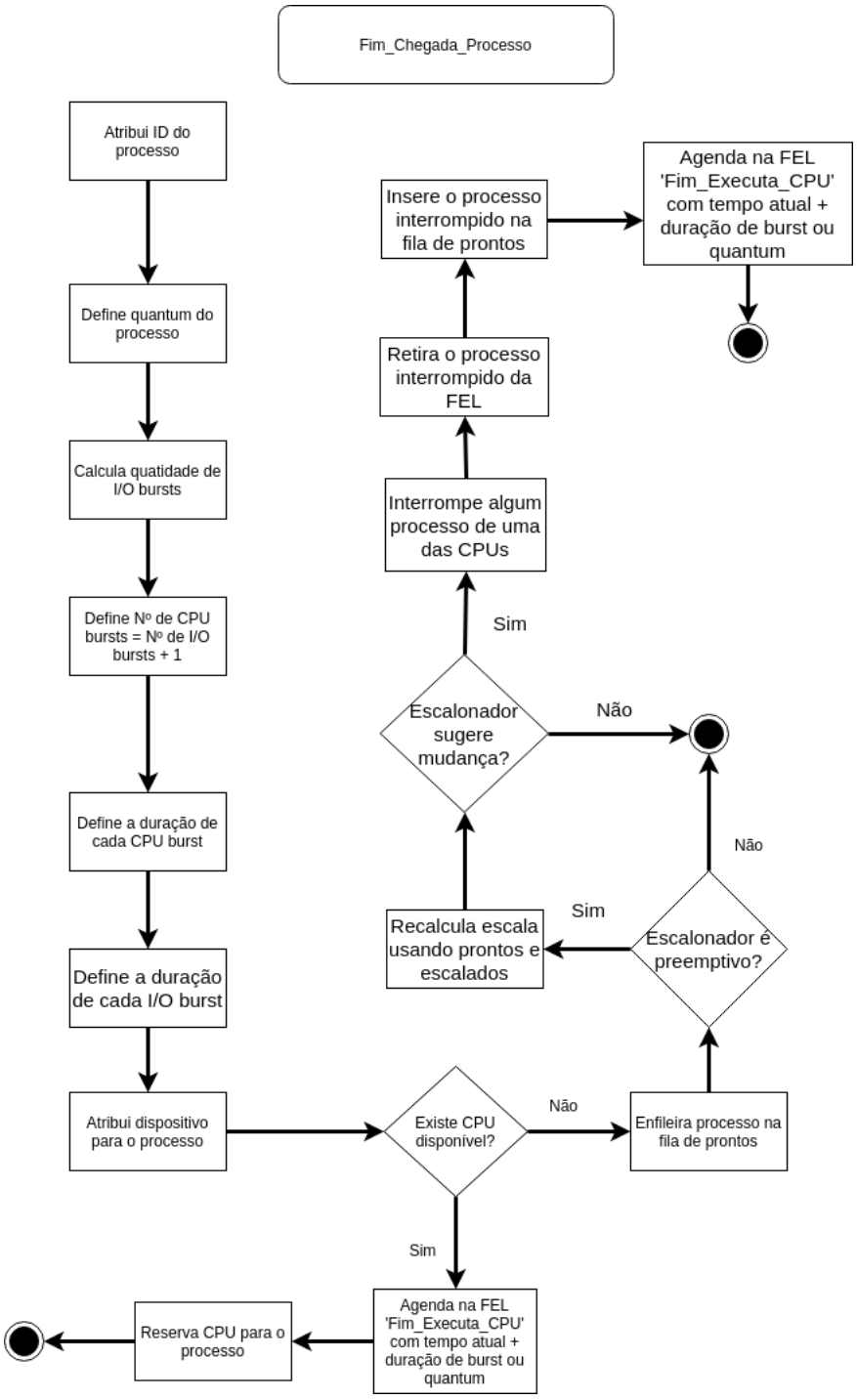
Cada atividade do diagrama é descrita em detalhes a seguir.

3.2.2.1 Evento: Fim da Chegada do Processo

Quando o processo chega ao sistema, são atribuídas a ele as informações que o definem, como o identificador, *quantum* (caso o método de escalonamento seja o RR), número de I/O *bursts* e CPU *bursts*, assim como suas durações e, em seguida, um dispositivo de I/O aleatório é atribuído para o processo.

Após a caracterização do processo, é verificado se existe uma CPU disponível para execução. Caso exista, a CPU é reservada e o processo é agendado. Caso não exista nenhuma CPU disponível para executar o processo, o mesmo é colocado na Fila de Prontos, onde aguarda por execução. Neste ponto, caso o algoritmo de escalonamento utilizado seja não-preemptivo, o evento é encerrado, caso seja preemptivo ele é então invocado para calcular se deverá ou não ocorrer alguma mudança de agendamento. Se o escalonador decidir efetuar alguma mudança, o processo que foi interrompido é removido da Lista de Eventos Futuros (FEL), a CPU que o executava é reservada para o novo processo selecionado pelo escalonador e este é agendado. O diagrama que demonstra esse evento pode ser observado na Figura 17.

Figura 17 – Diagrama do evento “Fim Chegada Processo”.



Fonte: Elaborado pelo autor.

3.2.2.2 Evento: Fim de Execução do Processo na CPU

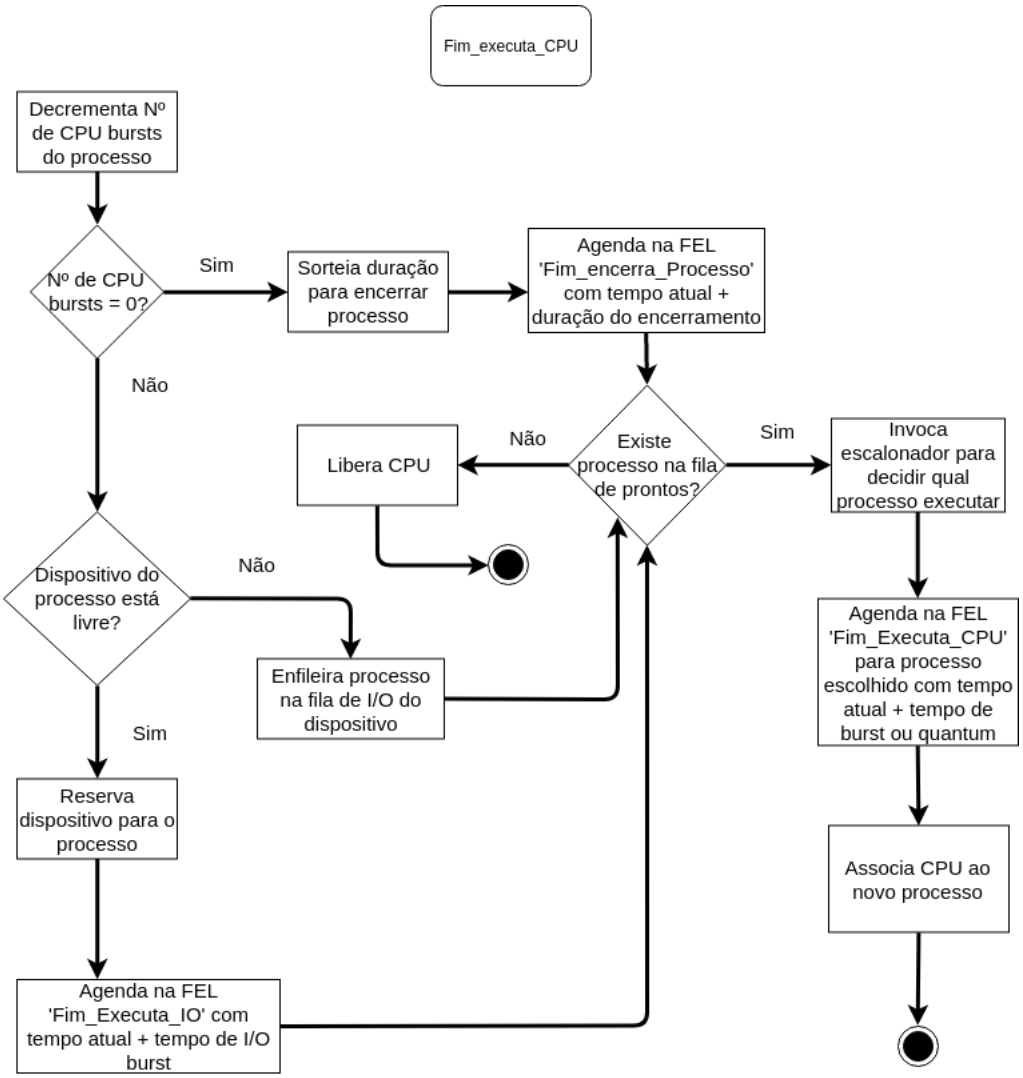
Ao término da execução na CPU, o processo tem seu número de CPU *bursts* decrementado e é verificado se esse número chegou a zero. Caso o número de CPU *bursts* do processo seja zero, significa que este terminou sua execução. É então sorteado um

tempo para o término do processo e o seu encerramento é agendado.

Após agendar o encerramento do processo, é verificado se existe algum processo na Fila de Prontos que esteja aguardando execução. Não existindo nenhum processo na Fila de Prontos, a CPU vai para o estado livre e o evento é encerrado. Existindo processo aguardando execução, o escalonador é invocado para decidir qual processo será agendado, realizando então a reserva da CPU e o agendamento da execução para o processo selecionado, encerrando assim o evento.

Tendo a confirmação de que o número de CPU *bursts* não chegou a zero, é verificado se o dispositivo atribuído para o processo está em estado livre e, se o dispositivo estiver ocupado em outra execução, o processo é colocado na fila I/O do dispositivo em questão. O dispositivo se encontrando no estado livre, este é então reservado para o processo e é agendada a execução de I/O. Em ambos os casos, após colocar o processo na fila I/O ou agendar execução de I/O, a Fila de Prontos é verificada. Se houver algum processo aguardando execução, o escalonador é invocado para selecionar o processo a ser agendado, a CPU é reservada para o processo e o agendamento é realizado, terminando assim o evento. Caso a Fila de Prontos esteja vazia, a CPU entra em estado livre e o evento é encerrado. No diagrama apresentado na Figura 18 é possível observar com clareza as transições.

Figura 18 – Diagrama do evento “Fim Executa CPU”.



Fonte: Elaborado pelo autor.

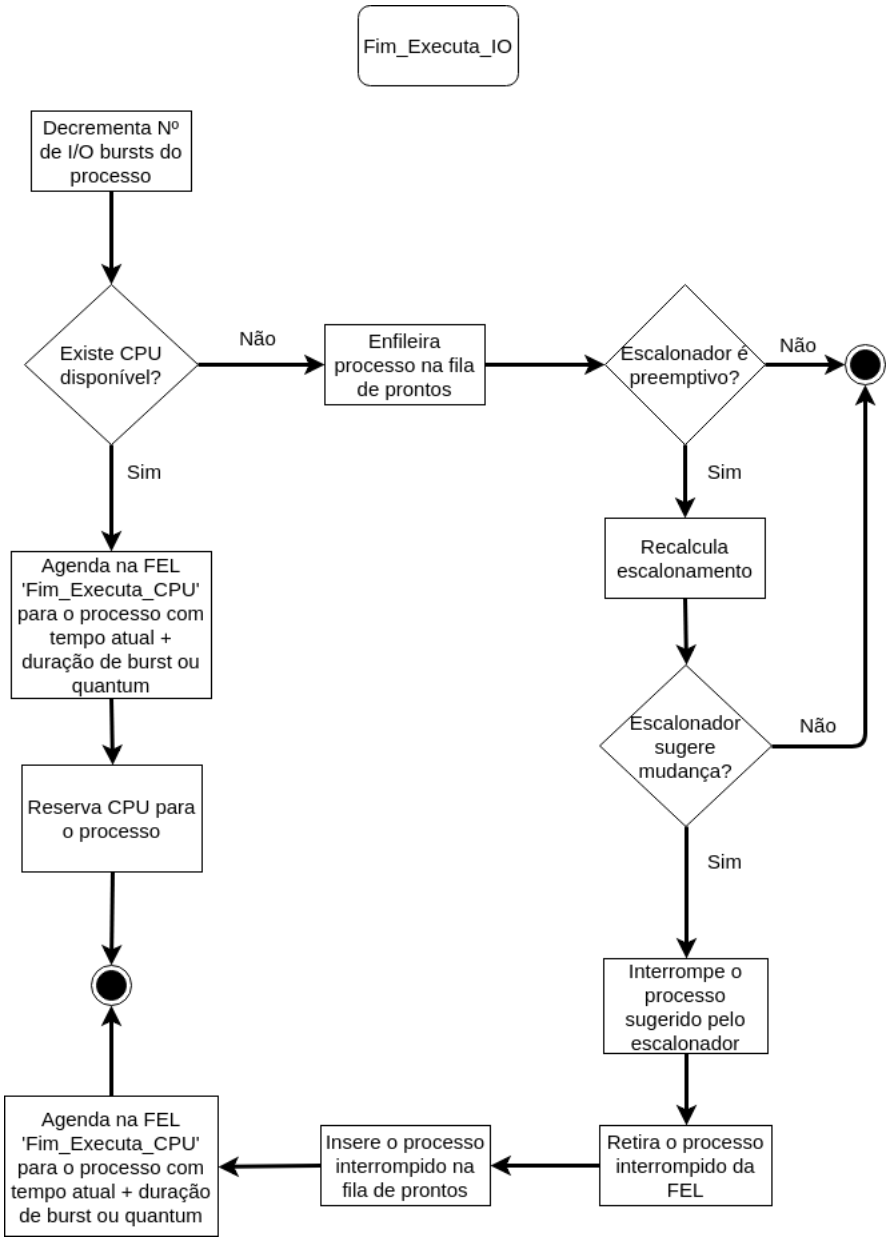
3.2.2.3 Evento: Fim Executa I/O

Ao encerrar uma execução I/O, o contador de I/O *bursts* do processo é decrementado e as CPUs do cenário são verificadas em busca de uma que esteja disponível para executar o próximo CPU *burst* do processo. Havendo CPU livre para execução, é feita a reserva da CPU e o agendamento da execução do processo, encerrando o evento. Caso todas as CPUs do cenário estejam ocupadas, o processo é inserido na Fila de Prontos.

Neste momento é verificado se o escalonador efetua preempção. Se não efetuar, o evento é encerrado. Sendo o escalonador preemptivo, este é invocado para recalcular o escalonamento, verificando os processos na fila de prontos e suas propriedades. Se o escalonador decidir interromper um processo para execução de outro, a CPU interrompida é associada ao novo processo e o processo interrompido é removido da FEL e inserido na

fila de prontos, encerrando o evento. O escalonador poderá decidir que nenhuma mudança de escala deve ser aplicada, nesta situação, o evento é encerrado. O diagrama que ilustra o evento pode ser observado na Figura 19.

Figura 19 – Diagrama do evento “Fim Executa I/O”.



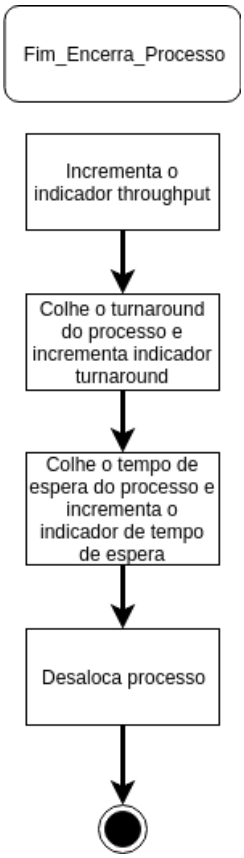
Fonte: Elaborado pelo autor.

3.2.2.4 Evento: Fim Encerra Processo

No encerramento do processo é onde são coletados seus dados para posteriormente efetuar as análises. Neste instante, o indicador de *throughput*, *turnaround* e tempo de espera são incrementados e o processo é desalocado do sistema. O diagrama que ilustra

este evento pode ser observado na Figura 20.

Figura 20 – Diagrama do evento “Fim Encerra Processo”.



Fonte: Elaborado pelo autor.

3.2.3 Simulador

Serão retratadas na presente seção todas as etapas de implementação do protótipo assim como o seu funcionamento e os recursos externos utilizados para auxiliar o desenvolvimento. Também serão expostas as decisões tomadas no decorrer do projeto, a validação e ganho de confiabilidade dos resultados e por fim a forma de interpretação dos dados obtidos.

3.2.3.1 Implementação

Para implementação do simulador, a linguagem Python foi escolhida devido à sua baixa curva de aprendizado, sua ampla funcionalidade e também por sua comunidade, que se mostrou sempre disposta a colaborar com os usuários da plataforma. A composição do simulador consiste em dez classes distintas, brevemente comentadas a seguir:

- **Coleções**: A classe Coleções abriga todos os objetos do tipo Processo, CPU e Dispositivo em listas distintas. Possui funções de busca de objetos, busca de CPUs que estejam livres e finalização de processos;

- **Processo**: A classe Processo armazena todas as informações de cada processo que faz parte da simulação, como tempo de chegada, número de bursts, prioridade e tempo de espera;

- **CPU**: A classe CPU armazena as informações das CPUs que constituem o modelo, como utilização e ociosidade; Dispositivo: Armazena as informações de cada dispositivo, como a fila de processos que aguardam utilização I/O e estado atual (livre ou ocupado);

- **Escalonador**: É uma classe abstrata que define o esqueleto de cada algoritmo de escalonamento, como as funções de seleção de processo e recálculo de escala; Fel: Classe que implementa a fila de eventos futuros (FEL), o algoritmo de passagem do tempo de simulação e o algoritmo de consumo de eventos;

- **Evento**: Nesta classe são implementadas as atividades do simulador, como a chegada de processos, execução na CPU e operação I/O. Também é a classe que envia eventos a serem agendados na FEL;

- **Cenário**: Classe que lê o arquivo de entrada com os dados necessários e os armazena para realização da simulação;

- **Fila**: Em Python existem apenas o objeto lista. Para criar uma pilha, fila ou outra estrutura, devemos utilizar a estrutura lista como base. Para isso é necessário implementar suas regras de manipulação. A classe fila é responsável por implementar as regras de manipulação da fila de prontos, como inserção, remoção e verificação de fila vazia.

- **Plot**: Classe que se mostrou essencial para validação do modelo e para utilização deste trabalho como ferramenta didática. A classe Plot é responsável por interpretar os tempos de espera e execução de cada processo assim como a ociosidade da CPU a fim de gerar gráficos de Gantt com as execuções dos processos e intervalos e gráficos de linhas com o percentual de utilização de cada CPU.

As bibliotecas do Python que simplificaram a realização deste projeto são descritas a seguir:

- **random**[1]: biblioteca de geração de números aleatórios. Possui funções para as principais distribuições de probabilidade utilizadas no projeto.

[1] Disponível em: <https://docs.python.org/3/library/random.html>

- **math**[2]: disponibiliza as principais funções matemáticas como mod(), div() e sum().
- **os**[3]: módulo que fornece a utilização de funções dependentes do sistema operacional. Suas principais funcionalidades são a manipulação de arquivos e pastas.
- **sys**[4]: módulo que fornece acesso a algumas variáveis mantidas pelo interpretador. Utilizada para captura de parâmetros fornecidos via linha de comando.
- **plotly**[5]: biblioteca que possibilita a geração de gráficos. Utilizada na validação do projeto.
- **scikit-fuzzy**[6]: módulo que disponibiliza ferramentas para se operar com lógica *fuzzy* em python.

Os diagramas de classe do simulador exibem todo o esqueleto de sua construção tal como suas variáveis, métodos e parâmetros. O leitor é convidado a visualizá-los no Apêndice A.

O simulador foi desenvolvido para operar em duas vertentes, determinístico e probabilístico. O modo determinístico aloca todos os recursos antes de iniciar a simulação e para um mesmo cenário de entrada sempre resultará na mesma ordem de agendamento dos processos e consequentemente sempre exibirá o mesmo resultado. O simulador no modo determinístico trabalha sempre com apenas uma CPU. O modo probabilístico foi desenvolvido para que se tivesse uma simulação mais próxima do mundo real, contando com a probabilidade de chegada de processos distintos em diferentes momentos com variadas características, dispositivos, operações de I/O e quantas CPUs o usuário desejar. Neste modo, as características dos processos são definidas por distribuições de probabilidade que usam como componentes as informações do arquivo de entrada.

É importante ressaltar que o projeto consiste de um protótipo de simulador de escalonador de processos, e, sendo assim, possui abstrações do mundo real que não são consideradas nos experimentos, como o trabalho realizado pelo escalonador de longo prazo e o tempo levado pelo escalonador para efetuar os cálculos e selecionar o processo a ser executado. Conceitos como memória e paginação também não estão presentes neste protótipo, mas o projeto tem estrutura para receber posteriormente estes módulos a fim de se tornar uma ferramenta ainda mais robusta.

O simulador de escalonador de processos **SimPro** foi desenvolvido com o intuito de ser uma ferramenta de código aberto gratuita para a comunidade, possibilitando que

[2] Disponível em: <https://docs.python.org/3/library/math.html>
[3] Disponível em: <https://docs.python.org/3/library/os.html>
[4] Disponível em: <https://docs.python.org/3/library/sys.html>
[5] Disponível em: <https://plot.ly/>
[6] Disponível em: <https://pythonhosted.org/scikit-fuzzy>

qualquer usuário possa fazer modificações, melhorias e até se aventurar em criar seu próprio algoritmo de escalonamento. O código fonte é disponibilizado na plataforma github através do endereço <https://github.com/danilodsa/SimPro>.

3.2.4 Entrada de Dados

Para que o simulador funcione corretamente e a simulação produza resultados, foi construído um modelo por onde são informados os dados da simulação de acordo com a vontade do usuário. Como dito anteriormente, o simulador opera em dois modos distintos, o probabilístico e o determinístico. Para cada modo existe um modelo específico de entrada de dados. Em ambos os casos, a entrada é feita mediante arquivo de configuração.

Figura 21 – Exemplo de cenário probabilístico.

```
#-------------------------------------------------------#
#             Cenario Basico Probabilistico             #
#-------------------------------------------------------#

S TS 500

#-------------------------------------------------------#
#                       Processo                        #
#-------------------------------------------------------#

P CH 15.0
P PR 1 5 10
P EN 0 0 0
P NI 1.0 2.0 5.0
P DI 5.0 10.0 15.0
P DC 1.0 2.0 5.0

#-------------------------------------------------------#
#                         CPU                           #
#-------------------------------------------------------#

C QT 1

#-------------------------------------------------------#
#                    Dispositivo                        #
#-------------------------------------------------------#

D QT 3
```

Fonte: Elaborado pelo autor.

Na Figura 21, podemos observar como devem ser os parâmetros informadas no cenário probabilístico. No arquivo, cada linha começa com um caractere marcador, usado para indicar ao simulador o que a linha contém. Os caracteres usados para descrever o cenário a ser simulado são os que seguem:

- `#: Comentários`
- `S: Simulação`

  - `TS: Informa quanto tempo será simulado, milissegundos`

- `P: Processo`

  - `CH: Tempo de chegada de processos.`
  - `PR: Prioridade do processo.`
  - `EN: Tempo de encerramento do processo`
  - `NI: Número de I/O bursts do processo`
  - `DI: Duração dos I/O bursts`
  - `DC: Duração dos CPU bursts`

- `C: CPU`

  - `QT: Quantidade de núcleos/CPUs que estarão presentes na simulação`

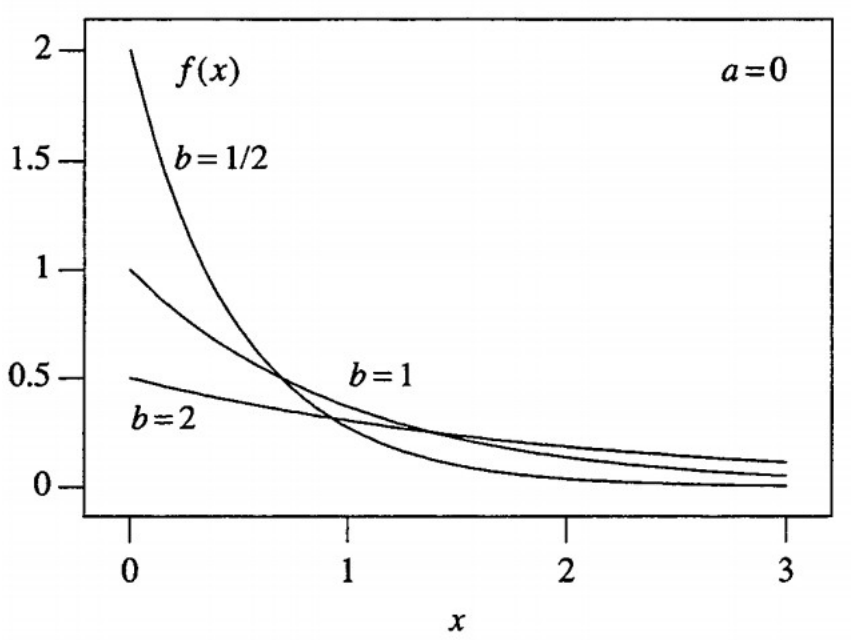
- `D: Dispositivo`

  - `QT: Quantidade de dispositivos de I/O que estarão presentes na simulação`

No cenário apresentado na Figura 21 acima, temos a seguinte caracterização para os processos. A chegada de processos ocorre em média a cada 15 milissegundos. Os processos recebem valor de prioridade de 1 a 10, onde a maioria dos processos tem prioridade 5. O número de I/O burst pode ser entre 1 e 5, sendo mais provável que processos tenham 2 I/O bursts. A duração das operações de I/O de cada processo variam entre 5 e 15 milissegundos, sendo maior a chance dessas operações ocorrerem por volta de 10 milissegundos. O mesmo ocorre com a duração dos picos de CPU, que variam entre 1 e 5, sendo mais provável que fiquem por volta de 2 milissegundos.

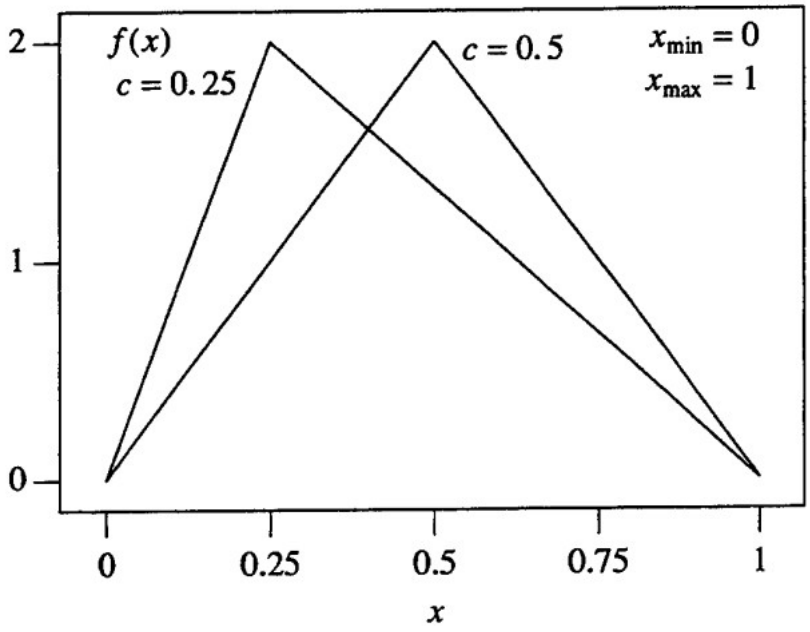
A chegada de processos (CH) obedece uma distribuição exponencial, que recebe como entrada um parâmetro b. A distribuição exponencial é representada pelo gráfico na Figura 22. Os parâmetros de entrada para definição de processos PR, EN, NI, DI e DC obedecem uma distribuição triangular, que possui como parâmetros três valores: mínimo (x_{min}), moda (c) e máximo (x_{max}). A distribuição triangular pode ser observada na Figura 23.

Figura 22 – Distribuição exponencial para diferentes valores de b.



Fonte: (SAUCIER; 2000)

Figura 23 – Distribuição triangular para diferentes valores de c.



Fonte: (SAUCIER; 2000)

Caso o leitor desejar se aprofundar nos conceitos matemáticos envolvido por trás

destas e outras distribuições de probabilidade é recomendada a leitura das obras Saucier (2000) e Walpole (1993).

Figura 24 – Exemplo de cenário determinístico.

```
1 #------------------------------------------------------#
2 #             Cenario Basico Deterministico            #
3 #------------------------------------------------------#
4
5 P 1 0 10 3
6 P 2 0 1 1
7 P 3 0 2 4
8 P 4 0 1 5
9 P 5 0 5 2
```

Fonte: Elaborado pelo autor.

A construção do arquivo de cenário determinístico se dá apenas informando as características de cada processo que fará parte da simulação no formato a seguir:

```
P [ID do processo] [tempo de chegada] [ tempo de CPU burst] [prioridade]
```

Considerando o arquivo de entrada exibido na Figura 24, se tem um total de 5 processos com identificador sequencial de 1 a 5. Todos os processos chegam no tempo 0 (zero), ou seja, ao inicializar a simulação. O tempos de pico de CPU são 10, 1, 2, 1 e 5 milissegundos para P_1, P_2, P_3, P_4 e P_5, respectivamente e a prioridade segue como 3 para P_1, 1 para P_2, 4 para P_3, 5 para P_4 e 2 para P_5. O simulador sempre considera o processo com maior valor de prioridade como sendo o mais prioritário.

3.2.5 Execução

O funcionamento do simulador se dá via linha de comando, utilizando o comando `python SimPro.py [método] [cenário.txt] [P/D]`, onde **método** corresponde ao algoritmo de escalonamento desejado e **cenário** será um arquivo de texto escrito no formato determinístico ou probabilístico apresentado e **P** ou **D** identificam se o cenário fornecido é probabilístico ou determinístico, respectivamente. É importante que o cenário e a identificação correspondam, pois cada método depende de alocação de recursos e formas de execução distintas.

Para exemplificar, seja o arquivo cenario01.txt, determinístico, exibido na Figura 24 acima. Para invocá-lo como entrada de dados e utilizar o método de escalonamento por prioridades, a linha de comando correspondente deverá ser:

```
python SimPro.py PRTY cenario01.txt D
```

Uma relação de cada método e sua abreviatura para parâmetro de invocação é exibida na Tabela 2.

Tabela 2 – Parâmetros de invocação para cada método de escalonamento.

Método	Parâmetro
Primeiro a Entrar, Primeiro a ser Atendido (*First-Come, First-Served*)	FCFS
Trabalho mais Curto Primeiro (*Shortest Job First*)	SJF
Próximo de Menor Tempo Restante (*Shortest Remaining Time First*)	SRT
Chaveamento Circular (*Round-Robin*)	RR
Prioridades	PRTY
Fuzzy Priority CPU Scheduling	FPCS
Improved Fuzzy-Based CPU Scheduling	IFCS
Proposed Fuzzy CPU Scheduling	PFCS

Fonte: Elaborado pelo autor.

3.2.6 Validação do modelo

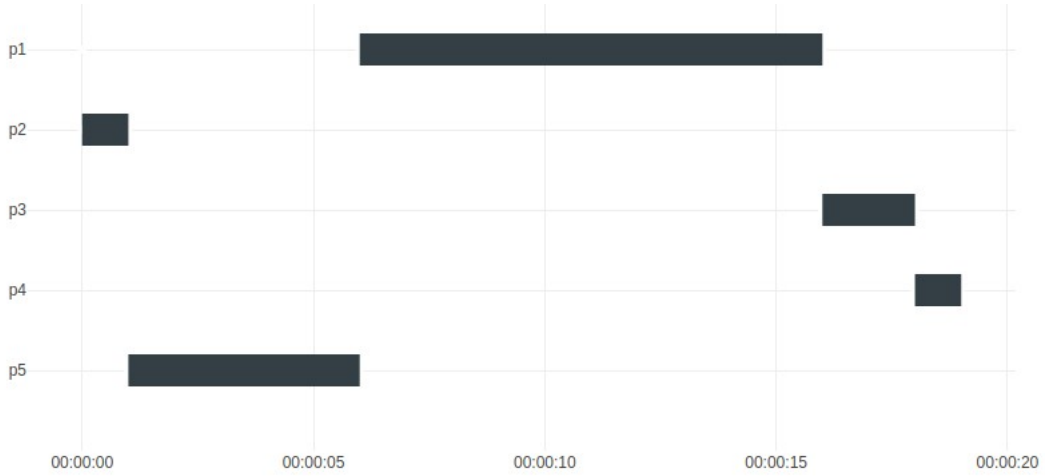
A implementação determinística foi de suma importância para o andamento do projeto, com ela foi possível executar os mesmos *benchmarks* utilizados pelos autores mencionados e pelos livros técnicos da literatura especializada para confirmar o funcionamento do simulador. Outra ferramenta que possibilitou a validação do protótipo foi a biblioteca **plotly**, que tornou possível a reprodução do gráfico de Gantt de toda a simulação, como apresentado na Figura 25, para que se pudesse observar o momento do acontecimento de cada evento e efetuar o comparativo com os datasets dos autores citados neste documento. Esta etapa se mostrou importante por tornar possível identificar e corrigir falhas na construção do simulador, até o momento em que foram obtidos os mesmos resultados encontrados pelos autores.

Um exemplo de dataset pode ser encontrado a seguir, extraído de Silberschatz (2013) onde a coluna **Processo** representa o identificador de cada processo participante da simulação, a coluna **Chegada** exibe o momento exato da chegada de cada processo ao sistema, a coluna **Tempo de Pico** determina a fatia de tempo que o processo irá executar na CPU e por fim a coluna **Prioridade**, que define a prioridade de cada processo:

Processo	Chegada	Tempo de Pico	Prioridade
P_1	0	10	3
P_2	0	1	5
P_3	0	2	2
P_4	0	1	1
P_5	0	5	4

Ao executar este conjunto de processos no simulador com o algoritmo de escalonamento por prioridades, tendo o maior valor como maior prioridade de execução, obtemos o gráfico de agendamento apresentado na Figura 25. A sequência de agendamentos obtida corresponde aos resultados apresentados pelo autor.

Figura 25 – Gráfico de Gantt da simulação com algoritmo de escalonamento por prioridades gerado pelo simulador SimPro.



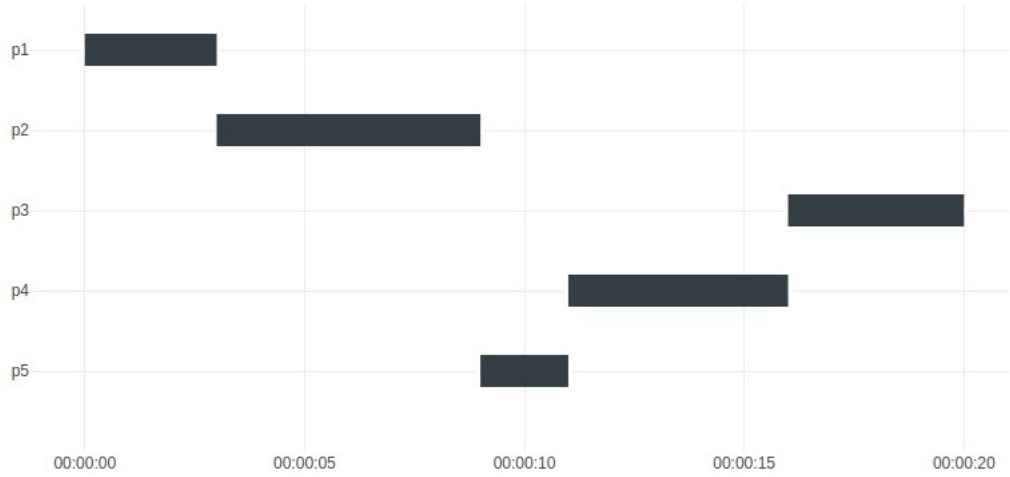
Fonte: Elaborado pelo autor.

Agora considere o conjunto de processos a seguir, obtido de Behera et al. (2012):

Processo	Chegada	Tempo de Pico	Prioridade
P1	0	10	3
P2	0	1	5
P3	0	2	2
P4	0	1	1
P5	0	5	4

Utilizando o algoritmo de escalonamento IFCS, desenvolvido por Behera et al. (2012), obtemos as prioridades dinâmicas calculadas para cada processo em tempo de execução: P1= 0.571; P2 = 0.875; P3 = 0.625; P4 = 0.75; P5 = 0.82. A sequência de agendamentos obtida corresponde aos resultados apresentados pelo autor, e encontram-se ilustradas na Figura 26.

Figura 26 – Gráfico de Gantt da simulação com algoritmo de escalonamento IFCS gerado pelo simulador SimPro.



Fonte: Elaborado pelo autor.

A fim de se obter maior confiabilidade na validação do experimento, todo os *datasets* apresentados por Ajmani(2013), Behera et al.(2011) e Silberschatz(2013) foram submetidos à simulação no simulador **SimPro** para todos os métodos de escalonamento apresentados. Somente após obter a mesma sequência de agendamentos reportada pelos autores em seus trabalhos foi que se pôde declarar o simulador como validado. Não foi possível replicar os experimentos feitos por Bashir et al. (2012), pois este utilizou como *dataset* uma sequência de processos com características geradas de forma aleatória, não divulgada em nenhuma obra.

3.2.7 Projeto Experimental

Para condução do experimento, foram criados cenários de execução variados para que fosse possível efetuar uma análise do comportamento de cada algoritmo em diferentes ambientes CPU *bound* e I/O *bound*. Os arquivos de entrada determinísticos foram construídos exatamente como nos experimentos executados pelos autores mencionados neste documento. Já os cenários probabilísticos foram pensados levando em conta um sistema altamente carregado, com a maior parte das chegadas de processos ocorrendo a cada 15 milissegundos. Em todos os cenários, o tempo de encerramento de processos não foi considerado. Essa decisão foi tomada pois o impacto dessa medida na simulação não foi relevante, mas escolheu-se manter esta funcionalidade para que, se desejado, fosse utilizado pelo usuário. Para todos os cenários, o *quantum* considerado para o algoritmo RR foi de 4 (quatro) milissegundos. Este tempo de *quantum* foi escolhido para que fosse obtida uma fila de espera razoável, com cada burst sendo finalizado, na maior parte dos casos em

5 (cinco) *quanta* e cada processo chegando em média a cada 4 (quatro) *quanta*. Foram simuladas execuções com *quatum* de 10 milissegundos, mas o algoritmo RR se comportou como o algoritmo FCFS.

Cada cenário experimental foi simulado 100 vezes para cada algoritmo de escalonamento, simulando um tempo de CPU de 500 milissegundos. Isso foi necessário para que fossem obtidos quantidades de valores suficientemente confiáveis para se embasar a análise. Como os experimentos são probabilísticos, se faz necessária sua replicação para que o máximo da gama de possibilidades seja explorado para se ter resultados analíticos confiáveis. Foram feitos experimentos em cenários que simulavam 1 segundo (1000 milissegundos), porém notou-se que os valores foram proporcionais aos experimentos que simulavam 500 milissegundos. Optou-se por manter o tempo a ser simulado em 500 milissegundos, que corresponde a cerca de dez vezes o tempo simulado pelos autores citados.

3.2.7.1 Cenário I

O primeiro cenário experimental foi construído com objetivo de analisar o comportamento de cada método de escalonamento em um sistema com um único processador em situação de alta carga de processamento, com alta demanda de chegada de processos. A escolha do tempo de *burst* foi feita para que houvesse maior quantidade de chegada de processos com *bursts* de 20 milissegundos e, ocasionalmente, processos grandes (até 100 milissegundos). O arquivo de entrada pode ser observado na Tabela 3.

Tabela 3 – Configuração do cenário I.

Descrição	Parâmetro	Valor
Tempo de Simulação	S TS	500
Chegada de processos	P CH	15
Prioridade dos processos	P PR	1 5 10
Nº de I/O bursts	P NI	0 0 0
Duração dos I/O bursts	P DI	0 0 0
Duração dos CPU bursts	P DC	1 20 100
Quantidade de núcleos	C QT	1
Quantidade de dispositivos	D QT	0

Fonte: Elaborado pelo autor.

É possível notar através dos dados da Tabela 3 que o cenário trata apenas de execuções na CPU. Cenários com execuções I/O serão descritos posteriormente, em outros cenários.

3.2.7.2 Cenário II

O segundo cenário experimental consiste nas mesmas características do primeiro, com a exceção do número de CPUs, que saiu de 1 (um) núcleo no Cenário I para 2 (dois) núcleos no Cenário II. Neste caso, a intenção foi simular um processador com dois núcleos e averiguar o comportamento dos algoritmos de escalonamento quando se deve escalonar processos dividindo a carga entre núcleos distintos. Os detalhes da configuração deste cenário podem ser observados na Tabela 4.

Tabela 4 – Configuração do cenário II.

Descrição	Parâmetro	Valor
Tempo de Simulação	S TS	500
Chegada de processos	P CH	15
Prioridade dos processos	P PR	1 5 10
Nº de I/O bursts	P NI	0 0 0
Duração dos I/O bursts	P DI	0 0 0
Duração dos CPU bursts	P DC	1 20 100
Quantidade de núcleos	C QT	2
Quantidade de dispositivos	D QT	0

Fonte: Elaborado pelo autor.

3.2.7.3 Cenário III

Como no primeiro e no segundo cenário, o terceiro caso de estudos também possui um comportamento puramente de execuções na CPU, ou seja, não existem operações de I/O. O cenário opera como os dois anteriores, porém simulando um processador com quatro núcleos. A Tabela 5 exibe o arquivo de configuração do presente cenário.

Tabela 5 – Configuração do cenário III.

Descrição	Parâmetro	Valor
Tempo de Simulação	S TS	500
Chegada de processos	P CH	15
Prioridade dos processos	P PR	1 5 10
Nº de I/O bursts	P NI	0 0 0
Duração dos I/O bursts	P DI	0 0 0
Duração dos CPU bursts	P DC	1 20 100
Quantidade de núcleos	C QT	4
Quantidade de dispositivos	D QT	0

Fonte: Elaborado pelo autor.

3.2.7.4 Cenário IV

Nessa parte do experimento, foram incluídas as operações de I/O variando entre 1 e 5 ocorrências de operações com tempo de duração do I/O *burst* variando entre 5 e 15 milissegundos. Nos casos de estudo onde foram incluídas operações de I/O, o tempo de CPU *burst* foi reduzido para que fosse simulado um cenário mais interativo, onde os processos gastam mais operações de I/O e menos tempo em operações na CPU. Este cenário conta apenas com um único núcleo de processamento e um único dispositivo. A tabela 6 abaixo exibe em detalhes a configuração do

Tabela 6 – Configuração do cenário IV.

Descrição	Parâmetro	Valor
Tempo de Simulação	S TS	500
Chegada de processos	P CH	15
Prioridade dos processos	P PR	1 5 10
Nº de I/O bursts	P NI	1 2 5
Duração dos I/O bursts	P DI	5 10 15
Duração dos CPU bursts	P DC	1 5 10
Quantidade de núcleos	C QT	1
Quantidade de dispositivos	D QT	1

Fonte: Elaborado pelo autor.

3.2.7.5 Cenário V

No quinto cenário experimental, os valores para os parâmetros são como no cenário anterior. Observando a configuração do cenário na Tabela 7 é possível perceber que foi mantida a frequência de chegada de processos. Isso foi definido para que fosse manipulado sempre um sistema carregado e sob constante estresse computacional. O número de I/O *bursts* tal como sua duração foi mantida para que fosse possível analisar as mudanças em relação ao cenário anterior.

Tabela 7 – Configuração do cenário V.

Descrição	Parâmetro	Valor
Tempo de Simulação	S TS	500
Chegada de processos	P CH	15
Prioridade dos processos	P PR	1 5 10
Nº de I/O bursts	P NI	1 2 5
Duração dos I/O bursts	P DI	5 10 15
Duração dos CPU bursts	P DC	1 5 10
Quantidade de núcleos	C QT	1
Quantidade de dispositivos	D QT	3

Fonte: Elaborado pelo autor.

3.2.7.6 Cenário VI

O sexto cenário experimental, construído de acordo com a Tabela 8 foi construído tendo em mente um sistema com mais recursos, ainda em uma situação de alta demanda de processos ingressando ao sistema. Neste cenário, contamos com quatro núcleos e quatro dispositivos de I/O. O objetivo desta fase do experimento é verificar como os algoritmos de escalonamento se comportam ao efetuar a manipulação de vários recursos.

Tabela 8 – Configuração do cenário VI.

Descrição	Parâmetro	Valor
Tempo de Simulação	S TS	500
Chegada de processos	P CH	15
Prioridade dos processos	P PR	1 5 10
Nº de I/O bursts	P NI	1 2 5
Duração dos I/O bursts	P DI	5 10 15
Duração dos CPU bursts	P DC	1 5 10
Quantidade de núcleos	C QT	4
Quantidade de dispositivos	D QT	4

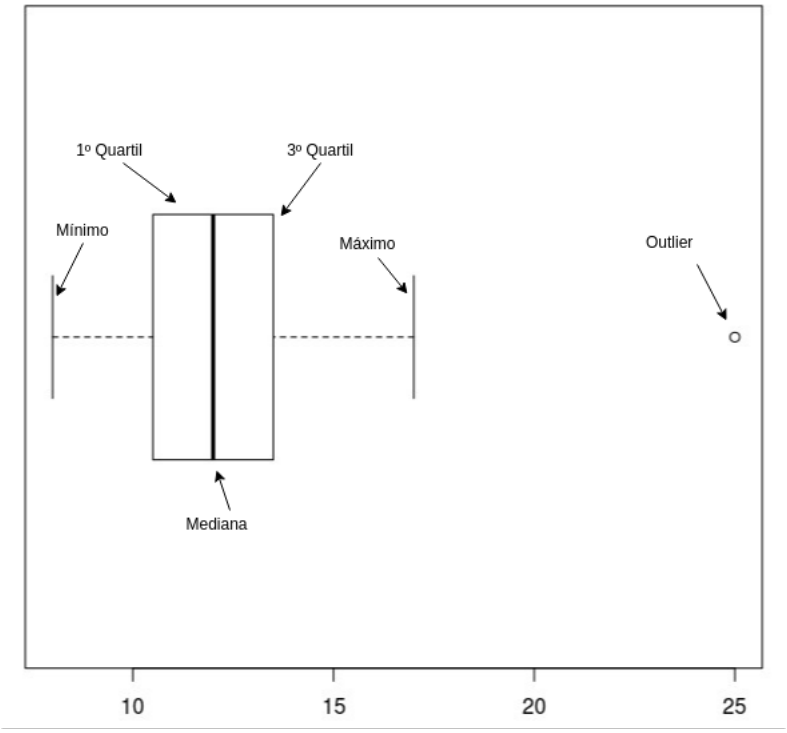
Fonte: Elaborado pelo autor.

3.2.8 Análise de Resultados

A análise dos resultados é feita mediante gráficos do tipo *boxplot*. Potter (2006) explica o *boxplot* como sendo um gráfico construído com base no resumo de cinco números: valor mínimo; primeiro quartil ou 25%; mediana (segundo quartil) ou 50%; terceiro quartil ou 75%; e valor máximo. O gráfico é formado por uma caixa construída paralelamente ao eixo de escala dos dados. Esta caixa engloba do primeiro ao terceiro quartil e nela é

traçada uma linha na posição da mediana. Posteriormente, uma linha é traçada paralela a o eixo dos dados, que vai de cada extremidade da caixa ao valor extremo dos dados. Existem ainda valores atípicos, ou *outliers*, que são valores muito afastados da grande maioria dos dados. Estes valores são marcados fora do eixo que engloba os cinco valores mencionados anteriormente, como mostra a Figura 27.

Figura 27 – Exemplo de gráfico boxplot.



Fonte: Elaborado pelo autor.

No *boxplot* apresentado na Figura 27, é possível observar o valor 8.00 como sendo o valor mínimo da amostra. O primeiro quartil tem início no valor 10.75 e segue até o valor 12.00, que corresponde à mediana da amostra. O terceiro quartil tem início na mediana e segue até o valor 13.25. O valor máximo da amostra (não incluindo outliers) é 17.00. O valor mais afastado, 25.00, corresponde a um *outlier*, que é um valor fora do intervalo esperado. Nesta interpretação gráfica obtemos, portanto, as medidas de dispersão da amostra, uma noção de sua amplitude (maior - menor), uma noção comparativa sobre a variabilidade dos resultados observados, e uma indicação de valores anômalos segundo a variância da amostra.

4 RESULTADOS E DISCUSSÃO

Nesta seção serão descritos os resultados obtidos com a realização dos experimentos simulados nos cenários apresentados. Um comparativo entre os algoritmos de escalonamento será exibido em forma de gráficos *boxplot* e tabelas com os valores do sumário estatístico dos resultados observados nas 100 replicações independentes do modelo de simulação para as variáveis tempo de resposta, *throughput*, *turnaround* e utilização da CPU nos Cenários I, II, III, IV, V e VI propostos. Cada subseção a seguir reporta e discute cada cenário.

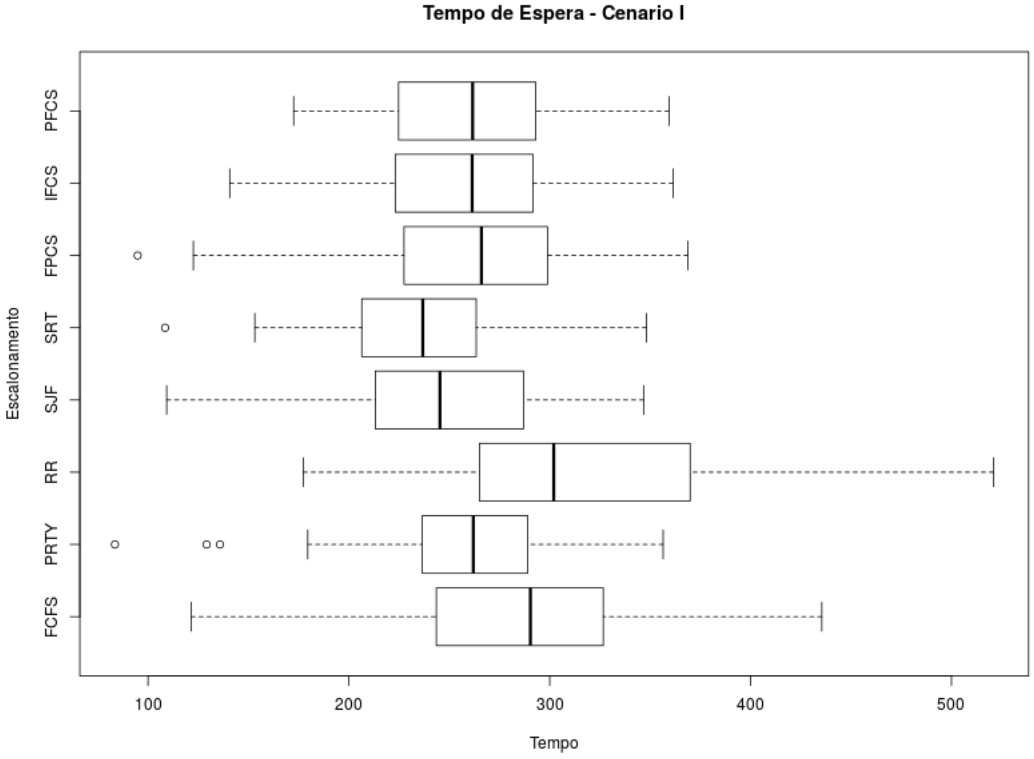
4.1 Cenário I

Esta seção reporta os resultados encontrados para as variáveis tempo de espera, *throughput*, *turnaround* e utilização da CPU quando executando-se o simulador com entrada caracterizada segundo o Cenário I.

4.1.1 Variável "Tempo de Espera"

Para iniciar, discutiremos os resultados referentes à variável tempo de espera, cujo gráfico *boxplot* comparativo entre todos os métodos testados encontra-se disponível na Figura 28.

Figura 28 – Resultado observado para a variável Tempo de Espera obtida pela execução de 100 replicações independentes dos algoritmos de escalonamento PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS sobre a caracterização de processos do Cenário I (único processador, alta taxa de chegada de processos, e alta carga de processamento).



Fonte: Elaborado pelo autor.

Pela Figura 28 é possível o tempo de espera de cada algoritmo de escalonamento e sua variação quando aplicado ao primeiro cenário experimental. É possível notar, analisando o valor da mediana, que o tempo de espera dos algoritmos PFCS e IFCS, que utilizam abordagem *fuzzy*, foi superior ao algoritmo de escalonamento por prioridades clássico (PRTY), enquanto o algoritmo FPCS, que também utiliza *fuzzy*, obteve o mesmo desempenho quando comparado ao algoritmo PRTY. Os algoritmos *fuzzy* ficam em desvantagem se comparados com os algoritmos SJF e SRT. Isso acontece pois a frequência de chegada de processos menores é maior do que a de processos maiores, fazendo com que, executar processos menores primeiro reduz mais o tempo de espera destes processos do que aumenta o tempo de espera dos processos maiores. Mas é importante ressaltar que os algoritmos SRT e SFJ facilitam a ocorrência de inanição de processos, enquanto os algoritmos baseados em *fuzzy* trabalham com o tempo de espera para evitar que isso aconteça.

É perceptível que o algoritmo RR teve um tempo de espera médio muito elevado quando comparado aos outro métodos. Isso ocorre pois, com apenas uma CPU, a fila de prontos está em constante crescimento, enquanto apenas um processo executa por vez. Isso aumenta o tempo de espera de cada processo, aumentando consequentemente o tempo de espera médio. As informações gerais obtidas através do estudo neste cenário podem ser observadas na Tabela 9.

Tabela 9 – Resumo com a estatística descritiva dos resultados observados para a variável Tempo de Espera obtidos a partir de 100 replicações independentes do modelo de simulação para os algoritmos PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS quando experimentados sobre a caracterização do Cenário I (único processador, alta demanda de processamento, alta taxa de chegada de novos processos), com destaque para os quatro melhores resultados, segundo a mediana.

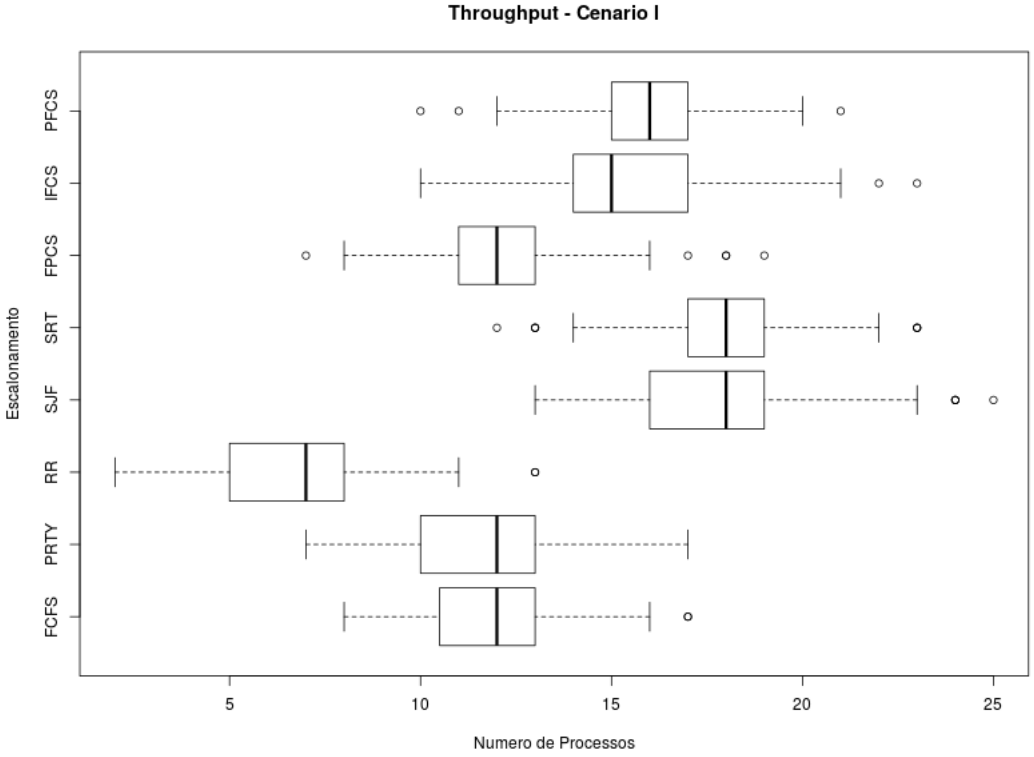
Método	Mínimo	1º Quartil	Mediana	Média	3º Quartil	Máximo
PFCS	172.50	224.80	**261.60**	261.50	293.1	359.40
IFCS	140.70	223.10	**261.40**	256.80	291.6	361.40
FPCS	94.83	227.46	266.00	263.04	298.89	368.82
SRT	108.05	206.40	**236.80**	236.50	263.40	348.10
SJF	109.03	213.20	**245.40**	249.10	286.90	346.70
RR	177.30	265.00	302.00	315.70	369.90	520.90
PRTY	83.54	236.55	261.96	260.56	289.02	356.46
FCFS	121.50	243.60	290.40	288.40	326.70	435.50

Fonte: Elaborado pelo autor.

4.1.2 Variável "*Throughput*"

Analisando o gráfico apresentado na Figura 29, é possível ver qual foi a vazão de processos desempenhada por cada algoritmo aplicado ao cenário apresentado.

Figura 29 – Resultado observado para a variável *Throughput* obtida pela execução de 100 replicações independentes dos algoritmos de escalonamento PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS sobre a caracterização de processos do Cenário I (único processador, alta taxa de chegada de processos, e alta carga de processamento).



Fonte: Elaborado pelo autor.

Os algoritmos PFCS e IFCS só são ultrapassados pelos algoritmos SRT e SJF devido a alta demanda de processos curtos. Já o algoritmo FPCS obteve a mesma mediana dos algoritmos PRTY e FCFS, porém, ao analisar a amplitude dos resultados, o FPCS acaba se destacando. Como ocorreu para o indicador Tempo de Espera, o algoritmo RR fica em desvantagem devido à alta demanda de processos a serem executados por apenas uma CPU. Os dados detalhados podem ser observados na Tabela 10.

Tabela 10 – Resumo com a estatística descritiva dos resultados observados para a variável *Throughput* obtidos a partir de 100 replicações independentes do modelo de simulação para os algoritmos PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS quando experimentados sobre a caracterização do Cenário I (único processador, alta demanda de processamento, alta taxa de chegada de novos processos), com destaque para os quatro melhores resultados, segundo a mediana.

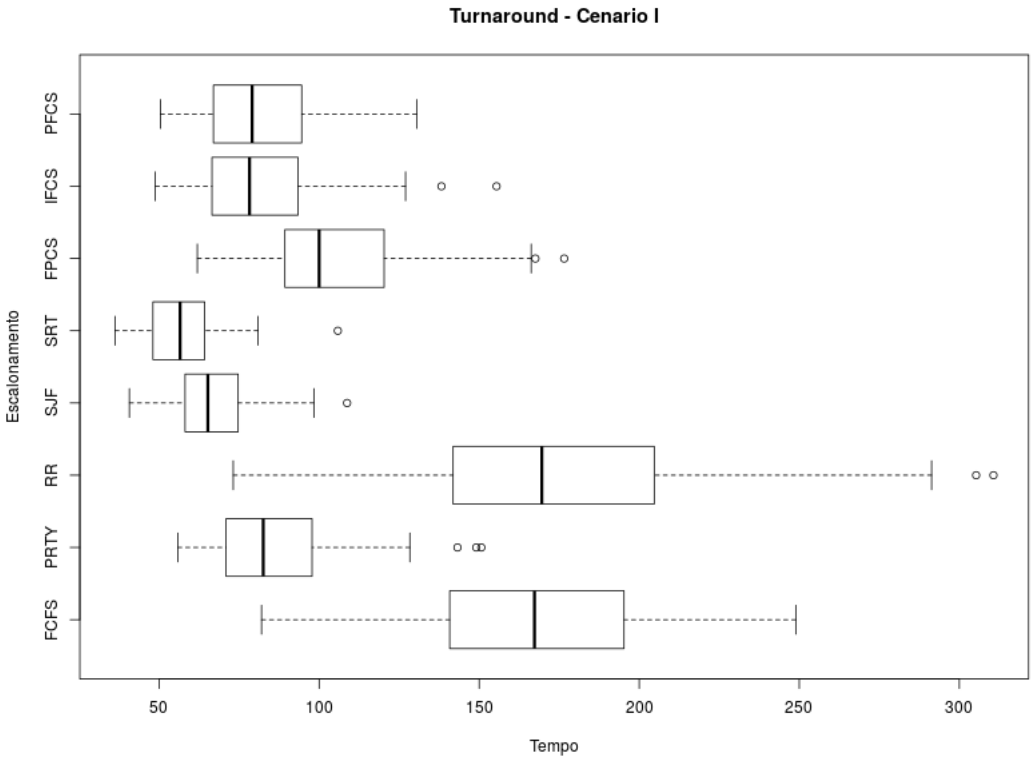
Método	Mínimo	1º Quartil	Mediana	Média	3º Quartil	Máximo
PFCS	10.00	15.00	**16.00**	15.79	17.00	21.00
IFCS	10.00	14.00	**15.00**	15.76	17.00	23.00
FPCS	7.00	11.00	12.00	12.41	13.00	19.00
SRT	12.00	17.00	**18.00**	17.93	19.00	23.00
SJF	13.00	16.00	**18.00**	18.08	19.00	25.00
RR	2.00	5.00	7.00	6.87	8.00	13.00
PRTY	7.00	10.00	12.00	11.66	13.00	17.00
FCFS	8.00	10.50	12.00	11.94	13.00	17.00

Fonte: Elaborado pelo autor.

4.1.3 Variável "*Turnaround*"

A Figura 30 ilustra o gráfico gerado com o objetivo de comparar o tempo de *turnaround* resultante de cada método de escalonamento implementado no simulador quando aplicado às condições do Cenário I.

Figura 30 – Resultado observado para a variável *Turnaround* obtida pela execução de 100 replicações independentes dos algoritmos de escalonamento PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS sobre a caracterização de processos do Cenário I (único processador, alta taxa de chegada de processos, e alta carga de processamento).



Fonte: Elaborado pelo autor.

Ao analisarmos o tempo de *turnaround* deste cenário, é possível reconhecer que os algoritmos PFCS e IFCS apenas não obtiveram vantagem em relação ao SRT e SJF, novamente devido a alta demanda de pequenos processos e a inferior probabilidade de chegada de processos maiores. Neste indicador, o algoritmo FPCS não se saiu muito bem, tendo mediana menor apenas que o RR e FCFS. Os dados detalhados podem ser observados na Tabela 11.

Tabela 11 – Resumo com a estatística descritiva dos resultados observados para a variável *Turnaround* obtidos a partir de 100 replicações independentes do modelo de simulação para os algoritmos PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS quando experimentados sobre a caracterização do Cenário I (único processador, alta demanda de processamento, alta taxa de chegada de novos processos), com destaque para os quatro melhores resultados, segundo a mediana.

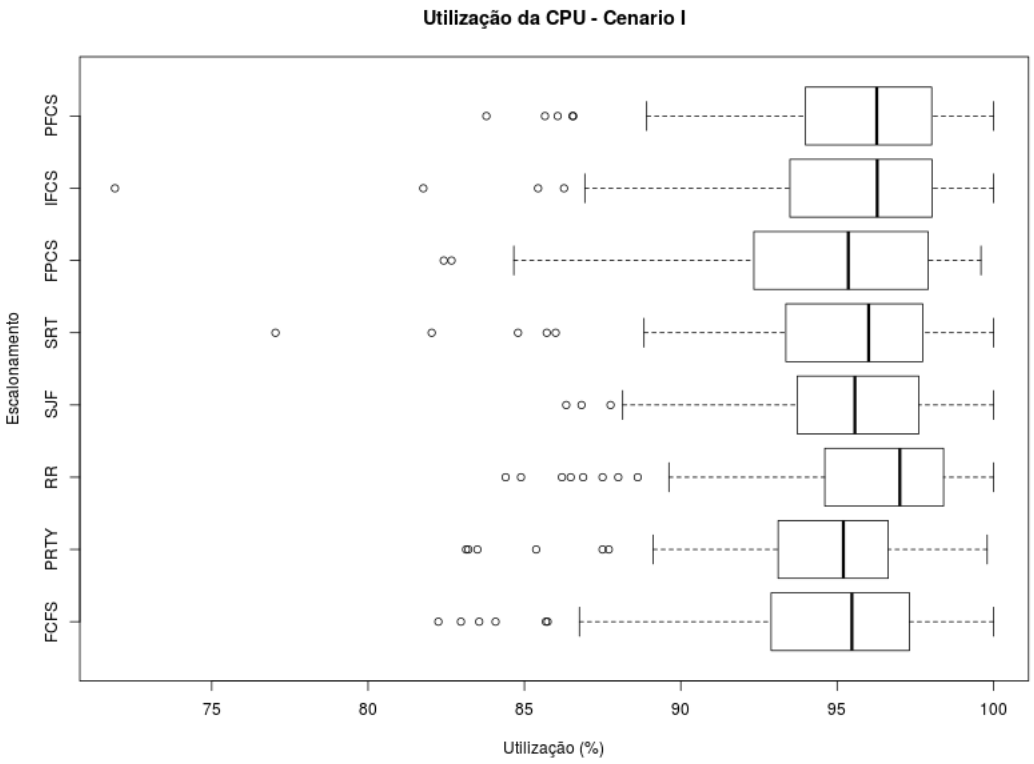
Método	Mínimo	1º Quartil	Mediana	Média	3º Quartil	Máximo
PFCS	50.40	66.97	**79.07**	82.01	94.55	130.46
IFCS	48.82	66.52	**78.25**	81.72	93.41	155.43
FPCS	61.94	89.30	100.00	106.25	120.31	176.56
SRT	36.21	48.08	**56.56**	56.46	64.24	105.84
SJF	40.71	58.12	**65.24**	66.78	74.70	108.73
RR	73.17	141.86	169.57	176.77	204.69	310.67
PRTY	55.82	70.93	82.54	86.53	97.79	150.67
FCFS	82.0	140.8	167.3	166.8	195.3	248.9

Fonte: Elaborado pelo autor.

4.1.4 Variável "Utilização da CPU"

No gráfico apresentado na Figura 31 é possível acompanhar o aproveitamento que cada algoritmo fez ao utilizar a CPU, em termos percentuais.

Figura 31 – Resultado observado para a variável Utilização da CPU obtida pela execução de 100 replicações independentes dos algoritmos de escalonamento PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS sobre a caracterização de processos do Cenário I (único processador, alta taxa de chegada de processos, e alta carga de processamento).



Fonte: Elaborado pelo autor.

Em se tratando do indicador “Utilização da CPU”, todos os algoritmos mantiveram uso efetivo da CPU acima dos 95% do tempo, na grande maioria das simulações, sendo que em algumas replicações chegou-se a atingir 100%. É importante observar que houveram ocorrência de outliers, tendo como execução com menor aproveitamento de CPU o algoritmo IFCS seguido do SRT. Com exceção dos dois mencionados, nenhum algoritmo teve utilização mínima da CPU inferior a 80%. Os valores detalhados de cada algoritmo podem ser observados na Tabela 12.

Tabela 12 – Resumo com a estatística descritiva dos resultados observados para a variável Utilização da CPU obtidos a partir de 100 replicações independentes do modelo de simulação para os algoritmos PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS quando experimentados sobre a caracterização do Cenário I (único processador, alta demanda de processamento, alta taxa de chegada de novos processos), com destaque para os quatro melhores resultados, segundo a mediana.

Método	Mínimo	1º Quartil	Mediana	Média	3º Quartil	Máximo
PFCS	83.78	93.98	**96.27**	95.47	98.03	100.00
IFCS	71.91	93.49	**96.28**	95.12	98.04	100.00
FPCS	82.42	92.34	95.36	94.55	97.91	99.60
SRT	77.04	93.36	**96.01**	95.06	97.74	100.00
SJF	86.34	93.73	95.57	95.23	97.61	100.00
RR	84.40	94.61	**97.01**	95.86	98.40	100.00
PRTY	83.13	93.11	95.20	94.47	96.63	99.80
FCFS	82.25	92.89	95.47	94.54	97.31	100.00

Fonte: Elaborado pelo autor.

De um ponto de vista geral, os algoritmos *fuzzy* se saíram bem quando aplicados a um cenário altamente carregado com recursos limitados, em alguns casos ultrapassando os algoritmos clássicos e em outros, chegando ao mesmo valor ou a um valor muito próximo.

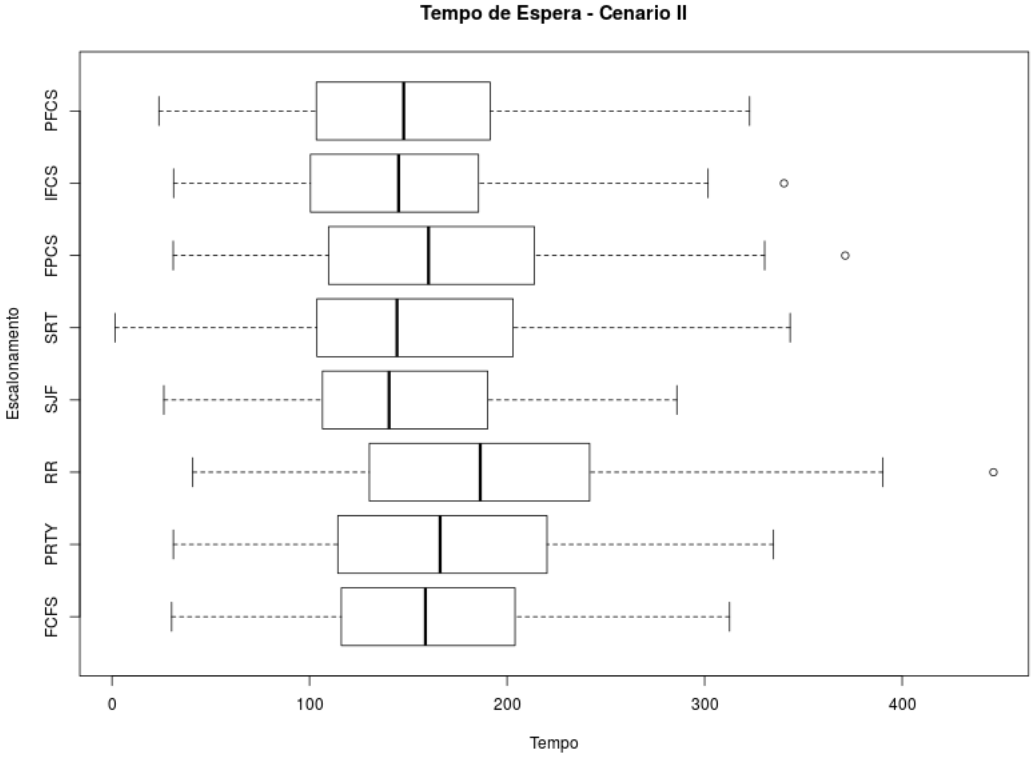
4.2 Cenário II

No segundo cenário experimental, o sistema conta com mais um núcleo, sendo portanto a análise do experimento focada em conhecer o comportamento de cada algoritmo diante da possibilidade de multiprocessamento.

4.2.1 Variável "Tempo de Espera"

O gráfico apresentado na Figura 32 ilustra o tempo médio de espera obtido por cada algoritmo aplicado ao Cenário experimental II.

Figura 32 – Resultado observado para a variável Tempo de Espera obtida pela execução de 100 replicações independentes dos algoritmos de escalonamento PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS sobre a caracterização de processos do Cenário II (dois processadores, alta taxa de chegada de processos, e alta carga de processamento).



Fonte: Elaborado pelo autor.

Ao analisarmos o tempo de espera gerado pelo agendamentos de cada política de escalonamento, é possível notar que os algoritmos PFCS e IFCS se mantiveram muito próximos aos algoritmos SJF e SRT e melhores que o PRTY. A amplitude dos resultados do algoritmo IFCS é menor do que a dos outros algoritmos, apresentando menor variabilidade dos resultados. Os valores obtidos podem ser observados em detalhes na Tabela 13.

Tabela 13 – Resumo com a estatística descritiva dos resultados observados para a variável Tempo de Espera obtidos a partir de 100 replicações independentes do modelo de simulação para os algoritmos PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS quando experimentados sobre a caracterização do Cenário II (dois processadores, alta demanda de processamento, alta taxa de chegada de novos processos), com destaque para os quatro melhores resultados, segundo a mediana.

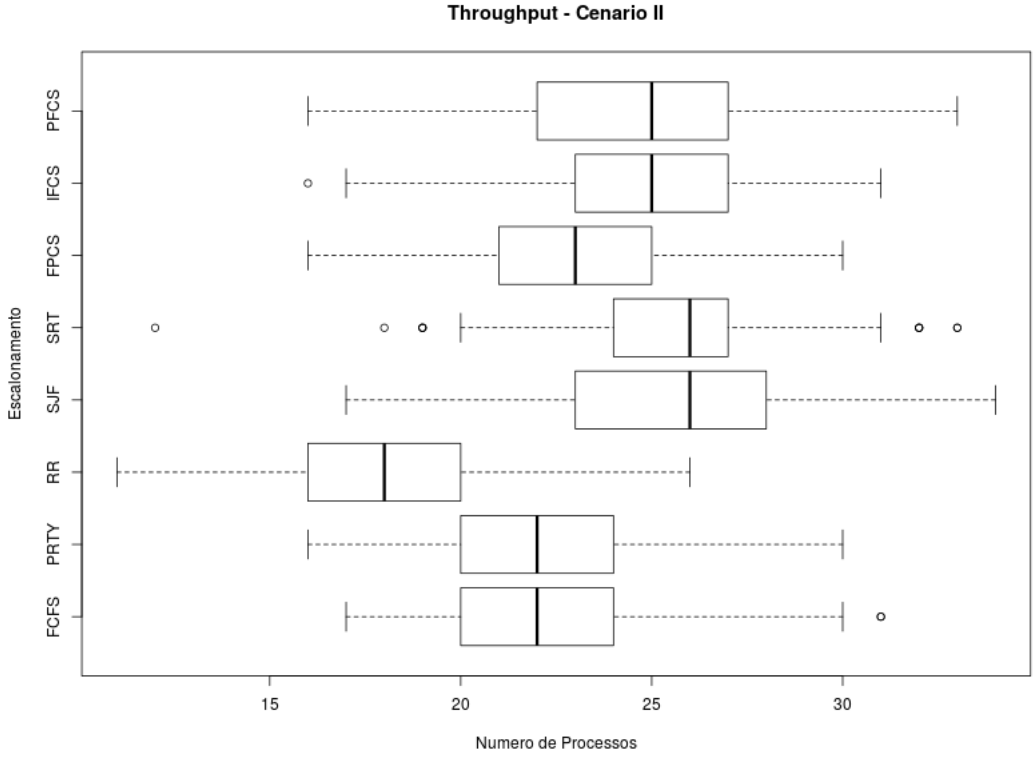
Método	Mínimo	1º Quartil	Mediana	Média	3º Quartil	Máximo
PFCS	23.75	103.44	**147.68**	149.98	191.47	322.77
IFCS	31.17	100.47	**145.08**	148.85	185.55	340.20
FPCS	31.00	109.60	160.30	164.40	213.90	371.2
SRT	1.50	103.70	**144.30**	149.50	203.10	343.3
SJF	26.25	106.56	**140.27**	148.84	190.17	285.95
RR	40.90	130.30	186.40	191.30	241.80	446.2
PRTY	31.07	114.32	166.11	167.85	220.21	334.87
FCFS	30.20	116.10	158.70	162.10	204.00	312.6

Fonte: Elaborado pelo autor.

4.2.2 Variável "*Throughput*"

O gráfico comparativo da vazão de processos que é atribuída à estratégia desempenhada por cada método de escalonamento no Cenário II pode ser observado na Figura 37.

Figura 33 – Resultado observado para a variável *Throughput* obtida pela execução de 100 replicações independentes dos algoritmos de escalonamento PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS sobre a caracterização de processos do Cenário II (dois processadores, alta taxa de chegada de processos, e alta carga de processamento).



Fonte: Elaborado pelo autor.

Ao analisarmos o *throughput* do segundo experimento, é possível perceber que, em relação ao algoritmo de escalonamento PRTY, o três métodos *fuzzy* se sobressaíram. Em relação aos algoritmos SRT e SJF, os algoritmos *fuzzy* ficaram muito próximos, porém, os algoritmos SRT e SJF podem causar inanição, enquanto que os métodos que usam lógica *fuzzy* operam para que isso não aconteça, alterando as prioridades conforme a necessidade. Outra observação é que o valor mínimo de *throughput* dos algoritmos *fuzzy* é superior ao mínimo do algoritmo SRT. Os valores podem ser melhor analisados através da Tabela 14.

Tabela 14 – Resumo com a estatística descritiva dos resultados observados para a variável *Throughput* obtidos a partir de 100 replicações independentes do modelo de simulação para os algoritmos PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS quando experimentados sobre a caracterização do Cenário II (dois processadores, alta demanda de processamento, alta taxa de chegada de novos processos), com destaque para os quatro melhores resultados, segundo a mediana.

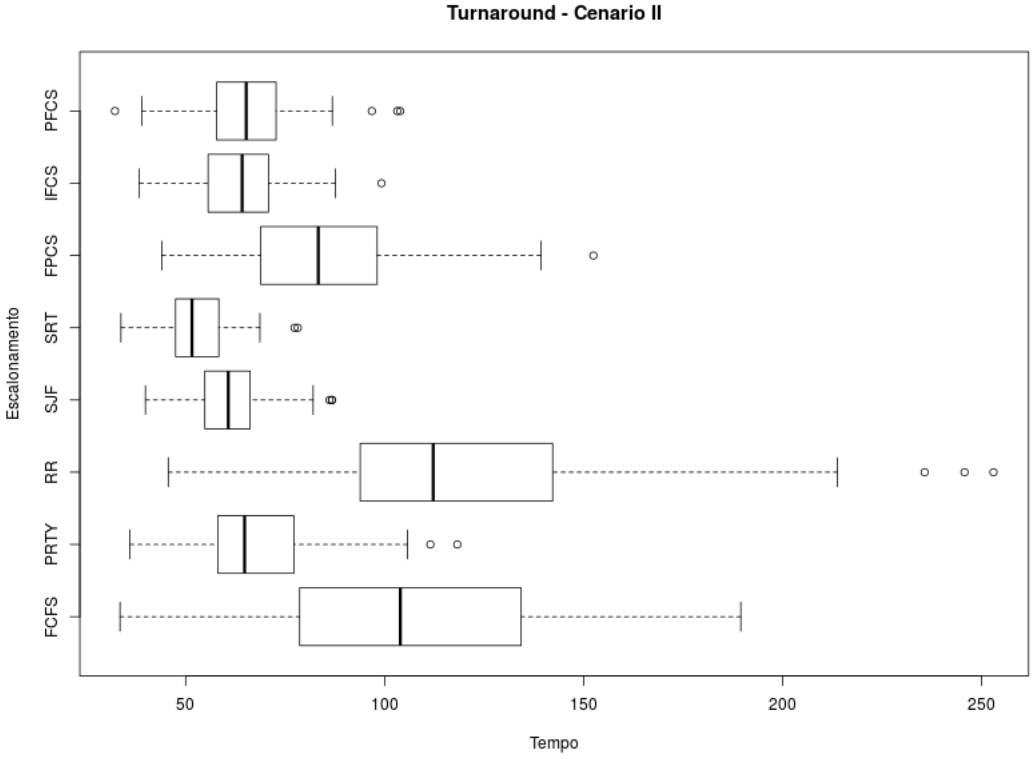
Método	Mínimo	1º Quartil	Mediana	Média	3º Quartil	Máximo
PFCS	16.00	22.00	**25.00**	24.95	27.00	33.00
IFCS	16.00	23.00	**25.00**	24.79	27.00	31.00
FPCS	16.00	21.00	23.00	23.23	25.00	30.00
SRT	12.00	24.00	**26.00**	25.58	27.00	33.00
SJF	17.00	23.00	**26.00**	25.98	28.00	34.00
RR	11.00	16.00	18.00	18.00	20.00	26.00
PRTY	16.00	20.00	22.00	22.33	24.00	30.00
FCFS	17.00	20.00	22.00	22.38	24.00	31.00

Neste ponto é possível observar que o algoritmo RR começa a exibir desempenho mais próximo aos demais métodos. Isso se deve ao fato de que, com mais recursos, é possível efetuar mais rapidamente o chaveamento entre os processos da fila de prontos, aumentando o desempenho do sistema computacional como um todo.

4.2.3 Variável "*Turnaround*"

O gráfico ilustrado na Figura 34 exibe o tempo de *turnaround* obtido por cada método de escalonamento ao ser simulado no cenário II.

Figura 34 – Resultado observado para a variável *Turnaround* obtida pela execução de 100 replicações independentes dos algoritmos de escalonamento PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS sobre a caracterização de processos do Cenário II (dois processadores, alta taxa de chegada de processos, e alta carga de processamento).



Fonte: Elaborado pelo autor.

Analisando a Figura 34, o tempo de *turnaround* do segundo experimento sugere que os algoritmos *fuzzy* PFCS e IFCS ficaram muito próximos do algoritmo PRTY. Novamente os algoritmos *fuzzy* ficaram em pequena desvantagem quando comparados aos algoritmos SJF e SRT, possivelmente pela alta demanda de processos curtos. A Tabela 15 apresenta em detalhes os resultados obtidos por cada algoritmo.

Tabela 15 – Resumo com a estatística descritiva dos resultados observados para a variável *Turnaround* obtidos a partir de 100 replicações independentes do modelo de simulação para os algoritmos PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS quando experimentados sobre a caracterização do Cenário II (dois processadores, alta demanda de processamento, alta taxa de chegada de novos processos), com destaque para os quatro melhores resultados, segundo a mediana.

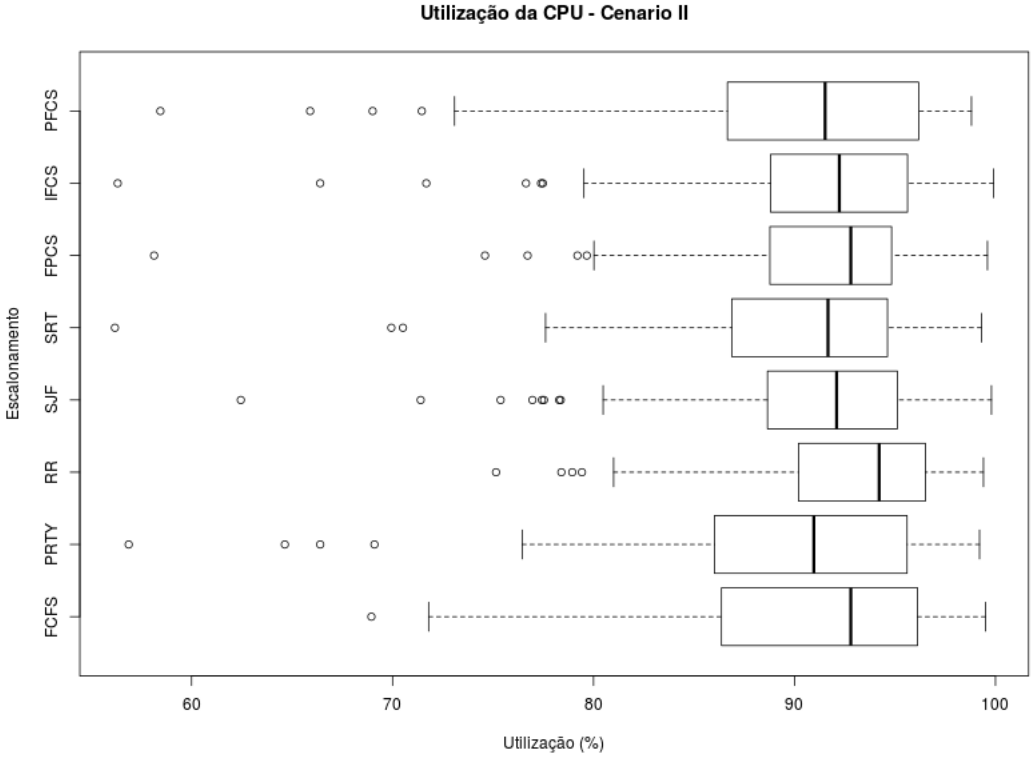
Método	Mínimo	1º Quartil	Mediana	Média	3º Quartil	Máximo
PFCS	32.24	57.78	65.22	65.39	72.72	103.92
IFCS	38.38	55.65	**64.19**	63.63	70.82	99.21
FPCS	44.00	68.83	83.33	85.39	98.13	152.46
SRT	33.74	47.47	**51.59**	52.86	58.34	78.19
SJF	39.89	54.82	**60.71**	61.09	66.20	86.79
RR	45.63	93.87	112.21	122.42	142.23	253.00
PRTY	36.00	58.10	**64.81**	68.59	77.19	118.26
FCFS	33.50	78.65	103.95	106.89	134.32	189.50

Fonte: Elaborado pelo autor.

4.2.4 Variável "Utilização da CPU"

Na Figura 35, é possível efetuar uma análise de como cada algoritmo conseguiu se aproveitar efetivamente o tempo da CPU considerando que o sistema computacional descrito opera sobre processos caracterizados pelo Cenário II.

Figura 35 – Resultado observado para a variável Utilização da CPU obtida pela execução de 100 replicações independentes dos algoritmos de escalonamento PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS sobre a caracterização de processos do Cenário II (dois processadores, alta taxa de chegada de processos, e alta carga de processamento).



Fonte: Elaborado pelo autor.

É notável que todos os algoritmos ficaram extremamente próximos de atingir os 100% de utilização. Todos os algoritmos mantiveram a mediana acima de 90% de utilização, ou seja, em mais da metade das execuções, o valor de utilização foi acima de 90%. Desta vez, o algoritmo que se saiu melhor foi o RR, mantendo a CPU tão ocupada quanto possível. Devido ao tamanho da sua fila de prontos, na maioria dos casos sempre há um processo em estado pronto aguardando execução. Mais detalhes sobre este indicador de desempenho podem ser observados na Tabela 16.

Tabela 16 – Resumo com a estatística descritiva dos resultados observados para a variável Utilização da CPU obtidos a partir de 100 replicações independentes do modelo de simulação para os algoritmos PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS quando experimentados sobre a caracterização do Cenário II (dois processadores, alta demanda de processamento, alta taxa de chegada de novos processos), com destaque para os quatro melhores resultados, segundo a mediana.

Método	Mínimo	1º Quartil	Mediana	Média	3º Quartil	Máximo
PFCS	58.45	86.66	91.52	89.81	96.18	98.80
IFCS	56.32	88.81	**92.23**	90.81	95.62	99.90
FPCS	58.13	88.78	**92.80**	91.33	94.83	99.60
SRT	56.19	86.88	91.67	90.04	94.63	99.30
SJF	62.45	88.66	92.09	90.74	95.12	99.80
RR	75.15	90.20	**94.21**	92.50	96.51	99.40
PRTY	56.87	86.01	90.96	89.64	95.60	99.21
FCFS	68.95	86.37	**92.80**	90.70	96.12	99.50

Fonte: Elaborado pelo autor.

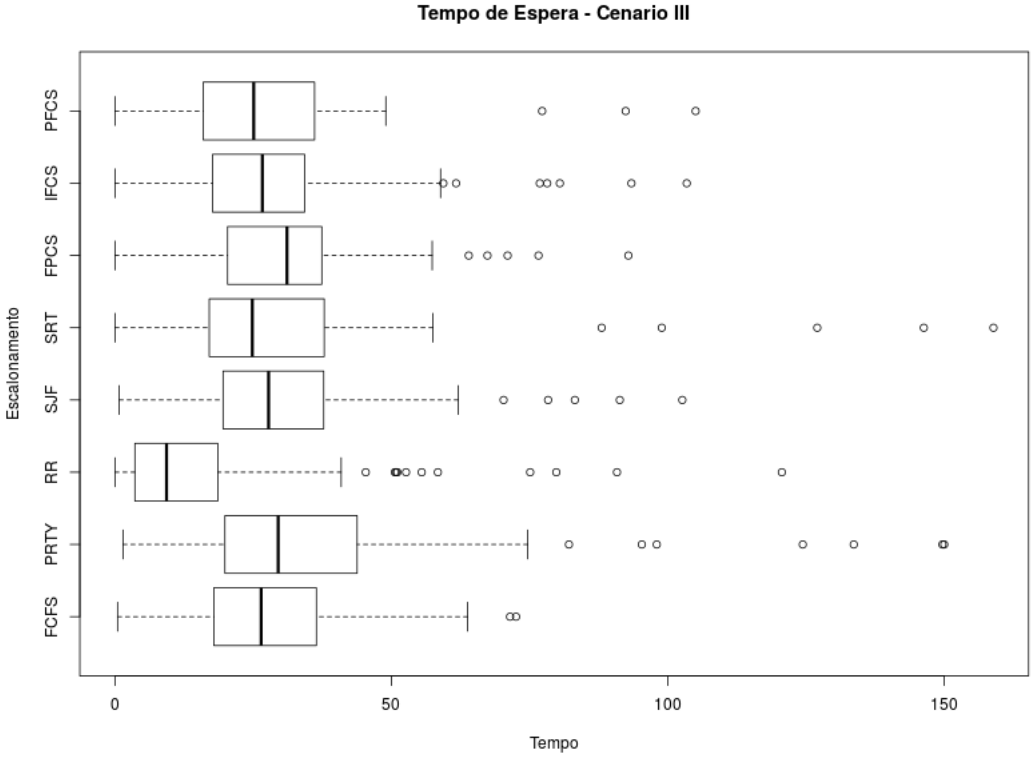
4.3 Cenário III

O cenário experimental III ilustra uma situação com maior quantidade de recursos disponíveis quando comparado aos cenários anteriormente experimentados, mais especificamente, com quatro núcleos de processamento, porém, sem atividade I/O envolvida.

4.3.1 Variável "Tempo de Espera"

O tempo de espera experimentado por cada algoritmo de escalonamento quando executando processos conforme caracterizado no Cenário III fé apresentado a seguir, no *boxplot* exibido pela Figura 36.

Figura 36 – Resultado observado para a variável Tempo de Espera obtida pela execução de 100 replicações independentes dos algoritmos de escalonamento PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS sobre a caracterização de processos do Cenário III (quatro processadores, alta taxa de chegada de processos, e alta carga de processamento).



Fonte: Elaborado pelo autor.

Analisando o tempo de espera alcançado pelos algoritmos, é possível confirmar o ganho do algoritmo RR em relação a quantidade de recursos, tendo menos da metade do tempo de espera dos outros algoritmos de escalonamento analisados. A maior amplitude de resultados foi obtida pelo algoritmo SRT, alcançando os maiores valores para tempo de espera. Enquanto isso, os algoritmos *fuzzy* se mantiveram superiores ao PRTY, com o FPCS tendo o melhor valor de mediana do experimento. Os valores detalhados são mostrados na Tabela 17.

Tabela 17 – Resumo com a estatística descritiva dos resultados observados para a variável Tempo de Espera a partir de 100 replicações independentes do modelo de simulação para os algoritmos PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS quando experimentados sobre a caracterização do Cenário III (quatro processadores, alta demanda de processamento, alta taxa de chegada de novos processos), com destaque para os quatro melhores resultados, segundo a mediana.

Método	Mínimo	1º Quartil	Mediana	Média	3º Quartil	Máximo
PFCS	0.00	15.96	**25.10**	27.06	36.11	105.05
IFCS	0.00	17.65	26.67	28.57	34.32	103.43
FPCS	0.00	20.38	31.12	30.28	37.42	92.88
SRT	0.00	17.06	**24.85**	30.89	37.86	158.92
SJF	0.76	19.55	27.79	30.33	37.76	102.64
RR	0.00	3.60	**9.33**	16.42	18.59	120.64
PRTY	1.44	19.89	29.52	37.26	43.81	150.01
FCFS	0.47	17.87	**26.44**	28.21	36.44	72.561

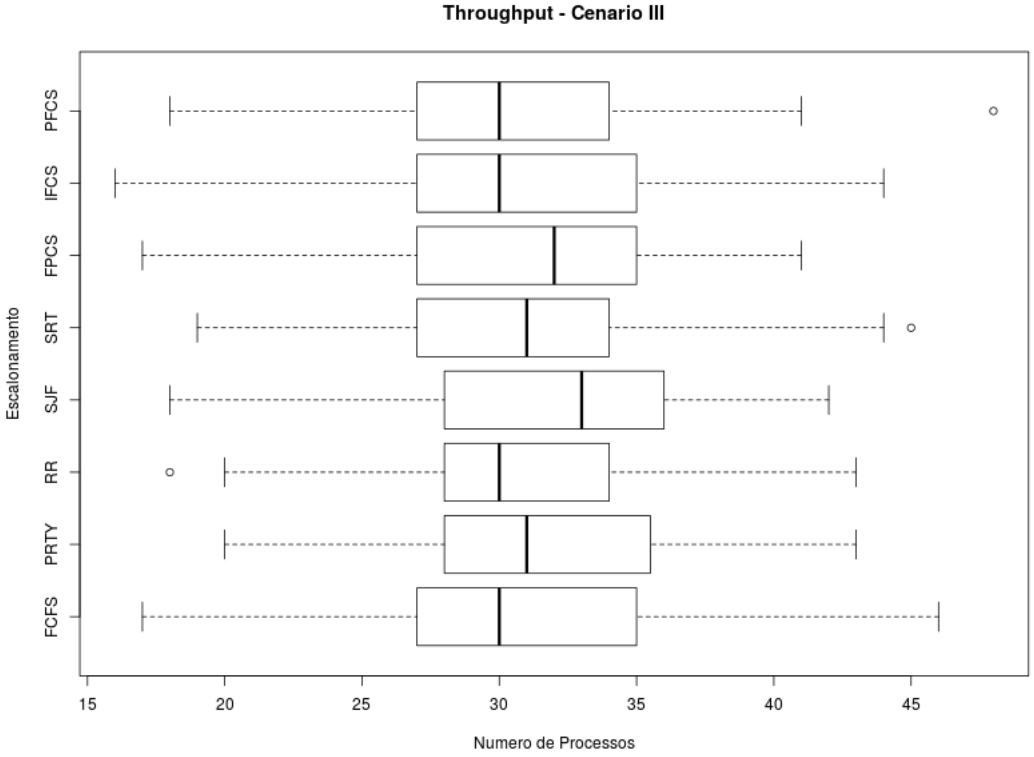
Fonte: Elaborado pelo autor.

Através da observação da Tabela 17, é perceptível que todos os algoritmos ou tiveram zero ou se aproximaram de zero no menor tempo de espera observado, dentre as 100 replicações independentes. Isso acontece devido a alta disponibilidade de recursos, que faz com que em alguns casos o processo chegue ao sistema e tenha sua demanda imediatamente atendida.

4.3.2 Variável "*Throughput*"

A Figura 37 apresenta os resultados observados para a variável Throughput coletados pela replicação do modelo de simulação por 100 vezes, de maneira independente. Ela retrata a variabilidade dos resultados observados, e permite-nos inferir sugestões acerca do desempenho dos métodos quando analisando a vazão em termos de quantos processos foram efetivamente finalizados durante o horizonte de simulação.

Figura 37 – Resultado observado para a variável *Throughput* obtida pela execução de 100 replicações independentes dos algoritmos de escalonamento PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS sobre a caracterização de processos do Cenário III (quatro processadores, alta taxa de chegada de processos, e alta carga de processamento).



Fonte: Elaborado pelo autor.

Analisando a Figura 37, que exibe a vazão de processos para cada algoritmo podemos afirmar que todos os algoritmos conseguiram efetuar os agendamentos de forma que no mínimo 15 processos fossem finalizados e, na maior parte dos casos, foram finalizados entre 25 e 35 processos. O algoritmo clássico SJF e o algoritmo *fuzzy* FPCS tiveram mediana 32 e 33, respectivamente, o que significa que em 50% dos casos houveram no mínimo 32 processos finalizados para o SJF e 33 para o FPCS fazendo destes os algoritmos com maior vazão de processos neste cenário. É possível visualizar estas informações na Tabela 18.

Tabela 18 – Resumo com a estatística descritiva dos resultados observados para a variável *Throughput* a partir de 100 replicações independentes do modelo de simulação para os algoritmos PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS quando experimentados sobre a caracterização do Cenário III (quatro processadores, alta demanda de processamento, alta taxa de chegada de novos processos), com destaque para os quatro melhores resultados, segundo a mediana.

Método	Mínimo	1º Quartil	Mediana	Média	3º Quartil	Máximo
PFCS	18.00	27.00	30.00	30.63	34.00	48.00
IFCS	16.00	27.00	30.00	30.70	35.00	44.00
FPCS	17.00	27.00	**32.00**	31.27	35.00	41.00
SRT	19.00	27.00	**31.00**	31.13	34.00	45.00
SJF	18.00	28.00	**33.00**	31.75	36.00	42.00
RR	18.00	28.00	30.00	30.30	34.00	43.00
PRTY	20.00	28.00	**31.00**	31.52	35.50	43.00
FCFS	17.00	27.00	30.00	31.20	35.00	46.00

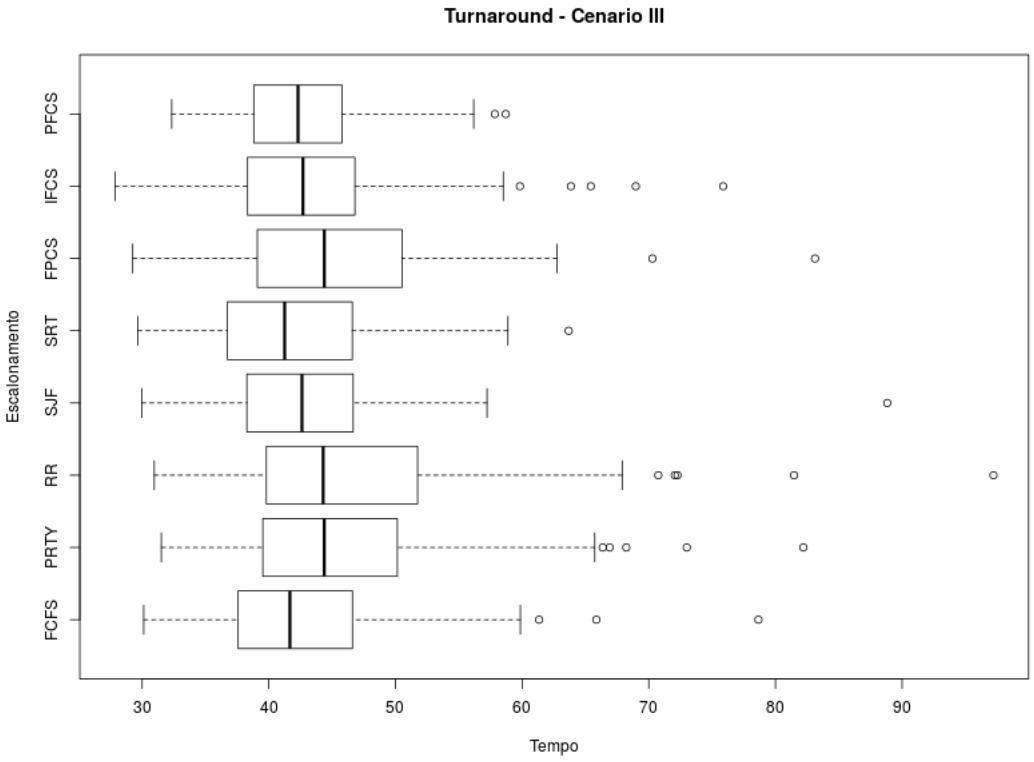
Fonte: Elaborado pelo autor.

Observando a Tabela 18 é possível perceber que o algoritmo que conseguiu a maior vazão de processos foi o PFCS, com uma mediana de 30 e um máximo de 48 processos.

4.3.3 Variável "*Turnaround*"

A Figura 38 ilustra os tempos de *turnaround* observado quando os algoritmos de escalonamento propostos neste trabalho foram aplicados à configuração de caracterização do sistema computacional definido no Cenário III.

Figura 38 – Resultado observado para a variável *Turnaround* obtida pela execução de 100 replicações independentes dos algoritmos de escalonamento PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS sobre a caracterização de processos do Cenário III (quatro processadores, alta taxa de chegada de processos, e alta carga de processamento).



Fonte: Elaborado pelo autor.

Em relação ao tempo de turnaround, o melhor desempenho foi obtido pelo algoritmo SRT, porém todos os algoritmos obtiveram um desempenho semelhante, com mediana entre 40 e 45 milissegundos. O algoritmo *fuzzy* PFCS obteve o melhor 3º quartil, com tempo de espera em 75% dos casos menor do que 45.78 milissegundos e também obteve o melhor valor máximo, 58,71 milissegundos. Os valores detalhados podem ser observados na Tabela 19.

Tabela 19 – Resumo com a estatística descritiva dos resultados observados para a variável *Turnaround* a partir de 100 replicações independentes do modelo de simulação para os algoritmos PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS quando experimentados sobre a caracterização do Cenário III (quatro processadores, alta demanda de processamento, alta taxa de chegada de novos processos), com destaque para os quatro melhores resultados, segundo a mediana.

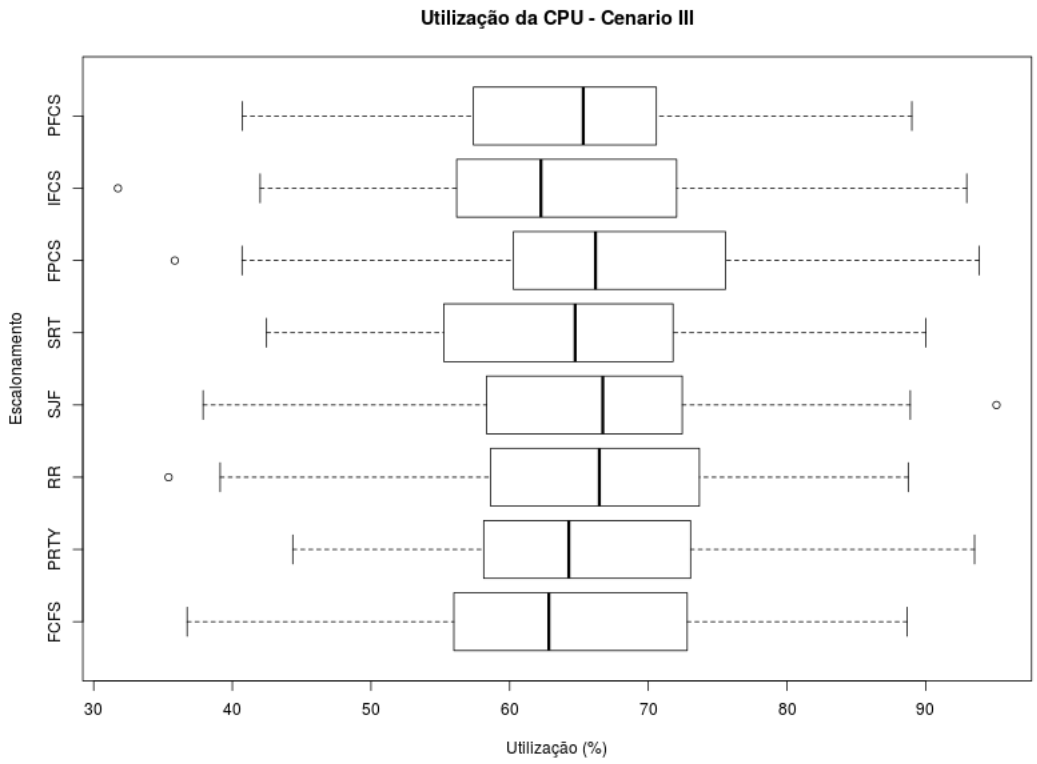
Método	Mínimo	1º Quartil	Mediana	Média	3º Quartil	Máximo
PFCS	32.33	38.81	**42.31**	42.60	45.78	58.71
IFCS	27.85	38.32	42.70	43.98	46.80	75.87
FPCS	29.26	39.09	44.38	45.75	50.52	83.12
SRT	29.67	36.73	**41.25**	42.12	46.59	63.68
SJF	29.97	38.26	**42.62**	42.88	46.66	88.83
RR	30.96	39.81	44.29	46.68	51.75	97.21
PRTY	31.51	39.53	44.38	45.95	50.15	82.21
FCFS	30.12	37.58	**41.67**	43.27	46.60	78.66

Fonte: Elaborado pelo autor.

4.3.4 Variável "Utilização da CPU"

O desempenho dos algoritmos de escalonamento quando submetidos às condições especificadas no Cenário III, sob a perspectiva da utilização efetiva da CPU, obtiveram desempenho tal como reportado no boxplot da Figura 39 apresentada a seguir.

Figura 39 – Resultado observado para a variável Utilização da CPU obtida pela execução de 100 replicações independentes dos algoritmos de escalonamento PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS sobre a caracterização de processos do Cenário III (quatro processadores, alta taxa de chegada de processos, e alta carga de processamento).



Fonte: Elaborado pelo autor.

Neste experimento, a utilização da CPU foi reduzida, o que já era esperado, visto que neste cenário houve uma disponibilidade maior de núcleos para dividir a carga de processamento dos processos que chegam no sistema. Ao aumentarmos a quantidade de núcleos, é provável que haja ociosidade na CPU se a taxa de chegada de processos for menor do que a capacidade de processar os CPU bursts existentes. Em outras palavras, se enxergarmos o sistema computacional simulado como um sistema produtor-consumidor, a taxa de consumo está acima da taxa de produção de demandas. Ainda assim, em todos os algoritmos, a utilização da CPU ultrapassou os 60% em metade das execuções. Os algoritmos FPCS e SJF tiveram medianas muito próximas, porém, ao desconsiderarmos os valores atípicos, o algoritmo FPCS leva vantagem no máximo alcançado e no terceiro quartil. Mais detalhes podem ser observados na Tabela 20.

Tabela 20 – Resumo com a estatística descritiva dos resultados observados para a variável Utilização da CPU a partir de 100 replicações independentes do modelo de simulação para os algoritmos PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS quando experimentados sobre a caracterização do Cenário III (quatro processadores, alta demanda de processamento, alta taxa de chegada de novos processos), com destaque para os quatro melhores resultados, segundo a mediana.

Método	Mínimo	1º Quartil	Mediana	Média	3º Quartil	Máximo
PFCS	40.73	57.38	**65.31**	64.40	70.58	89.00
IFCS	31.75	56.18	62.26	63.66	72.04	92.96
FPCS	35.85	60.27	**66.19**	67.03	75.56	93.85
SRT	42.47	55.25	64.72	64.05	71.79	90.02
SJF	37.89	58.33	**66.72**	65.42	72.45	95.11
RR	35.40	58.62	**66.47**	65.63	73.68	88.75
PRTY	44.36	58.13	64.26	65.76	73.05	93.53
FCFS	36.74	55.98	62.82	64.05	72.81	88.67

Fonte: Elaborado pelo autor.

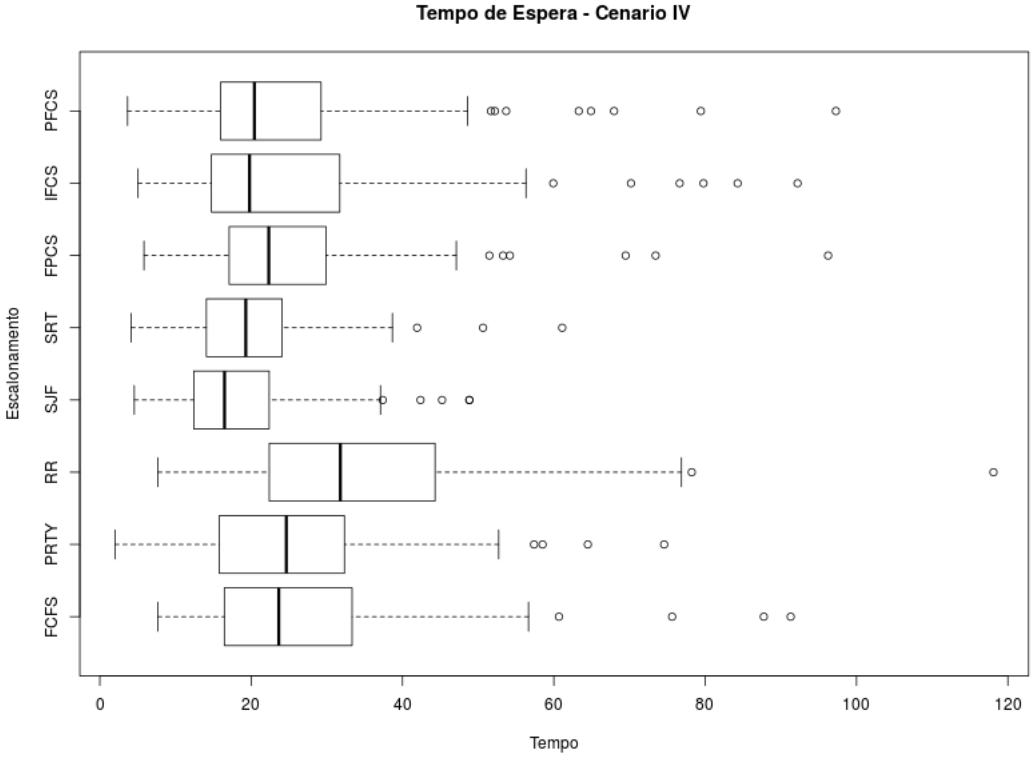
4.4 Cenário IV

No quarto cenário experimental foram incluídas na simulação as operações I/O pela primeira vez, isto é, há existência de situações que intercalam CPU *bursts* com I/O *bursts*, sendo portanto importante analisar os resultados considerando interrupções na execução de cada processo.. Este cenário conta com um único núcleo na CPU e um único dispositivo I/O para tratar os pedidos de leitura e escrita externos à CPU. O objetivo é observar o comportamento dos algoritmos de escalonamento quando trabalham com processos que possuam mais de um pico de CPU e, além disso, o envolvimento de operações I/O e seu impacto no desempenho do sistema como um todo.

4.4.1 Variável "Tempo de Espera"

Nesta seção apresentaremos inicialmente qual foi o desempenho dos algoritmos PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS no cenário com um núcleo e um dispositivo de I/O em relação ao Tempo de Espera experimentado pelos processos ao longo das 100 replicações independentes do modelo de simulação. Os resultados observados são apresentados a seguir, no *boxplot* ilustrado na Figura 40.

Figura 40 – Resultado observado para a variável Tempo de Espera obtida pela execução de 100 replicações independentes dos algoritmos de escalonamento PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS sobre a caracterização de processos do Cenário IV (alta taxa de chegada de processos, processador único e um dispositivo de I/O).



Fonte: Elaborado pelo autor.

Ao analisar o gráfico, é perceptível que o tempo de espera é visivelmente menor quando comparado aos cenários apresentados anteriormente, já que a fila de prontos sofre efeito dos processos que estão realizando operações de I/O. Os algoritmos *fuzzy* se sobressaem neste indicador quando comparados ao PRTY e ficam bem próximos aos algoritmos de escalonamento SRT e SJF. Vale ressaltar que existem alguns casos isolados que ultrapassam os 80 milissegundos de tempo de espera. É possível analisar a proximidade dos resultados pela Tabela 21.

Tabela 21 – Resumo com a estatística descritiva dos resultados observados para a variável Tempo de Espera a partir de 100 replicações independentes do modelo de simulação para os algoritmos PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS quando experimentados sobre a caracterização do Cenário IV (alta taxa de chegada de processos, processador único e um dispositivo de I/O), com destaque para os quatro melhores resultados, segundo a mediana.

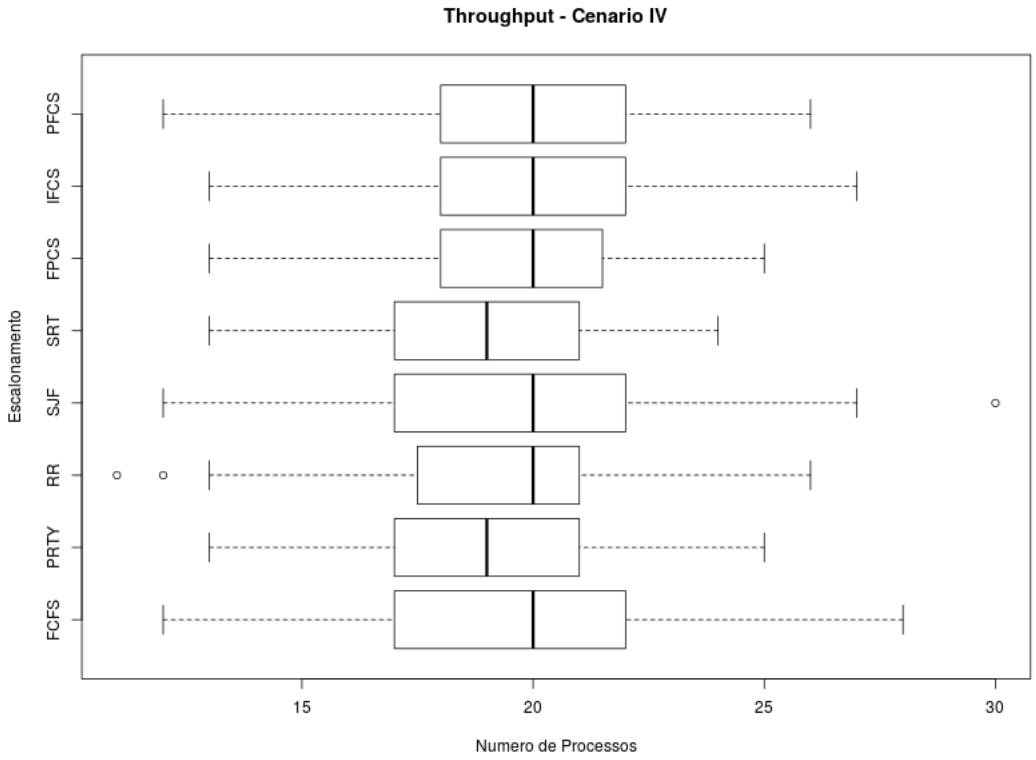
Método	Mínimo	1º Quartil	Mediana	Média	3º Quartil	Máximo
PFCS	3.66	15.95	**20.42**	25.10	29.24	97.27
IFCS	05.05	14.74	**19.78**	25.34	31.71	92.21
FPCS	5.83	17.09	22.33	25.82	29.89	96.24
SRT	4.13	14.08	**19.29**	20.64	24.09	61.10
SJF	4.55	12.41	**16.48**	18.71	22.38	48.83
RR	7.67	22.39	31.78	35.38	44.30	118.10
PRTY	2.00	15.79	24.65	25.88	32.33	74.58
FCFS	7.65	16.45	23.65	27.11	33.33	91.30

Fonte: Elaborado pelo autor.

4.4.2 Variável "*Throughput*"

A Figura 41 exibe o gráfico boxplot com os resultados observados nas 100 replicações do modelo de simulação que implementa as estratégias de escalonamentos avaliadas neste projeto. Por meio dele é possível ter ciência de como os algoritmos se comportaram no Cenário IV.

Figura 41 – Resultado observado para a variável *Throughput* obtida pela execução de 100 replicações independentes dos algoritmos de escalonamento PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS sobre a caracterização de processos do Cenário IV (alta taxa de chegada de processos, processador único e um dispositivo de I/O).



Fonte: Elaborado pelo autor.

Ao compararmos o *throughput* de cada algoritmo através do gráfico exibido na Figura 41, é possível perceber que os algoritmos SRT e SJF ficam em desvantagem quando comparados aos algoritmos *fuzzy*. O SRT possui mediana inferior e o SJF, mesmo possuindo a mesma mediana, possui maior dispersão dos resultados. O algoritmo PRTY também possui mediana inferior ao comparar com o FPCS, IFCS e PFCS, mostrando que os algoritmos *fuzzy* cumprem a proposta para qual foram desenvolvidos. Os dados podem ser observados em detalhes na Tabela 22.

Tabela 22 – Resumo com a estatística descritiva dos resultados observados para a variável *Throughput* a partir de 100 replicações independentes do modelo de simulação para os algoritmos PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS quando experimentados sobre a caracterização do Cenário IV (alta taxa de chegada de processos, processador único e um dispositivo de I/O), com destaque para os melhores resultados, segundo a mediana.

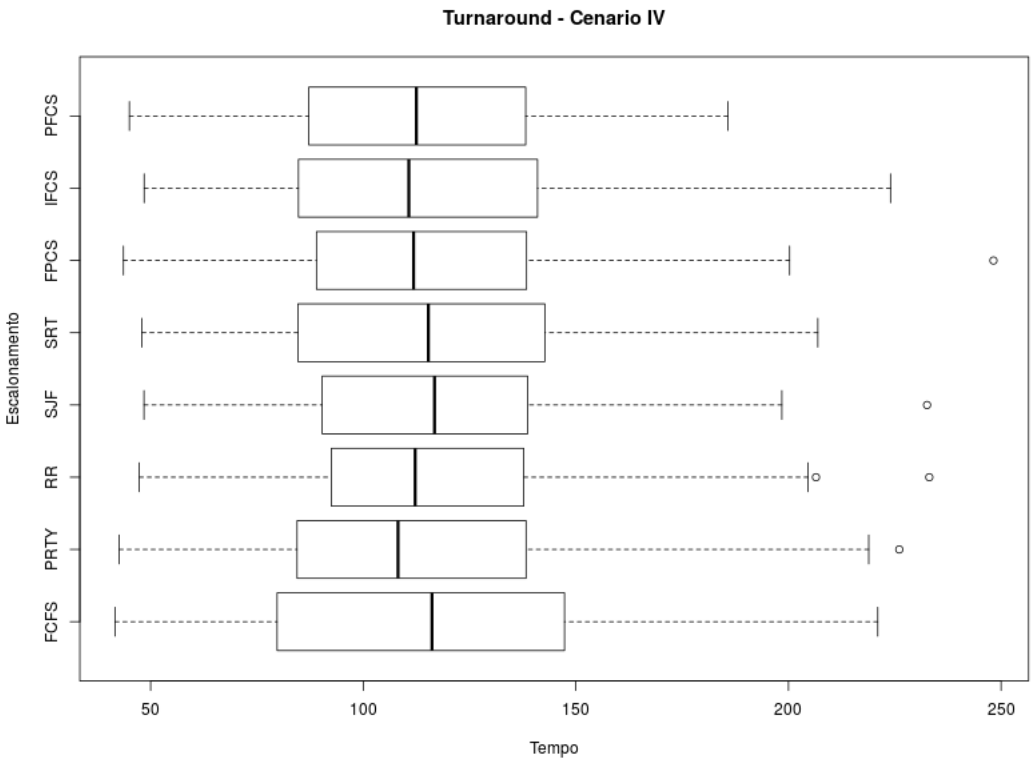
Método	Mínimo	1º Quartil	Mediana	Média	3º Quartil	Máximo
PFCS	12.00	18.00	**20.00**	19.70	22.00	26.00
IFCS	13.00	18.00	**20.00**	19.76	22.00	27.00
FPCS	13.00	18.00	**20.00**	19.56	21.50	25.00
SRT	13.00	17.00	19.00	18.91	21.00	24.00
SJF	12.00	17.00	**20.00**	19.68	22.00	30.00
RR	11.00	17.50	**20.00**	19.30	21.00	26.00
PRTY	13.00	17.00	19.00	19.07	21.00	25.00
FCFS	12.00	17.00	**20.00**	19.60	22.00	28.00

Fonte: Elaborado pelo autor.

4.4.3 Variável "*Turnaround*"

O gráfico apresentado na Figura 42 exibe o tempo de *turnaround* experimentado por cada algoritmo de escalonamento no Cenário IV, com apenas um núcleo e um dispositivo de I/O.

Figura 42 – Resultado observado para a variável *Turnaround* obtida pela execução de 100 replicações independentes dos algoritmos de escalonamento PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS sobre a caracterização de processos do Cenário IV (alta taxa de chegada de processos, processador único e um dispositivo de I/O).



Fonte: Elaborado pelo autor.

Ao analisarmos o tempo de *turnaround* através do gráfico exposto na Figura x é possível notar que o tempo médio de *turnaround* foi muito semelhante entre os algoritmos. O algoritmo PFCS obteve o menor valor máximo, porém, através dele foi obtida a menor dispersão de valores. O algoritmo IFCS obteve o maior valor mínimo, significando que, no pior dos casos, o IFCS saiu à frente dos outros algoritmos. Os detalhes do experimento podem ser observados na Tabela x abaixo.

Tabela 23 – Resumo com a estatística descritiva dos resultados observados para a variável *Turnaround* a partir de 100 replicações independentes do modelo de simulação para os algoritmos PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS quando experimentados sobre a caracterização do Cenário IV (alta taxa de chegada de processos, processador único e um dispositivo de I/O), com destaque para os quatro melhores resultados, segundo a mediana.

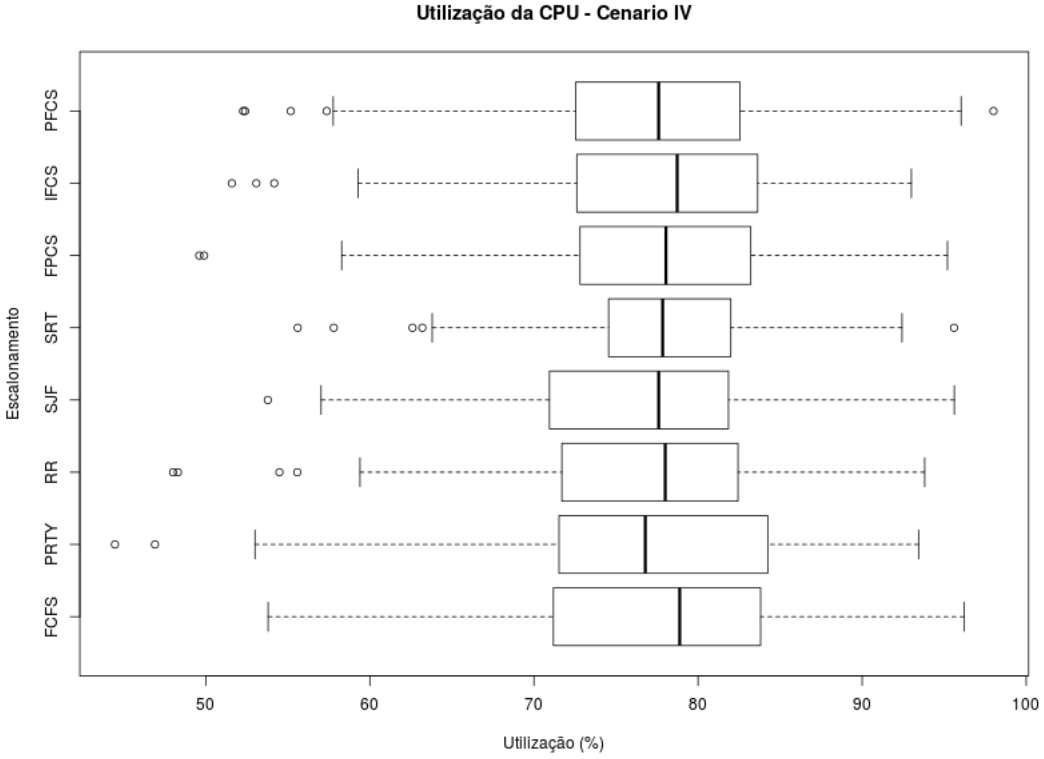
Média	Mínimo	1º Quartil	Mediana	Média	3º Quartil	Máximo
PFCS	45.00	87.18	112.50	113.67	138.21	185.77
IFCS	48.48	84.77	**110.73**	115.49	140.99	224.00
FPCS	43.61	89.01	**111.84**	114.16	138.41	248.21
SRT	47.94	84.65	115.32	116.91	142.70	206.90
SJF	48.43	90.34	116.76	115.60	138.65	232.59
RR	47.33	92.50	**112.21**	117.11	137.74	233.12
PRTY	42.60	84.44	**108.18**	114.46	138.29	226.06
FCFS	41.61	79.75	116.17	116.39	147.33	221.00

Fonte: Elaborado pelo autor.

4.4.4 Variável "Utilização da CPU"

Em termos de desempenho, quando analisamos o indicador de utilização da CPU, considera-se o algoritmo que maximizou seu uso como sendo o mais adequado para esta métrica. Pelo gráfico da Figura 43 podemos identificar, dentre os algoritmos experimentados, qual melhor se adaptou à caracterização proposta pelo cenário IV.

Figura 43 – Resultado observado para a variável Utilização da CPU obtida pela execução de 100 replicações independentes dos algoritmos de escalonamento PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS sobre a caracterização de processos do Cenário IV (alta taxa de chegada de processos, processador único e um dispositivo de I/O).



Fonte: Elaborado pelo autor.

Efetuando uma análise geral, é possível dizer que os algoritmos fizeram uma boa utilização da CPU, mantendo a utilização, na maioria dos casos, próxima a 80%. Analisando o desempenho de cada algoritmo é visível que o algoritmo que chegou mais próximo de 100% de uso da CPU com um único núcleo foi o PFCS, alcançando em pelo menos uma replicação independente do modelo o valor de 98%, como é exibido na Tabela x. O algoritmo com maior mediana e menor ocorrência de valores anômalos foi o FCFS. O algoritmo SRT foi o de menor variabilidade dos resultados.

Tabela 24 – Resumo com a estatística descritiva dos resultados observados para a variável Utilização da CPU a partir de 100 replicações independentes do modelo de simulação para os algoritmos PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS quando experimentados sobre a caracterização do Cenário IV (alta taxa de chegada de processos, processador único e um dispositivo de I/O), com destaque para os quatro melhores resultados, segundo a mediana.

Método	Mínimo	1º Quartil	Mediana	Média	3º Quartil	Máximo
PFCS	52.28	72.54	77.60	76.75	82.56	98.00
IFCS	51.59	72.61	**78.73**	77.15	83.63	93.00
FPCS	49.61	72.81	**78.04**	77.28	83.22	95.20
SRT	55.60	74.55	77.84	77.61	82.00	95.60
SJF	53.78	70.95	77.60	76.60	81.85	95.62
RR	48.01	71.70	**78.00**	76.57	82.44	93.81
PRTY	44.47	71.53	76.79	76.84	84.25	93.44
FCFS	53.80	71.19	**78.88**	77.31	83.81	96.21

Fonte: Elaborado pelo autor.

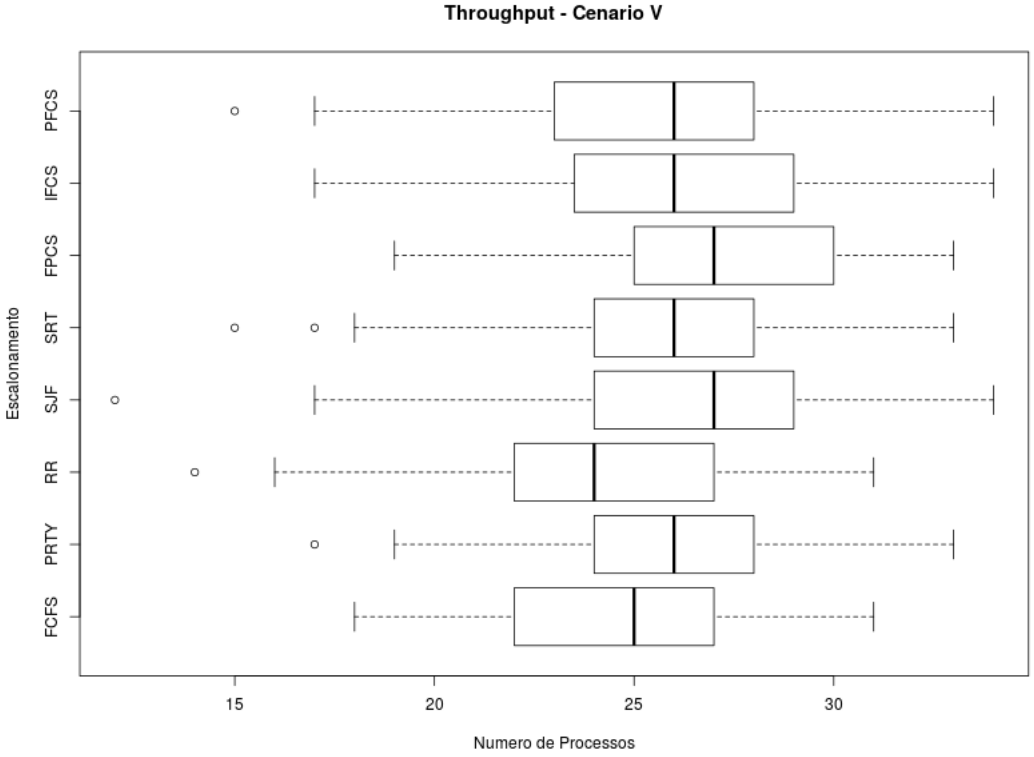
4.5 Cenário V

No cenário experimental V, foi mantido um único núcleo, porém o número de dispositivos de I/O foi elevado para 3. Essa experimentação foi executada com o objetivo de observar o comportamento dos algoritmos quando quando o ambiente sofre (i) alto índice de chegada de processos I/O bound ao sistema e (ii) alto índice de processos encerrando operações de I/O e ingressando a fila de prontos.

4.5.1 Variável "Tempo de Espera"

O tempo de espera desempenhado por cada algoritmo quando aplicado na simulação do cenário experimental V pode ser observado no gráfico apresentado pela Figura 44.

Figura 44 – Resultado observado para a variável Tempo de Espera obtida pela execução de 100 replicações independentes dos algoritmos de escalonamento PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS sobre a caracterização de processos do Cenário V (alta taxa de chegada de processos, único processador e três dispositivos de I/O).



Fonte: Elaborado pelo autor.

Analisando o gráfico, é possível afirmar que os algoritmos *fuzzy* perderam em desempenho se comparados aos algoritmos SRT e SJF, porém ficaram a frente do algoritmo de escalonamento por prioridades (PRTY) mais uma vez desempenhando o papel para qual foram projetados. Maiores detalhes podem ser observados na Tabela 25.

Tabela 25 – Resumo com a estatística descritiva dos resultados observados para a variável Tempo de Espera a partir de 100 replicações independentes do modelo de simulação para os algoritmos PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS quando experimentados sobre a caracterização do Cenário V (alta taxa de chegada de processos, único processador e três dispositivos de I/O), com destaque para os quatro melhores resultados, segundo a mediana.

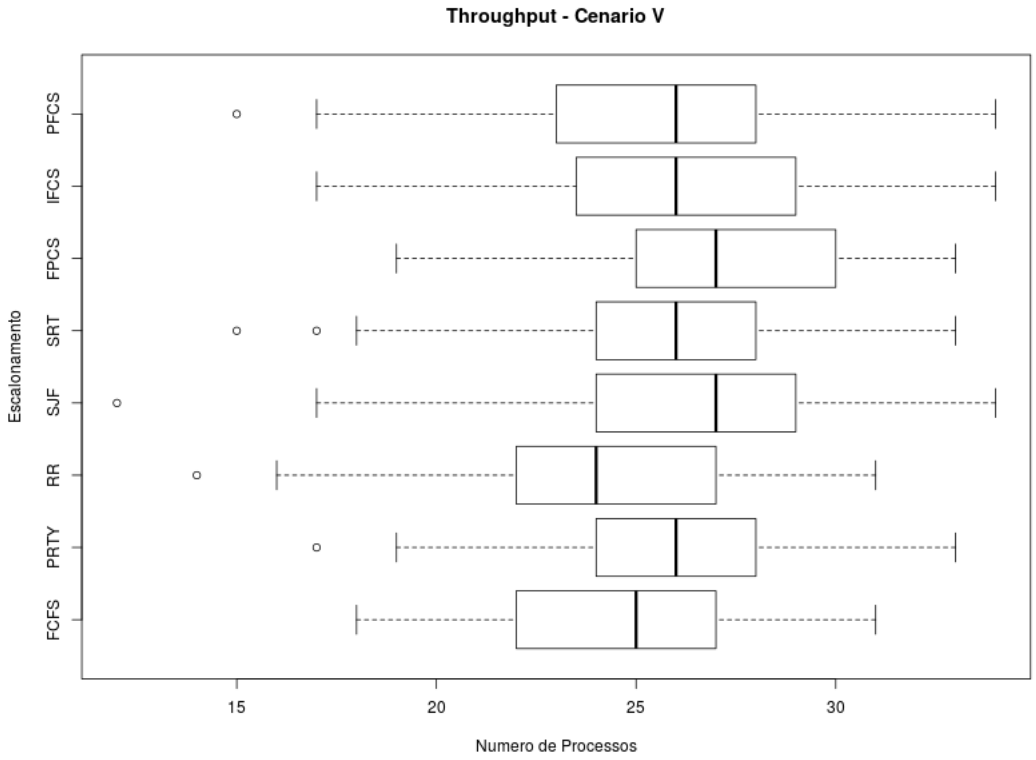
Método	Mínimo	1º Quartil	Mediana	Média	3º Quartil	Máximo
PFCS	6.47	39.53	79.23	89.87	126.97	265.36
IFCS	6.87	33.76	**69.03**	82.85	128.51	239.637
FPCS	11.04	42.91	78.20	85.40	108.45	258.80
SRT	6.28	32.45	**64.59**	68.33	90.83	242.90
SJF	12.58	30.80	**55.54**	61.68	83.85	178.56
RR	13.81	56.61	104.19	109.14	150.40	319.57
PRTY	5.50	43.13	84.36	95.87	132.55	313.80
FCFS	5.63	38.75	**75.07**	91.37	133.09	243.48

Fonte: Elaborado pelo autor.

4.5.2 Variável "*Throughput*"

O gráfico exibido na Figura 45 mostra o desempenho relacionado ao *throughput* (ou vazão) de processos concluídos obtida por cada algoritmo de escalonamento submetido.

Figura 45 – Resultado observado para a variável *Throughput* obtida pela execução de 100 replicações independentes dos algoritmos de escalonamento PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS sobre a caracterização de processos do Cenário V (alta taxa de chegada de processos, único processador e três dispositivos de I/O).



Fonte: Elaborado pelo autor.

Nota-se que o algoritmo *fuzzy* FPCS e o algoritmo SJF resultaram em uma vazão maior na maioria dos casos. Já os algoritmos *fuzzy* PFCS e IFCS se igualaram ao algoritmo PRTY no valor da mediana, porém obtiveram o maior valor máximo da amostra. Mais detalhes podem ser visualizados na Tabela 26.

Tabela 26 – Resumo com a estatística descritiva dos resultados observados para a variável *Throughput* a partir de 100 replicações independentes do modelo de simulação para os algoritmos PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS quando experimentados sobre a caracterização do Cenário V (alta taxa de chegada de processos, único processador e três dispositivos de I/O), com destaque para os melhores resultados, segundo a mediana.

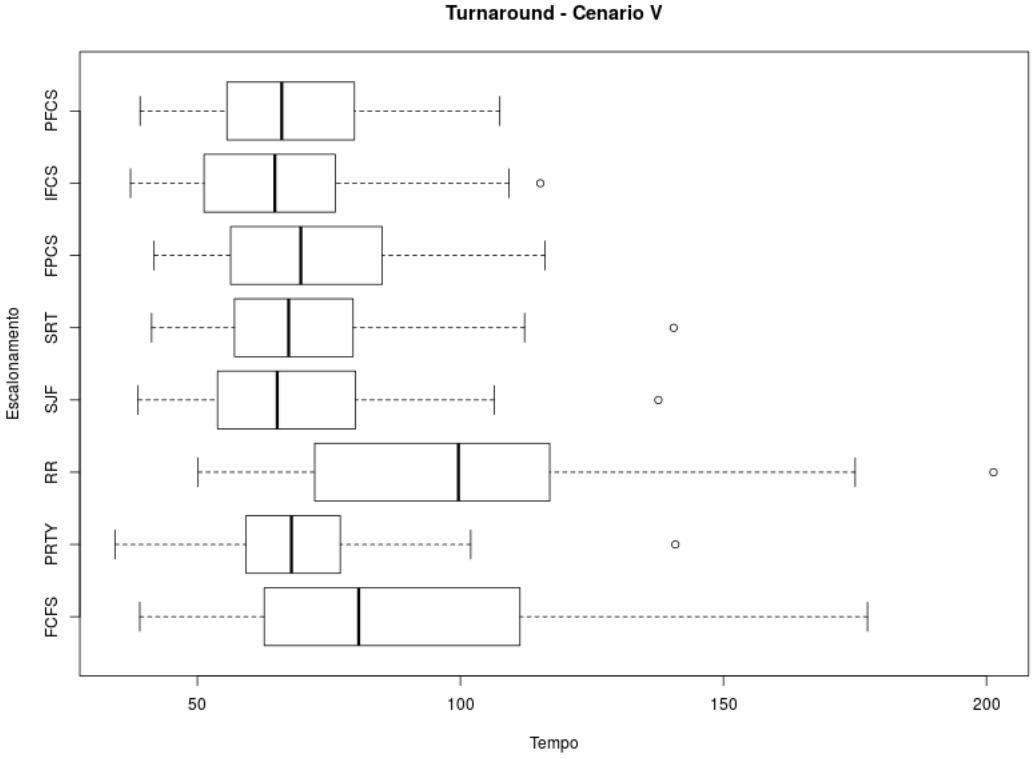
Método	Mínimo	1º Quartil	Mediana	Média	3º Quartil	Máximo
PFCS	15.00	23.00	**26.00**	25.59	28.00	34.00
IFCS	17.00	23.50	**26.00**	25.77	29.00	34.00
FPCS	19.00	25.00	**27.00**	27.17	30.00	33.00
SRT	15.00	24.00	**26.00**	26.02	28.00	33.00
SJF	12.00	24.00	**27.00**	26.28	29.00	34.00
RR	14.00	22.00	24.00	24.09	27.00	31.00
PRTY	17.00	24.00	**26.00**	26.13	28.00	33.00
FCFS	18.00	22.00	25.00	24.75	27.00	31.00

Fonte: Elaborado pelo autor.

4.5.3 Variável "*Turnaround*"

A Figura 46 apresenta o gráfico obtido através dos valores resultantes da análise do indicador *turnaround*.

Figura 46 – Resultado observado para a variável *Turnaround* obtida pela execução de 100 replicações independentes dos algoritmos de escalonamento PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS sobre a caracterização de processos do Cenário V (alta taxa de chegada de processos, único processador e três dispositivos de I/O).



Fonte: Elaborado pelo autor.

Analisando no gráfico o valor da mediana e amplitude dos resultados é evidente que os algoritmos IFCS e PFCS se sobressaíram sobre os demais. O algoritmo PRTY obteve o melhor valor mínimo, porém com a ocorrência de um outlier com valor elevado. É possível verificar os dados em detalhes analisando a Tabela 27 abaixo.

Tabela 27 – Resumo com a estatística descritiva dos resultados observados para a variável *Turnaround* a partir de 100 replicações independentes do modelo de simulação para os algoritmos PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS quando experimentados sobre a caracterização do Cenário V (alta taxa de chegada de processos, único processador e três dispositivos de I/O), com destaque para os quatro melhores resultados, segundo a mediana.

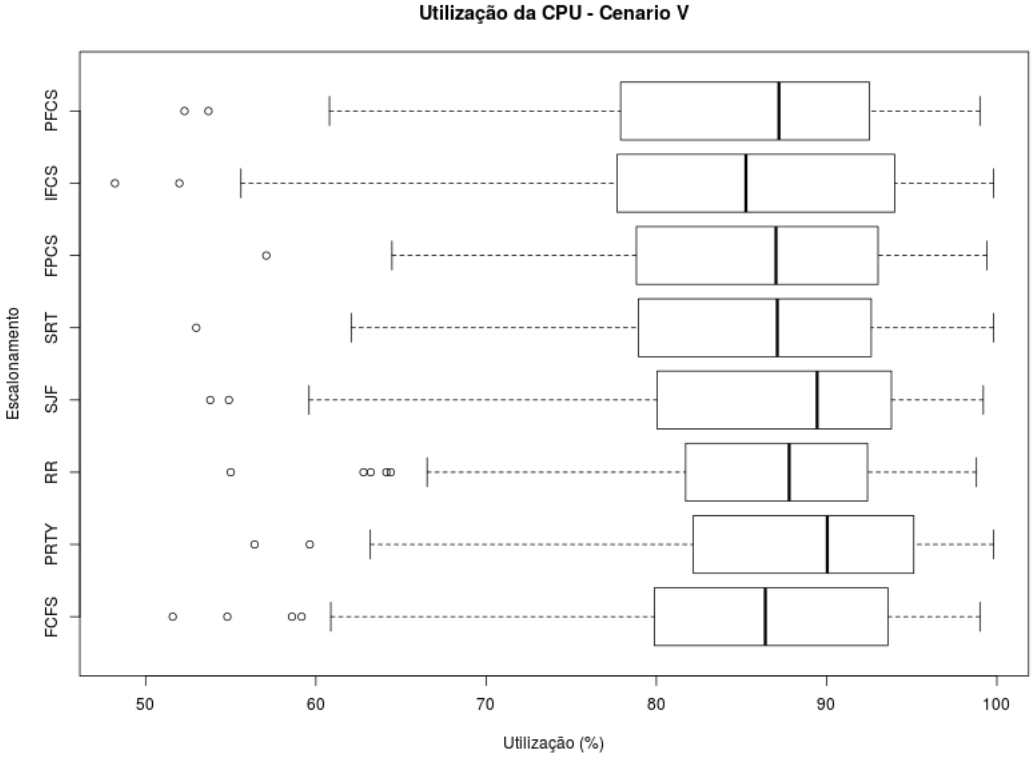
Método	Mínimo	1º Quartil	Mediana	Média	3º Quartil	Máximo
PFCS	39.12	55.62	**66.04**	67.90	79.86	107.50
IFCS	37.29	51.27	**64.73**	65.61	76.21	115.19
FPCS	41.77	56.32	69.67	70.99	85.13	116.07
SRT	41.31	57.05	**67.35**	70.31	79.60	140.54
SJF	38.72	53.83	**65.19**	67.48	80.06	137.63
RR	50.14	72.30	99.63	99.23	116.98	201.33
PRTY	34.33	59.25	67.88	68.75	77.17	140.87
FCFS	39.06	62.71	80.67	89.55	111.33	177.38

Fonte: Elaborado pelo autor.

4.5.4 Variável "Utilização da CPU"

É possível analisar o aproveitamento da CPU por parte de cada algoritmo aplicado ao cenário experimental V no gráfico apresentado na Figura 47.

Figura 47 – Resultado observado para a variável Utilização da CPU obtida pela execução de 100 replicações independentes dos algoritmos de escalonamento PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS sobre a caracterização de processos do Cenário V (alta taxa de chegada de processos, único processador e três dispositivos de I/O).



Fonte: Elaborado pelo autor.

Neste indicador, o algoritmo que mais se destacou foi o PRTY, alcançando uma mediana de 90.4% de utilização e máximo de 99.8%. De forma geral, os algoritmos na maior parte das replicações tiveram aproveitamento entre 85% e 90%. Uma análise mais profunda e detalhada pode ser feita observando os valores apresentados na Tabela 28.

Tabela 28 – Resumo com a estatística descritiva dos resultados observados para a variável Utilização da CPU a partir de 100 replicações independentes do modelo de simulação para os algoritmos PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS quando experimentados sobre a caracterização do Cenário V (alta taxa de chegada de processos, único processador e três dispositivos de I/O), com destaque para os quatro melhores resultados, segundo a mediana.

Método	Mínimo	1º Quartil	Mediana	Média	3º Quartil	Máximo
PFCS	52.29	77.90	**87.20**	84.72	92.52	99.01
IFCS	48.20	77.69	85.26	84.07	94.00	99.80
FPCS	57.09	78.83	87.03	85.55	93.02	99.40
SRT	52.98	78.94	87.10	85.23	92.62	99.80
SJF	53.80	80.04	**89.44**	85.77	93.81	99.20
RR	55.00	81.72	**87.80**	85.76	92.40	98.80
PRTY	56.40	82.17	**90.04**	87.66	95.12	99.80
FCFS	51.59	79.88	86.40	85.17	93.60	99.01

Fonte: Elaborado pelo autor.

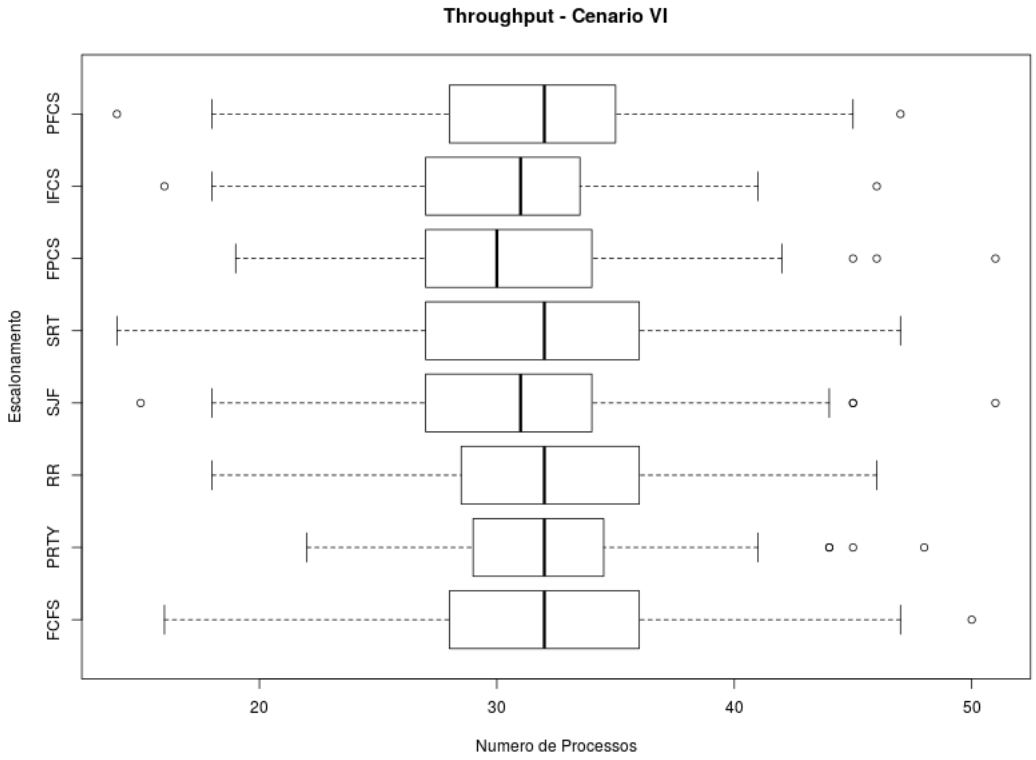
4.6 Cenário VI

Serão apresentados nesta seção os resultados obtidos com a aplicação de cada método de escalonamento ao cenário experimental VI, cuja configuração para experimentação considerou 4 (quatro) núcleos e 4 (quatro) dispositivos de I/O.

4.6.1 Variável "Tempo de Espera"

No gráfico apresentado pela Figura 48, é exposto o desempenho de cada algoritmo aplicado ao cenário experimental IV com relação ao tempo de espera obtido pelas execuções.

Figura 48 – Resultado observado para a variável Tempo de Espera obtida pela execução de 100 replicações independentes dos algoritmos de escalonamento PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS sobre a caracterização de processos do Cenário VI (alta taxa de chegada de processos, quatro processadorres e quatro dispositivos de I/O).



Fonte: Elaborado pelo autor.

Neste cenário o tempo de espera entre todos os algoritmos teve variações de 0 a 5 milissegundos. O baixo tempo de espera se deve a grande quantidade de núcleos da CPU disponibilizada pelo cenário. Com quatro núcleos de processamento e 4 dispositivos de I/O, a fila de prontos raramente terá mais de quatro processos. Como o valor da mediana para os algoritmos é praticamente zero, vamos analisar o valor máximo de tempo de espera. Neste quesito, o algoritmo PRTY leva vantagem, tendo apenas 2 milissegundos de espera no seu pior caso. Como no geral o tempo de espera foi muito próximo de zero, houve uma grande ocorrência de outliers. A Tabela 29 exibe os valores detalhados do experimento.

Tabela 29 – Resumo com a estatística descritiva dos resultados observados para a variável *Throughput* a partir de 100 replicações independentes do modelo de simulação para os algoritmos PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS quando experimentados sobre a caracterização do Cenário VI (alta taxa de chegada de processos, quatro processadorres e quatro dispositivos de I/O), com destaque para os quatro melhores resultados, segundo a mediana.

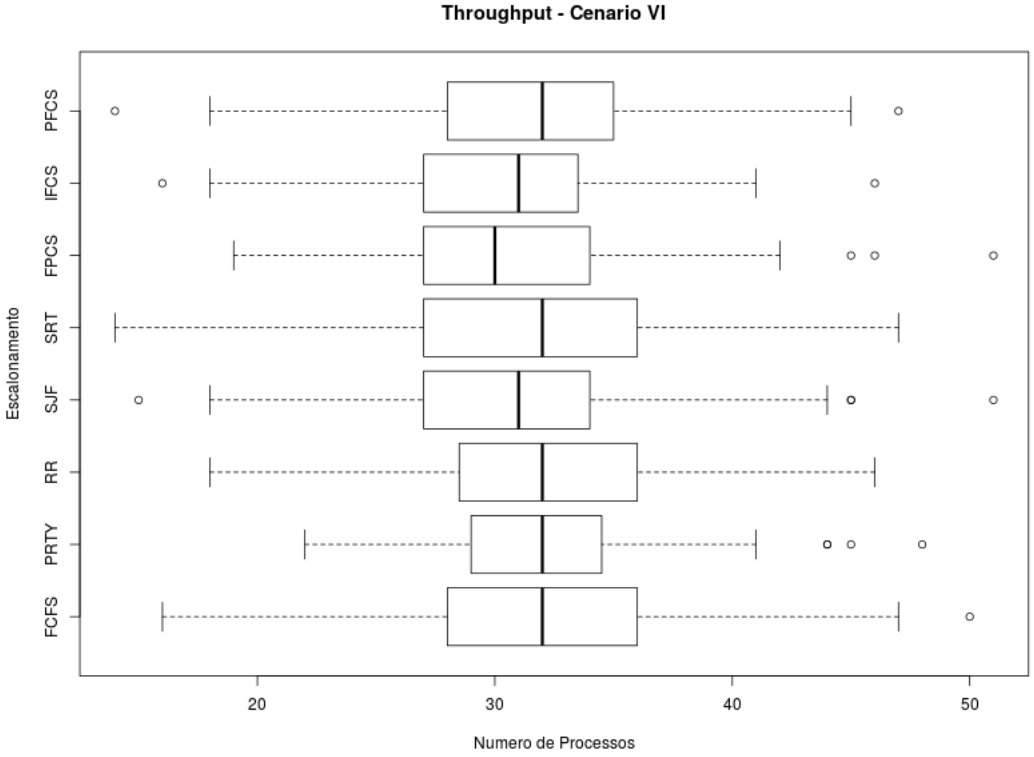
Método	Mínimo	1º Quartil	Mediana	Média	3º Quartil	Máximo
PFCS	0.00	0.00	0.02	0.26	0.24	2.50
IFCS	0.00	0.00	**0.00**	0.28	0.17	4.00
FPCS	0.00	0.00	**0.00**	0.23	0.12	2.16
SRT	0.00	0.00	0.09	0.39	0.61	4.33
SJF	0.00	0.00	0.03	0.33	0.24	3.00
RR	0.00	0.00	0.02	0.29	0.28	2.50
PRTY	0.00	0.00	**0.00**	0.12	0.08	2.00
FCFS	0.00	0.00	**0.00**	0.21	0.10	4.16

Fonte: Elaborado pelo autor.

4.6.2 Variável "*Throughput*"

O desempenho de cada algoritmo ao se analisar a vazão de processos é apresentado no gráfico exibido pela Figura 49, que considerou na sua mensuração o efeito de quatro núcleos e quatro dispositivos de I/O.

Figura 49 – Resultado observado para a variável *Throughput* obtida pela execução de 100 replicações independentes dos algoritmos de escalonamento PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS sobre a caracterização de processos do Cenário VI (alta taxa de chegada de processos, quatro processadorres e quatro dispositivos de I/O).



Fonte: Elaborado pelo autor.

Nesta análise fica em vantagem também o algoritmo PRTY que, apesar de apresentar alguns outliers, conseguiu uma baixa variabilidade de resultados, mantendo seus extremo próximos à mediana. O algoritmo *fuzzy* PFCS obteve a mesma mediana do PRTY, porém uma maior amplitude de resultados. Para mais detalhes, os valores analisados podem ser consultados na Tabela 30.

Tabela 30 – Resumo com a estatística descritiva dos resultados observados para a variável *Throughput* a partir de 100 replicações independentes do modelo de simulação para os algoritmos PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS quando experimentados sobre a caracterização do Cenário VI (alta taxa de chegada de processos, quatro processadorres e quatro dispositivos de I/O), com destaque para os melhores resultados, segundo a mediana.

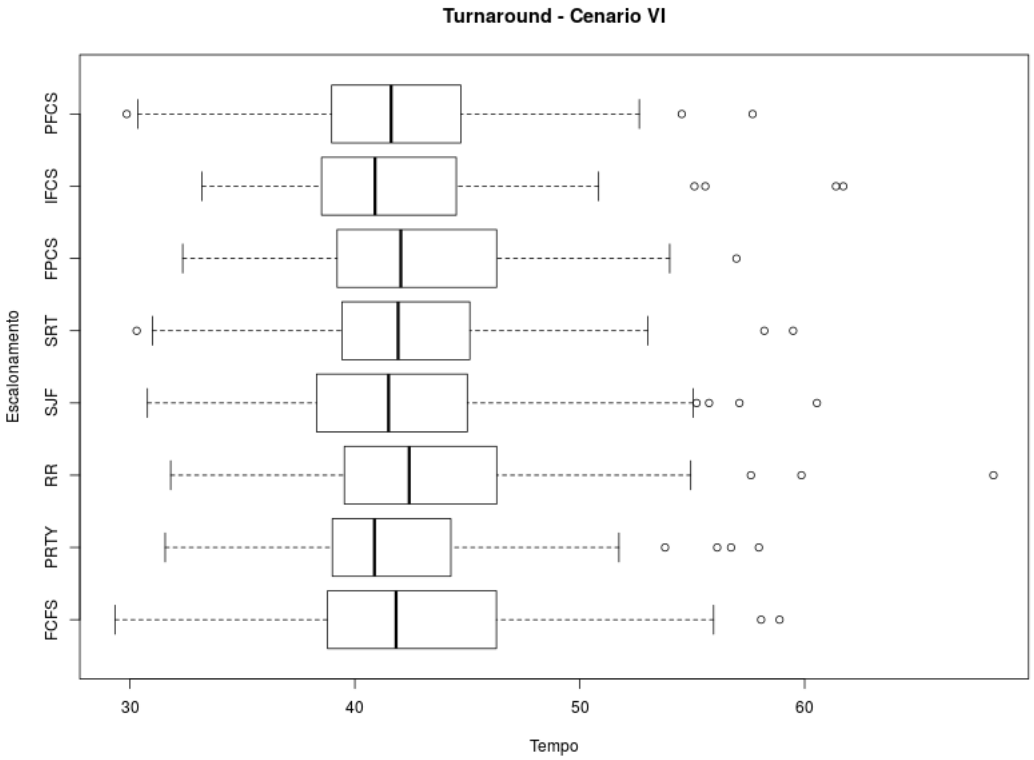
Método	Mínimo	1º Quartil	Mediana	Média	3º Quartil	Máximo
PFCS	14.00	28.00	**32.00**	31.82	35.00	47.00
IFCS	16.00	27.00	31.00	30.68	33.50	46.00
FPCS	19.00	27.00	30.00	31.01	34.00	51.00
SRT	14.00	27.00	**32.00**	31.69	36.00	47.00
SJF	15.00	27.00	31.00	30.87	34.00	51.00
RR	18.00	28.50	**32.00**	32.22	36.00	46.00
PRTY	22.00	29.00	**32.00**	31.90	34.50	48.00
FCFS	16.00	28.00	**32.00**	31.82	36.00	50.00

Fonte: Elaborado pelo autor.

4.6.3 Variável "*Turnaround*"

O indicador de desempenho *turnaround* de cada algoritmo de escalonamento simulado no cenário experimental VI pode ser observado na Figura 50

Figura 50 – Resultado observado para a variável *Turnaround* obtida pela execução de 100 replicações independentes dos algoritmos de escalonamento PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS sobre a caracterização de processos do Cenário VI (alta taxa de chegada de processos, quatro processadorres e quatro dispositivos de I/O).



Fonte: Elaborado pelo autor.

Os algoritmos PFCS, IFCS e PRTY foram os que mais se destacaram quando o objetivo foi minimizar o tempo de turnaround. Dentre eles, o algoritmo IFCS foi o que obteve menor amplitude de resultados. Uma análise mais detalhada dos valores apresentados pelos algoritmos neste indicador pode ser observada na Tabela 31

Tabela 31 – Resumo com a estatística descritiva dos resultados observados para a variável *Turnaround* a partir de 100 replicações independentes do modelo de simulação para os algoritmos PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS quando experimentados sobre a caracterização do Cenário VI (alta taxa de chegada de processos, quatro processadorres e quatro dispositivos de I/O), com destaque para os quatro melhores resultados, segundo a mediana.

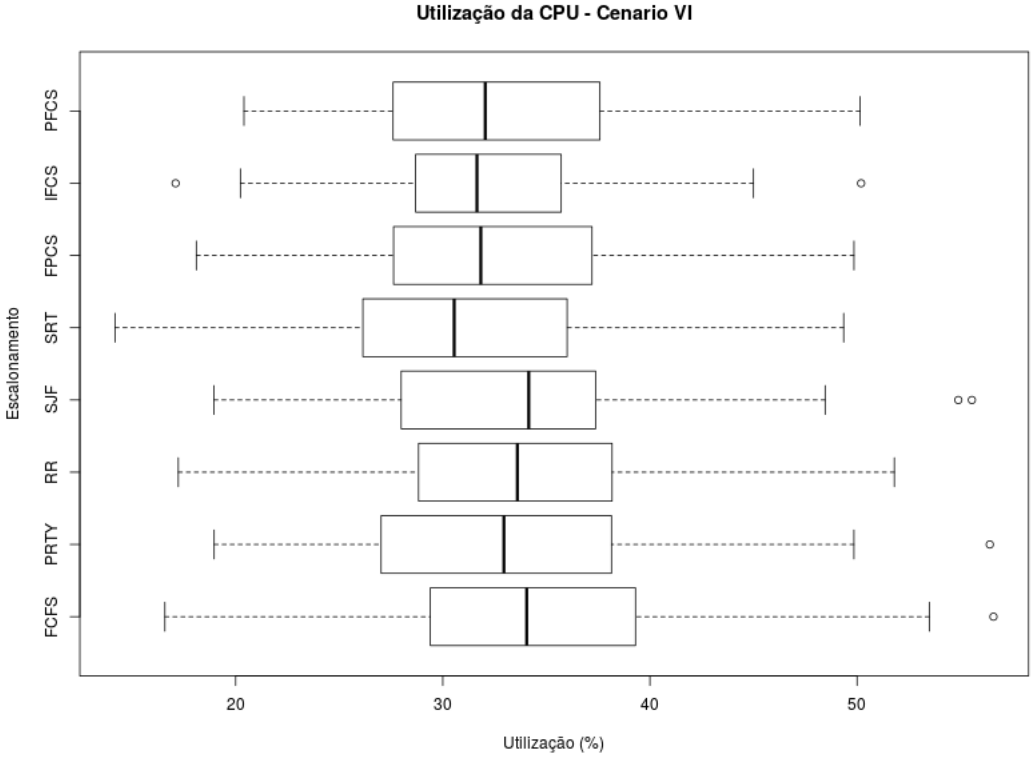
Método	Mínimo	1º Quartil	Mediana	Média	3º Quartil	Máximo
PFCS	29.86	38.97	**41.61**	42.04	44.71	57.69
IFCS	33.20	38.52	**40.90**	41.89	44.51	61.72
FPCS	32.34	39.21	42.05	42.71	46.30	56.97
SRT	30.30	39.43	41.92	41.92	45.11	59.49
SJF	30.78	38.30	**41.50**	41.97	45.00	60.55
RR	31.81	39.53	42.42	43.33	46.31	68.40
PRTY	31.57	39.00	**40.87**	41.92	44.27	57.96
FCFS	29.33	38.78	41.83	43.01	46.29	58.88

Fonte: Elaborado pelo autor.

4.6.4 Variável "Utilização da CPU"

Iremos analisar agora qual o percentual de carga da CPU para cada algoritmo submetido ao cenário simulado experimentou. O gráfico de boxplot comparativo para este indicador de desempenho pode ser observado na Figura 51.

Figura 51 – Resultado observado para a variável Utilização da CPU obtida pela execução de 100 replicações independentes dos algoritmos de escalonamento PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS sobre a caracterização de processos do Cenário VI (alta taxa de chegada de processos, quatro processadorres e quatro dispositivos de I/O).



Fonte: Elaborado pelo autor.

É possível notar que os algoritmos conseguiram atingir uso efetivo da CPU entre 30% e 35%. O algoritmo SJF obteve vantagem com mediana no valor 34.15%. O valor máximo atingido no experimento foi de 56.59%, realizado pelo algoritmo FCFS. Como o Cenário VI foi o mais generoso em termos de recursos computacionais (quatro núcleos na CPU e quatro dispositivos de I/O), já era esperado que os algoritmos de escalonamento possuíssem à disposição algum núcleo disponível, na maioria dos casos. Novamente, é um cenário onde quem produz a demanda (processos) o faz com taxa menor do que a taxa de consumo da demanda. Os dados detalhados podem ser observados na Tabela 32.

Tabela 32 – Resumo com a estatística descritiva dos resultados observados para a variável Utilização da CPU a partir de 100 replicações independentes do modelo de simulação para os algoritmos PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS quando experimentados sobre a caracterização do Cenário VI (alta taxa de chegada de processos, quatro processadorres e quatro dispositivos de I/O), com destaque para os quatro melhores resultados, segundo a mediana.

Método	Mínimo	1º Quartil	Mediana	Média	3º Quartil	Máximo
PFCS	20.40	27.60	32.05	33.11	37.59	50.15
IFCS	17.10	28.69	31.65	32.24	35.72	50.20
FPCS	18.10	27.62	31.84	32.78	37.21	49.85
SRT	14.17	26.14	30.55	31.35	36.00	49.35
SJF	18.95	27.99	34.15	33.26	37.38	**55.55**
RR	17.23	28.83	33.60	33.84	38.17	**51.80**
PRTY	18.96	27.02	32.95	33.39	38.14	**56.41**
FCFS	16.58	29.40	34.05	34.33	39.30	**56.59**

Fonte: Elaborado pelo autor.

4.7 Resumo dos Resultados Experimentais

Nas seções anteriores foram apresentados resultados sobre o desempenho dos algoritmos implementados neste Trabalho de Conclusão de curso, com breve discussão sobre os que se síram melhor **dentro de cada cenário**. Nesta seção os resultados serão revisitados, porém agora tentando identificar quão robusto cada algoritmo foi diante da variação **entre cenários**. Para tal serão sumarizados os resultados sobre o desempenho de cada algoritmo diante de todos os cenários em que foram experimentados, considerando as variáveis Tempo de Espera, ***Throughput***, ***Turnaround*** e Utilização da CPU. Para isso optou-se por utilizar, nesta comparação, a mediana dos valores observados de cada uma destas variáveis, por ser menos suscetível aos seus valores extremos.

A Tabela 33 apresenta estes resultados para a variável Tempo de Espera, destacando sempre os quatro melhores algoritmos e o valor médio de cada algoritmo ao longo dos 6 cenários.

Tabela 33 – Desempenho, segundo mediana, dos algoritmos PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS nos diferentes cenários considerando a variável Tempo de Espera, com destaque para o valor médio observado entre todos os experimentos.

	Cenário I	Cenário II	Cenário III	Cenário IV	Cenário V	Cenário VI	Média
PFCS	**261.60**	**147.68**	**25.10**	20.42	79.23	0.02	**54.49**
IFCS	**261.40**	**145.08**	26.67	**19.78**	**69.03**	**0.00**	**52.11**
FPCS	266.00	160.30	31.12	22.33	78.20	**0.00**	58.39
SRT	**236.80**	**144.30**	**24.85**	**19.29**	**64.59**	0.09	**50.62**
SJF	**245.40**	**140.27**	27.79	**16.48**	**55.54**	0.03	**48.02**
RR	302.00	186.40	**9.33**	31.78	104.19	0.02	66.34
PRTY	261.96	166.11	29.52	24.65	84.36	**0.00**	60.92
FCFS	290.40	158.70	**26.44**	23.65	75.07	**0.00**	56.77

Fonte: Elaborado pelo autor.

Os algoritmos *fuzzy* PFCS e IFCS e os algoritmos clássicos SRT e SJF foram os quatro melhores algoritmos na análise de tempo de espera. Vale lembrar que os algoritmos SRT e SJF garantem um tempo de espera ótimo devido a execução de processos menores primeiro, com a ressalva de que esta prática pode causar o postergação indefinida (inanição) de processos relativamente grandes. Enquanto isso, os algoritmos *fuzzy* utilizam tempo de espera como uma das formas de priorizar algoritmos que já se encontram na fila de prontos por períodos longos, apresentado maiores chances de que processos grandes sejam também contemplados por um núcleo durante o horizonte de simulação de 500 ms. Portanto, concluímos que, pelo menos dentro do escopo de experimentação realizado, estes quatro algoritmos são boas escolhas quando o Tempo de Espera é importante. Nesta situação, segundo os resultados, a recomendação é, em ordem de menor valor médio de Tempo de Espera:

`SJF « SRT « IFCS « PFCS « FCFS « FPCS « PRTY « RR`

A próxima análise será feita agora sobre os resultados da variável Throughput. A Tabela 34 resume os resultados, com destaque em negrito para os quatro melhores algoritmos de cada cenário, e valor médio de cada algoritmo entre todos os cenários.

Tabela 34 – Desempenho, segundo mediana, dos algoritmos PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS nos diferentes cenários considerando a variável *Throughput*, com destaque para o valor médio observado entre todos os experimentos.

	Cenário I	Cenário II	Cenário III	Cenário IV	Cenário V	Cenário VI	Média
PFCS	**16.00**	**25.00**	30.00	**20.00**	**26.00**	**32.00**	**24.83**
IFCS	**15.00**	**25.00**	30.00	**20.00**	**26.00**	31.00	**24.50**
FPCS	12.00	23.00	**32.00**	**20.00**	**27.00**	30.00	24.00
SRT	**18.00**	**26.00**	**31.00**	19.00	**26.00**	**32.00**	**25.33**
SJF	**18.00**	**26.00**	**33.00**	**20.00**	**27.00**	31.00	**25.83**
RR	7.00	18.00	30.00	**20.00**	24.00	**32.00**	21.83
PRTY	12.00	22.00	**31.00**	19.00	**26.00**	**32.00**	23.66
FCFS	12.00	22.00	30.00	**20.00**	25.00	**32.00**	23.50

Fonte: Elaborado pelo autor.

Após ser efetuada uma análise na Tabela 34, os algoritmos *fuzzy* IFCS e PFCS e os algoritmos clássicos SRT e SJF foram os mais robustos na análise do indicador *Throughput*. Dentre os oito algoritmos simulados, estes quatro foram os que apresentaram maior quantidade de processos concluídos. É importante salientar que os algoritmos SRT e SJF priorizam sempre os processos curtos, devido a isso obtêm um valor elevado de *Throughput*. Enquanto isso os algoritmos IFCS e PFCS efetuam cálculos que incluem o tempo solicitado pelo processo e o tempo que este já aguarda na fila de prontos, fazendo com que processos longos também sejam executados de forma justa juntamente com processos curtos. Portanto, concluímos que, pelo menos dentro do escopo de experimentação realizado, estes quatro algoritmos são boas escolhas quando o *Throughput* (vazão) é importante. Nesta situação, segundo os resultados, a recomendação é, em ordem de menor valor médio de *Throughput*:

`SJF « SRT « PFCS « IFCS « FPCS « PRTY « FCFS « RR`

Seguindo com a mesma idéia de consolidação da análise, a Tabela 35 apresenta os valores de mediana observados para a a variável *Turnaround*.

Tabela 35 – Desempenho, segundo mediana, dos algoritmos PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS nos diferentes cenários considerando a variável *Turnaround*, com destaque para para o valor médio observado entre todos os experimentos.

	Cenário I	Cenário II	Cenário III	Cenário IV	Cenário V	Cenário VI	Média
PFCS	**79.07**	65.22	**42.31**	112.50	**66.04**	**41.61**	**67.79**
IFCS	**78.25**	**64.19**	42.70	**110.73**	**64.73**	**40.90**	**66.91**
FPCS	100.00	83.33	44.38	**111.84**	69.67	42.05	75.21
SRT	**56.56**	**51.59**	**41.25**	115.32	**67.35**	41.92	**62.33**
SJF	**65.24**	**60.71**	**42.62**	116.76	**65.19**	**41.50**	**65.33**
RR	169.57	112.21	44.29	**112.21**	99.63	42.42	96.72
PRTY	82.54	**64.81**	44.38	**108.18**	67.88	**40.87**	68.11
FCFS	167.3	103.95	**41.67**	116.17	80.67	41.83	91.93

Fonte: Elaborado pelo autor.

Analisando a Tabela 35, temos novamente os algoritmos *fuzzy* PFCS e IFCS dentre os quatro melhores juntamente com os algoritmos clássicos SRT e SJF. Em alguns casos os algoritmos *fuzzy* citados obtiveram melhores valores de *turnaround* do que o SRT e SJF, que garantem um bom tempo de *turnaround* devido a sua construção. Portanto, concluímos que, pelo menos dentro do escopo de experimentação realizado, estes quatro algoritmos são boas escolhas quando o *Turnaround* é importante. Nesta situação, segundo os resultados, a recomendação é, em ordem de menor valor médio de *Turnaround*:

`SRT « SJF « IFCS « PFCS « PRTY « FPCS « FCFS « RR`

Por fim a análise se estende agora para o caso específico da Utilização da CPU. Os valores observados de cada algoritmo, em cada cenário, foram computados e apresentados acima, sendo suas medianas replicadas na Tabela 36 para permitir uma melhor análise.

Tabela 36 – Desempenho, segundo mediana, dos algoritmos PFCS, IFCS, FPCS, SRT, SJF, RR, PRTY e FCFS nos diferentes cenários considerando a variável Utilização da CPU, com destaque para para o valor médio observado entre todos os experimentos.

	Cenário I	Cenário II	Cenário III	Cenário IV	Cenário V	Cenário VI	Média
PFCS	**96.27**	91.52	**65.31**	77.60	**87.20**	32.05	74.99
IFCS	**96.28**	**92.23**	62.26	**78.73**	85.26	31.65	74.40
FPCS	95.36	**92.80**	**66.19**	**78.04**	87.03	31.84	**75.21**
SRT	96.01	91.67	64.72	77.84	87.10	30.55	74.64
SJF	**95.57**	92.09	**66.72**	77.60	**89.44**	**34.15**	**75.92**
RR	**97.01**	**94.21**	**66.47**	**78.00**	**87.80**	**33.60**	**76.18**
PRTY	95.20	90.96	64.26	76.79	**90.04**	**32.95**	75.03
FCFS	95.47	**92.80**	62.82	**78.88**	86.40	**34.05**	**75.07**

Fonte: Elaborado pelo autor.

Analisando a utilização da CPU através da Tabela 36 é visível que o algoritmo RR obteve melhor desempenho neste indicador, ficando entre os quatro melhores em todos os experimentos. Isso ocorre pela alta rotatividade de processo, mantendo a CPU tão ocupada quanto possível. O algoritmo SJF ficou entre os melhores em quatro dos seis experimentos realizados, enquanto o algoritmo clássico FCFS ficou entre os melhores de três experimentos juntamente com os *fuzzy* PFCS, IFCS e FPCS, sendo que destes se destaca com valor médio mais alto o algoritmo FPCS. Portanto, concluímos que, pelo menos dentro do escopo de experimentação realizado, estes quatro algoritmos são boas escolhas quando a Utilização da CPU é importante. Nesta situação, segundo os resultados, a recomendação é, em ordem de menor valor médio de Utilização da CPU:

`RR « SJF « FPCS « FCFS « PRTY « PFCS « SRT « IFCS`

Estes são resultados analisados de acordo com cada variável. É arriscado assumir que eles bastam para recomendar algoritmos para propósito geral, visto que, a cada situação os algoritmos podem ter ou não um bom desempenho. Para comparação, temos a Tabela 37, que mostra em ordem crescente de desempenho para cada algoritmo segundo a variável analisada.

Tabela 37 – Algoritmos de escalonamento em ordem de desempenho

Indicador	Classificação
Tempo de Espera	`SJF « SRT « IFCS « PFCS « FCFS « FPCS « PRTY « RR`
Throughput	`SJF « SRT « PFCS « IFCS « FPCS « PRTY « FCFS « RR`
Turnaround	`SRT « SJF « IFCS « PFCS « PRTY « FPCS « FCFS « RR`
Utilização CPU	`RR « SJF « FPCS « FCFS « PRTY « PFCS « SRT « IFCS`

Fonte: Elaborado pelo autor.

5 CONSIDERAÇÕES FINAIS

O projeto descrito neste documento consistiu no desenvolvimento de um protótipo de simulador de escalonador de processos, no estudo da aplicação de técnicas de inteligência computacional, especificamente lógica *fuzzy*, no ambiente de gerenciamento de CPU e na análise experimental em forma de comparativo entre algoritmos que utilizam *fuzzy* em sua construção e os algoritmos clássicos mais encontrados na literatura. Diante do exposto entende-se que os objetivos esperados com a realização do projeto foram atingidos com êxito.

Como resultado, os algoritmos que utilizam abordagem *fuzzy* PFCS e IFCS foram revelados como algoritmos robustos da experimentação juntamente com os algoritmos clássicos SRT e SJF, porém, os algoritmos SRT e SJF possuem estratégias de gerenciamento que desfavorecem processos com CPU *bursts* longos, principalmente em sistemas onde os processos podem assumir magnitudes diversas em termos de tempo de execução na CPU. A construção destes algoritmos favorece processos curtos e desfavorece processos longos, facilitando a ocorrência de postergação indefinida (inanição), que é quando o processo aguarda na fila de prontos por tempo indefinido, sem expectativa de quando poderá executar. Isso faz com que, em um sistema onde existe aglomeração de processos curtos e processos longos, os algoritmos de escalonamento baseados em lógica *fuzzy* citados acima possam ser boas escolhas para atingir bom desempenho nas variáveis analisadas, e evitar a postergação indefinida. A aplicabilidade dos resultados também se dá pela possibilidade de desenvolvimento de um *hardware* independente que implemente os métodos de escalonamento *fuzzy* citados, uma vez que, com isso, haja poder computacional suficiente para que as tomadas de decisão efetuadas pelos métodos não causem impacto negativo no sistema. É importante ressaltar que os resultados apresentados são teóricos, preliminares e sem nenhuma pretensão de ser imediatamente aplicados à uma situação real sem antes validar a viabilidade das soluções propostas, carecendo portanto de mais investigação em maior escopo e com caracterização mais realista, similar ou idêntica às observadas em sistemas computacionais comerciais ou de *software* livre.

O protótipo de simulador de escalonamento de processos aqui apresentado é transmitido como legado à comunidade acadêmica com o objetivo de estar em constante desenvolvimento. Sendo um protótipo, a ferramenta poderia ser modificada e incrementada a fim de se tornar um software completo, dotado de interface gráfica animada integrada via *callback* com o motor de simulação, que considere o gerenciamento e paginação de memória dos processos, gerenciamento de *threads*, captura de processos reais no sistema operacional e que considere o tempo gasto em tomadas de decisão e trocas de contexto.

Pode ainda servir de suporte para implementação de novas políticas de escalonamento ou de outras pré-existentes, mas não citadas ou implementadas neste trabalho. Pode-se ainda serem implementadas e avaliadas outras configurações para o algoritmo RR. Outro legado importante é o experimento de maior magnitude envolvendo os algoritmos baseados em lógica fuzzy, mais especificamente o FPCS de Bashier et al (2011), IFCS de Behera et al (2012) e PFCS de Ajmani (2013). No melhor de nosso conhecimento, não encontramos nenhum outro trabalho que tenha experimentado todos estes algoritmos em cenários probabilísticos, sendo portanto uma investigação inédita neste sentido.

6 Referências

ADIPUTRA, F.; HIDAYA, H. T., **Simulasi Algoritma Penjadualan Proses**. Program Magister Ilmu Komputer, Universitas Gadjah Mada, 2010.

AJMANI, P.; SETHI, M., **Proposed Fuzzy CPU Scheduling Algorithm (PFCS) for Real Time Operating Systems**. International Journal of Information Technology, Vol. 5, 2 ed., No. 2, 2013.

ARTERO, A. O., **Inteligência Artificial: Teórica e Prática**. 1a. edição. São Paulo: Livraria da Física, ISBN: 9788578610296, 2009.

BANKS, J. (Ed.)., **Handbook of simulation: principles, methodology, advances, applications, and practice**. John Wiley & Sons, 1998.

BANKS, J., **Discrete event system simulation**. Pearson Education India, 2005.

BASHIR, A.; DOJA, M. N.; BISWAS, R.; ALAM, M., **Fuzzy Priority CPU Scheduling Algorithm**. International Journal of Computer Science Issues, Vol. 8, 6 ed., No. 1, 2011.

BEHERA, H. S. PATTANAYAK, R.; MALLICK, P., **An Improved Fuzzy-Based CPU Scheduling (IFCS) Algorithm for Real Time Systems**. International Journal of Soft Computing and Engineering, Vol. 2, 1 ed., 2012.

BITTENCOURT, G. **Inteligência Artificial: ferramentas e teorias.** Florianópolis, UFSC, ISBN 85-328-0138-2, 1998.

BUTT, M. A.; AKRAM, M., **A novel fuzzy decision-making system for CPU scheduling algorithm**. Neural Computing and Applications, v. 27, n. 7, p. 1927-1939, 2016.

COPPIN, B., **Artificial intelligence illuminated**. Jones and Bartlett Publishers, 2004.

COPPIN, B., **Inteligência artificial**. Rio de Janeiro: LTC, 2012

ISOTANI, S. et al. **Uma Ferramenta de Apoio à Aprendizagem de Sistemas Operacionais.** In: 29o Congresso da Sociedade Brasileira de Computação. 2009.

IYODA, E. M., **Inteligência Computacional no Projeto Automático de Redes Neurais Híbridas e Redes Neurofuzzy Heterogêneas**. 2000. 173f. Tese (Mestre em Engenharia Elétrica) - Universidade Estadual de Campinas. Campinas. 2000.

KIRAN, A. S. **Simulation and scheduling. Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice**, p. 677-717, 1998.

MAIA, L. P., **Sosim: Simulador para o ensino de sistemas operacionais**. Rio de Janeiro, 2001.

NETO, L. B. et al., **Minicurso de sistema especialista nebuloso**. XXXVIII Simpósio Brasileiro de Pesquisa Operacional, Goiânia-GO, 2006.

PAUL, R. J.; BALMER, D. W., **Simulation modelling**. Londres: Chartwell-Bratt, 1993.

PIDD, M., **Complementarity in systems modelling. Systems modelling: Theory and practice**, v. 1, p. 20, 2004.

POTTER, K. et al., **Methods for presenting statistical information: The box plot.** Visualization of large and unstructured data sets, v. 4, p. 97-106, 2006.

RIBEIRO, T. P.; LIMA, R. L. B.; LOBO, E. A., **Simuladores de Gerência de Memória e Processador para Auxílio às Aulas Teóricas de Sistemas Operacionais.** In: Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE). 2014. p. 1028.

REIS, F. P.; JÚNIOR, P. A. P.; COSTA, H. A. X., **TBC-SO/WEB: Um software educacional para o ensino de políticas de escalonamento de processos e de alocação de memória em sistemas operacionais.** In: Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE). 2009.

ROBINSON, S., **Simulation: the practice of model development and use**. Chichester: Wiley, 2004.

RUSSEL, S., NORVING, P., **Artificial Intelligence - A Modern Approach**. Prentice-Hall, ISBN 0137903952, 2002.

RUSSEL, S.; NORVING, P., **Inteligência Artificial**. 2a edição, Rio de Janeiro: Elsevier, 2004.

SAUCIER, R., **Computer generation of statistical distributions**. ARMY RESEARCH LAB ABERDEEN PROVING GROUND MD, 2000.

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G., **Fundamentos de Sistemas Operacionais**. 8. ed. Grupo Gen-LTC, 2013.

TANENBAUM, A. S., **Modern Operating Systems**. Prentice Hall PTR, 2001.

TANENBAUM, A. S., **Sistemas Operacionais Modernos**. 3. ed. Pearson, 2010.

TRANSCHEIT, R. **Sistemas fuzzy**. Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2004.

WALPOLE, R. E.; MYERS, R. H., **Probability and statistics for engineers and scientists**. Pearson Education, 1993.

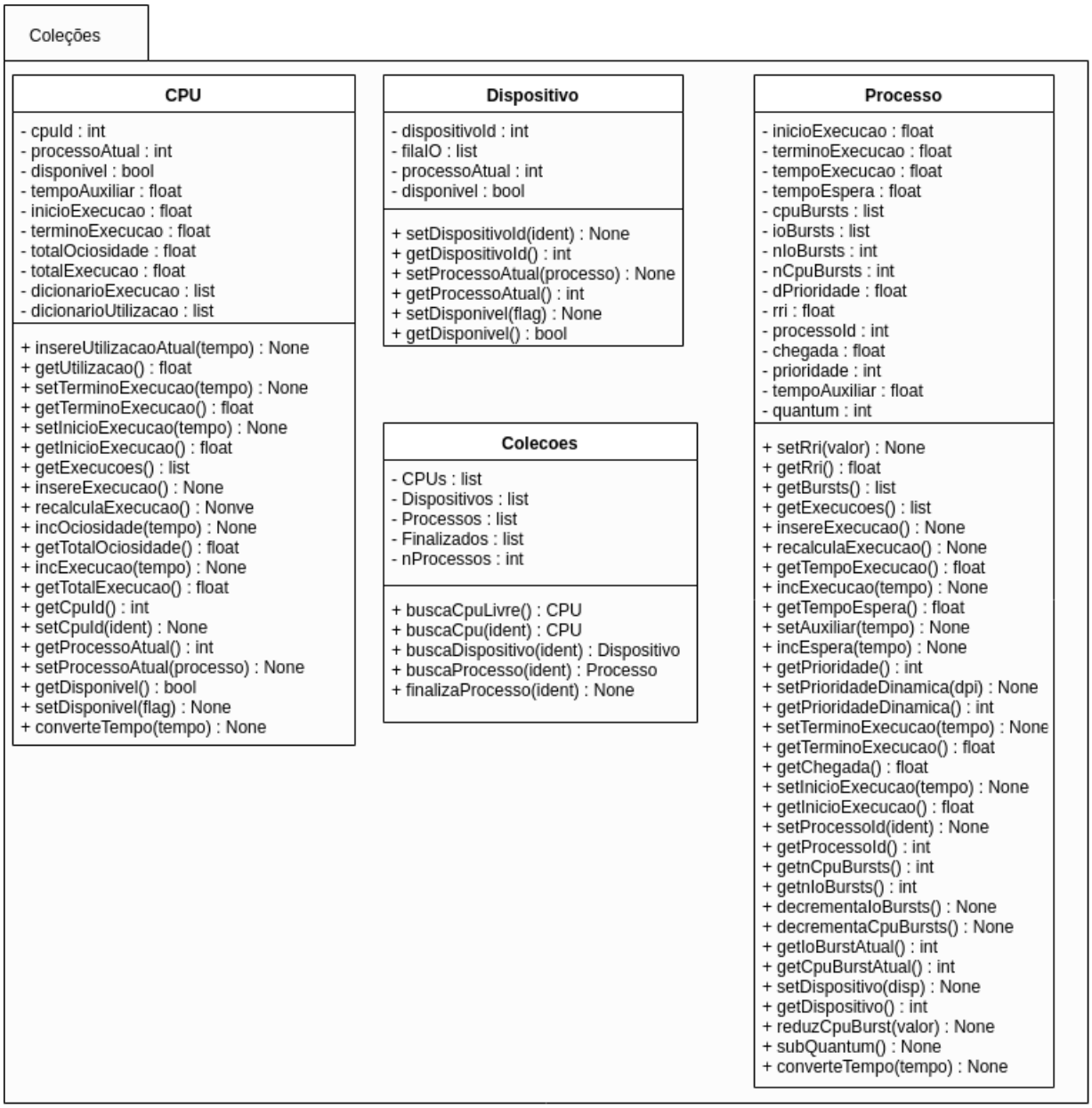
ZADEH, L. A. et al., **Fuzzy sets**. Information and control, v. 8, n. 3, p. 338-353, 1965.

ZADEH, L. A. **Outline of a new approach to the analysis of complex systems and decision processes**. IEEE Transactions on systems, Man, and Cybernetics, n. 1, p. 28-44, 1973.

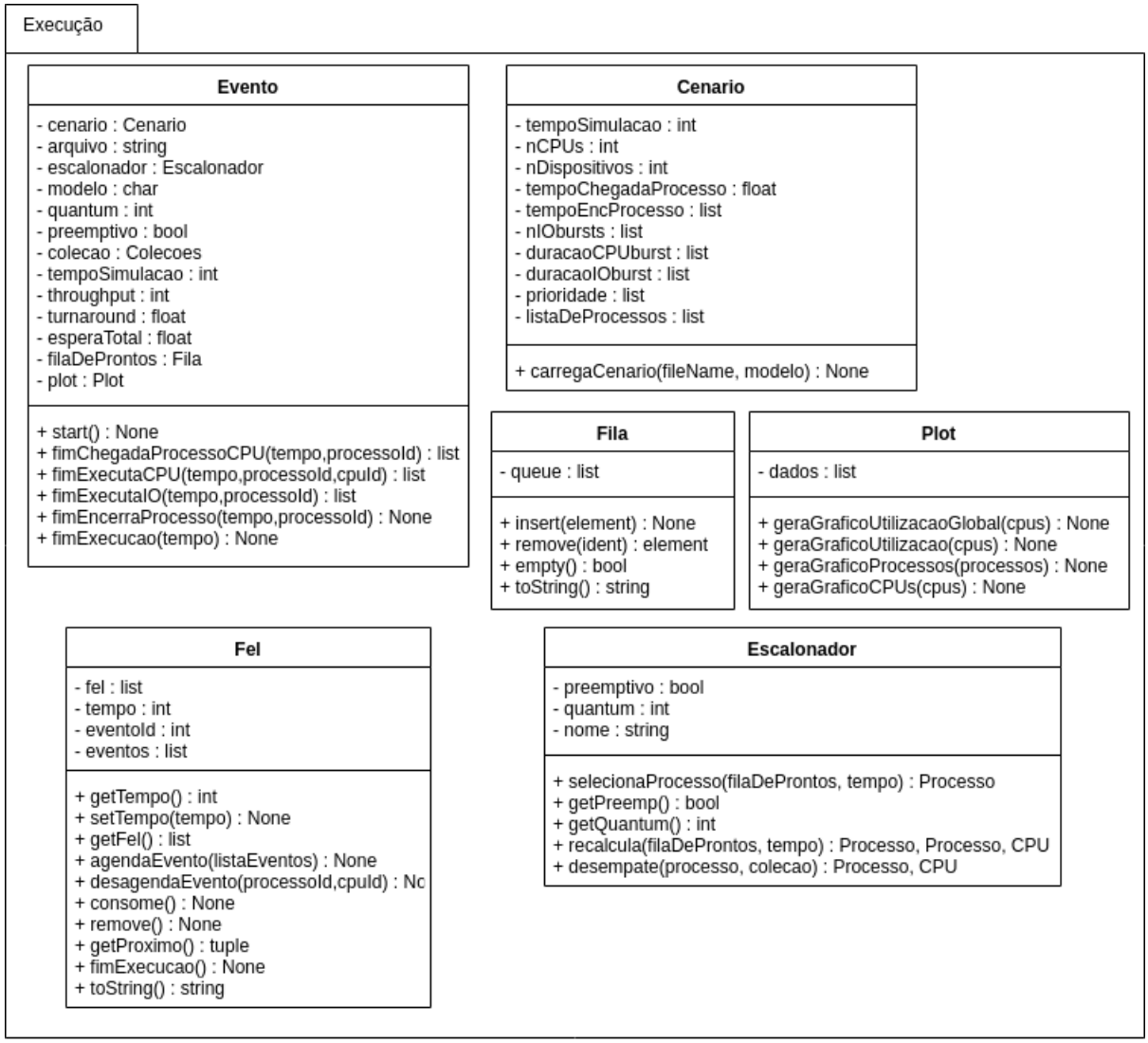
ZADEH, L. A., **The concept of a linguistic variable and its application to approximate reasoning—I**. Information sciences, v. 8, n. 3, p. 199-249, 1975.

Apêndices

APÊNDICE A – Diagrama de Classes



Fonte: Elaborado pelo autor.



Fonte: Elaborado pelo autor.