PROCSIM: ARQUITETURA E IMPLEMENTAÇÃO DE UM SIMULADOR PEDAGÓGICO DE GERENCIAMENTO DE PROCESSOS E ESCALONAMENTO

Henrique Feliciano de Azevedo, Danton Cavalcanti Franco Junior – Orientador

Curso de Bacharel em Ciência da Computação
Departamento de Sistemas e Computação
Universidade Regional de Blumenau (FURB) – Blumenau, SC – Brasil

hfazevedo@furb.br, dfranco@furb.br

Resumo: *O ensino de sistemas operacionais apresenta desafios devido à natureza abstrata de conceitos como escalonamento de processos e concorrência, demandando ferramentas pedagógicas que permitam a visualização prática. Para suprir essa necessidade, este trabalho detalha a arquitetura e implementação do ProcSim, um simulador de gerenciamento de processos cujo objetivo é fornecer um ambiente de alta fidelidade para a experimentação de algoritmos e análise do comportamento de processos. Desenvolvido em C# com uma interface em WPF, o simulador permite a criação de cenários customizados, com múltiplos núcleos e dispositivos de I/O, executando processos cujas instruções são decompostas em micro-operações. A aplicação oferece monitoramento em tempo real do uso de recursos através de uma interface inspirada no Gerenciador de Tarefas do Windows. O projeto final implementa um escalonador por prioridades híbrido que ajusta dinamicamente a prioridade dos processos com base em heurísticas de aging e perfil de uso. O simulador resultante se apresenta como uma ferramenta pedagógica que traduz a complexidade teórica dos escalonadores modernos em uma experiência visual e interativa.*

Palavras-chave*: Sistemas operacionais. Escalonamento de processos. Simulador educacional. Ensino de computação.*

1 INTRODUÇÃO

O ensino de Sistemas Operacionais (SO) é um dos pilares da Ciência da Computação, responsável por elucidar como o software gerencia o hardware de forma eficiente. Contudo, a natureza abstrata de seus conceitos centrais, como gerenciamento de processos, concorrência e o escalonamento da Unidade Central de Processamento (*Central Processing Unit* - CPU), representa um desafio pedagógico significativo (TANENBAUM; WOODHULL, 2008). Para transpor essa barreira, ambientes de simulação são amplamente reconhecidos como ferramentas pedagógicas eficazes, pois permitem que os alunos visualizem e interajam com algoritmos em um ambiente controlado, o que solidifica a compreensão teórica (COSTA et al., 2018).

Apesar da importância dessas ferramentas, uma lacuna é observada em simuladores que aliam alta fidelidade técnica a uma rica interface de visualização. Muitas das soluções disponíveis, embora valiosas, ou simplificam excessivamente a mecânica interna do sistema, tratando a execução de um processo como um mero atraso de tempo, ou carecem de opções de personalização e de interfaces que permitam um monitoramento detalhado das métricas de desempenho (CRUZ; JESUS; MARQUES FILHO, 2023). Essa simplificação pode levar a uma compreensão incompleta dos mecanismos que governam os sistemas operacionais modernos.

Para superar tais limitações, este trabalho detalha o desenvolvimento do ProcSim, um simulador de gerenciamento de processos projetado especificamente como uma ferramenta pedagógica de alta fidelidade. O seu principal diferencial reside em um motor de simulação que abandona a abstração simplista, comum em outras ferramentas, que trata a execução de um processo como um mero atraso de tempo. Em vez disso, o ProcSim processa uma lista detalhada de instruções e micro-operações para cada processo, emulando de forma mais autêntica o trabalho realizado por um núcleo de CPU.

Essa abordagem, aliada a uma arquitetura modular que separa o núcleo de simulação (*backend*) da interface de usuário (*User Interface* – UI) (*frontend*), resulta em um laboratório virtual robusto. Nele, o estudante pode não apenas configurar e observar cenários complexos, mas também interagir diretamente com a simulação em tempo real, por exemplo, alterando a prioridade de um processo ou encerrando-o para analisar o impacto imediato no comportamento do sistema. Desta forma, o ProcSim transforma conceitos teóricos em eventos visuais, interativos e mensuráveis. A fim de elucidar a aplicação prática da arquitetura proposta e o seu uso como ferramenta pedagógica, um guia de utilização é apresentado no "Apêndice B - Jornada do Usuário".

O objetivo geral deste trabalho é, portanto, desenvolver um simulador visual que permita a simulação de algoritmos de escalonamento em um ambiente *multicore*, o monitoramento detalhado de métricas de desempenho e a visualização interativa do ciclo de vida dos processos.

Para alcançar tal objetivo, foram definidos os seguintes objetivos específicos:

a) implementar os algoritmos de escalonamento Round Robin e por Prioridades, com um modelo híbrido inspirado em sistemas operacionais comerciais;
b) desenvolver uma interface gráfica interativa, inspirada no Gerenciador de Tarefas do Windows, utilizando as tecnologias WPF e LiveCharts para a visualização de dados em tempo real;
c) simular de forma detalhada o gerenciamento de processos, incluindo a estrutura completa de Blocos de Controle de Processo (*Process Control Blocks* – PCBs), e operações de Entrada/Saída (*Input/Output* – I/O) com filas e canais dedicados;
d) estruturar o simulador com uma arquitetura modular e desacoplada, utilizando o padrão *Model-View-ViewModel* (MVVM), para garantir a manutenibilidade e a extensibilidade da ferramenta.

2 FUNDAMENTAÇÃO TEÓRICA

Esta seção apresenta a base conceitual que fundamenta o design e a implementação do ProcSim, abordando os elementos de sistemas operacionais, as técnicas de escalonamento e o uso de tecnologias de software específicas.

2.1 CONCEITOS, TÉCNICAS E FERRAMENTAS

A seguir são descritos os principais conceitos e tecnologias que norteiam o desenvolvimento do simulador.

2.1.1 Processos e Gerenciamento

Em um SO, um processo é a instância de um programa em execução. Para gerenciar a concorrência, o sistema mantém para cada processo uma estrutura de dados chamada PCB, que armazena seu contexto completo: identificador, estado, contador de programa, valores dos registradores e informações de escalonamento. Um processo transita por diversos estados durante seu ciclo de vida, sendo os principais: novo, pronto, executando, bloqueado e concluído. A troca de contexto, mecanismo pelo qual a CPU alterna entre processos, depende do salvamento e restauração eficientes dessas informações no PCB (SILBERSCHATZ; GALVIN; GAGNE, 2015). No ProcSim, o PCB é uma estrutura central que armazena dados essenciais para a simulação, como o estado atual do processo, o contador de programa, os valores dos registradores e a prioridade de escalonamento, permitindo uma representação fiel da troca de contexto.

2.1.2 Chamadas de Sistema

A separação entre os modos de execução modo de usuário e modo de kernel é um princípio fundamental para a proteção e estabilidade de um sistema operacional. Aplicações de usuário não podem ter acesso direto ao hardware ou a recursos críticos do sistema. A chamada de sistema (*system call*) é o mecanismo que serve como uma ponte controlada e segura entre esses dois modos. Ela funciona como uma interface de programação de aplicação (*Application Programming Interface* - API) para o *kernel*, permitindo que um processo em modo de usuário solicite serviços privilegiados, como acesso a arquivos, gerenciamento de processos ou, como no caso do ProcSim, a realização de uma operação de I/O (SILBERSCHATZ; GALVIN; GAGNE, 2015). No ProcSim, as `Syscalls`, como `IoRequest` e `Exit`, são modeladas como instruções especiais que disparam uma interrupção de software, transferindo o controle para o `SystemCallDispatcher` na camada core.

2.1.3 Interrupções, *Dispatcher* e Troca de Contexto

Uma interrupção é um sinal enviado à CPU que altera seu fluxo normal de execução, sendo um mecanismo essencial para a multitarefa preemptiva e para a interação com hardware. As interrupções podem ser de dois tipos: de hardware, geradas por dispositivos como o timer do *quantum* ou um `IODevice`; e de software (*traps*), geradas por uma instrução para solicitar um serviço do SO (uma *syscall*) ou para sinalizar um erro.

Ao receber uma interrupção, a CPU conclui a instrução em andamento, salva seu contexto mínimo (como o contador de programa) e desvia a execução para uma Rotina de Serviço de Interrupção (ISR - *Interrupt Service Routine*). A ISR é um código do *kernel* que trata o evento. No ProcSim, os `IInterruptHandler` constroem a sequência de `MicroOp` que representam uma ISR. Após o tratamento, o controle é devolvido ao escalonador, que pode decidir manter o processo corrente ou executar outro. Caso um novo processo seja escolhido, o `Dispatcher` é o módulo do *kernel* invocado para realizar a troca de contexto, que consiste na tarefa de salvar o estado completo do processo antigo em seu PCB e carregar o estado do novo processo nos registradores da CPU (TANENBAUM; WOODHULL, 2008).

2.1.4 Hardware de Temporização e o *Tick* do Sistema

Sistemas computacionais modernos dependem de um componente de hardware essencial conhecido como Temporizador de Intervalo Programável (PIT - *Programmable Interval Timer*), ou de funcionalidades equivalentes

integradas em controladores de interrupção mais avançados. A função primária deste hardware é gerar uma interrupção de temporizador (*timer interrupt*) em uma frequência fixa e programável (SILBERSCHATZ; GALVIN; GAGNE, 2015).

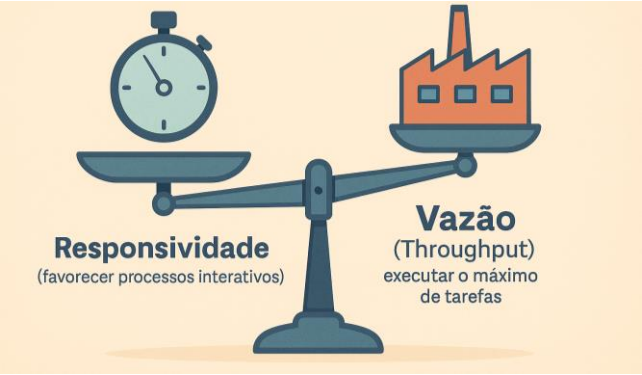
Essa interrupção periódica, comumente chamada de *tick* do sistema, é o que permite ao sistema operacional implementar a preempção. A cada *tick*, a execução do processo atual é compulsoriamente interrompida, e o controle é transferido para o *kernel*. Nesse momento, o SO pode decrementar o *quantum* do processo, verificar se ele expirou e, se necessário, invocar o escalonador para realizar uma troca de contexto. Portanto, o hardware de temporização é a base que possibilita a multitarefa preemptiva em sistemas de tempo compartilhado.

2.1.5 Escalonamento de CPU e seus Objetivos

O escalonador de CPU é o componente do SO que seleciona qual processo na fila de prontos deve ser executado.

Seu design busca equilibrar múltiplos objetivos, muitas vezes conflitantes, como maximizar a vazão do sistema (*throughput*), minimizar o tempo de resposta para usuários interativos (*response time*) e garantir que todos os processos recebam uma parcela justa de tempo da CPU (*fairness*), evitando o fenômeno de inanição (*starvation*) (DOWNEY, 2008). Este dilema de otimização é um ponto central no projeto de escalonadores modernos, como o Escalonador Completamente Justo (*Completely Fair Scheduler* - CFS) do Linux, que foi projetado para oferecer um balanceamento justo e eficiente entre processos interativos e não interativos (LOVE, 2010). O desafio pode ser visualizado como um equilíbrio entre a capacidade de resposta e a produtividade total do sistema, conforme ilustrado na Figura 1.

Figura 1 – O dilema do escalonador: o equilíbrio entre responsividade e vazão



Fonte: elaborado pelo autor (2025).

Os algoritmos de escalonamento são classificados como não-preemptivos, onde um processo em execução mantém o controle da CPU até que a libere voluntariamente, ou preemptivos. Em um modelo preemptivo, o SO pode interromper um processo em execução para alocar a CPU a outro, mesmo que o primeiro não tenha terminado sua tarefa. A preempção é, portanto, o ato de forçar essa troca de contexto, sendo um mecanismo fundamental para a operação de sistemas multitarefa interativos e de tempo compartilhado, pois garante que nenhuma aplicação monopolize o processador e que o sistema permaneça responsivo aos usuários.

2.1.6 Algoritmos de Escalonamento Implementados

O escalonador é a parte fundamental no gerenciamento de processos, e a escolha de seu algoritmo define o comportamento do sistema em termos de eficiência e responsividade. Esse trabalho implementa dois algoritmos preemptivos distintos que permitem explorar desde os fundamentos do tempo compartilhado até as heurísticas de sistemas modernos.

2.1.6.1 Round Robin

O algoritmo Round Robin (RR) é um dos modelos de escalonamento preemptivo mais clássicos e fundamentais. Sua operação se baseia em uma fila de processos prontos, tratada como uma estrutura circular, e em uma unidade de tempo fixa denominada quantum (TANENBAUM; WOODHULL, 2008). Um processo selecionado da fila executa por, no máximo, a duração de um quantum. Se o processo terminar ou se bloquear para uma operação de I/O antes do fim do quantum, a CPU é liberada e o próximo processo da fila é escalonado. Caso contrário, ao expirar o *quantum*, o processo é interrompido por hardware (*timer interrupt*), seu contexto é salvo, e ele é movido para o final da fila de prontos.

A eficácia do RR é altamente dependente da escolha do tamanho do quantum. Um *quantum* muito pequeno resulta em trocas de contexto excessivas, desperdiçando ciclos de CPU em tarefas de sistema em vez de trabalho útil. Por outro lado, um quantum muito grande faz com que o RR se aproxime do comportamento do algoritmo não-preemptivo FCFS (*First-Come, First-Served*), degradando o tempo de resposta para processos curtos e interativos (SILBERSCHATZ;

GALVIN; GAGNE, 2015). No ProcSim, o RR serve como um algoritmo de base essencial para a demonstração desses conceitos e como um ponto de comparação para algoritmos mais complexos.

2.1.6.2 Escalonamento por Prioridades Híbrido

Para atender às diversas demandas de aplicações modernas, os sistemas operacionais comerciais raramente utilizam algoritmos simples. Em vez disso, empregam modelos de escalonamento por prioridades que são dinâmicos e adaptativos. Um sistema de prioridades estático, onde cada processo possui um nível de prioridade fixo, é suscetível ao problema de *starvation*, no qual processos de baixa prioridade podem ser indefinidamente preteridos por um fluxo contínuo de processos de maior prioridade.

Para solucionar essa e outras questões, o ProcSim implementa um escalonador por prioridades híbrido, inspirado em heurísticas utilizadas por sistemas como o Windows, que combina uma prioridade base com ajustes dinâmicos em tempo de execução. Os principais mecanismos são listados a seguir.

2.1.6.2.1 Prioridade Base e Dinâmica

Cada processo possui uma prioridade estática (ou base), definida em sua criação, que serve como referência. O escalonador, no entanto, opera com base em uma prioridade dinâmica, que é recalculada periodicamente para refletir o comportamento recente do processo.

2.1.6.2.2 Boost para Processos I/O-Bound

Processos interativos (como editores de texto e interfaces gráficas) passam a maior parte do tempo bloqueados, aguardando eventos de I/O, como um clique de mouse ou uma leitura de disco. Para garantir a alta responsividade do sistema, é vantajoso que esses processos sejam atendidos rapidamente quando retornam à fila de prontos. O escalonador do Windows, por exemplo, concede um aumento temporário de prioridade (*priority boost*) a um processo que acaba de completar uma operação de I/O, uma heurística que privilegia a experiência do usuário (RUSSINOVICH; SOLOMON; IONESCU, 2012).

2.1.6.2.3 Envelhecimento (*aging*)

Como contramedida direta à inanição, o mecanismo de *aging* aumenta gradualmente a prioridade dinâmica de processos que permanecem na fila de prontos por longos períodos (SILBERSCHATZ; GALVIN; GAGNE, 2015). Isso garante que, eventualmente, até mesmo o processo de menor prioridade base terá sua prioridade dinâmica elevada o suficiente para ser selecionado pelo escalonador, garantindo a equidade no acesso à CPU a longo prazo.

2.1.7 Tecnologias de Desenvolvimento

A implementação foi realizada sobre a plataforma .NET 9 utilizando a linguagem C#. A interface gráfica foi desenvolvida com Windows Presentation Foundation (WPF), um framework de UI que permite a criação de interfaces ricas e a implementação do padrão de projeto MVVM para o desacoplamento da lógica de negócio da apresentação. O padrão MVVM, introduzido pela Microsoft, promove uma clara separação de responsabilidades, onde a *View* (a interface XAML) é desacoplada da *ViewModel* (que expõe dados e comandos), facilitando a testabilidade e a manutenção do código (SMITH, 2009). A visualização de gráficos em tempo real é realizada pela biblioteca LiveCharts2.

2.2 TRABALHOS CORRELATOS

A seguir, são analisados trabalhos que abordam a simulação de conceitos de SO para fins educacionais.

Quadro 1 – MOSS: Uma ferramenta para o auxílio do ensino de sistemas operacionais

Referência	Costa et al. (2018)
Objetivos	Desenvolver uma ferramenta para o ensino de sistemas operacionais, com foco em escalonamento de processos, gerenciamento de memória e arquivos.
Principais funcionalidades	Simulação de processos com escalonamento preemptivo e não-preemptivo, gerenciamento de memória com paginação, controle de arquivos.
Ferramentas de desenvolvimento	MIPS Assembly e Java, integrado ao simulador MARS.
Resultados e conclusões	A ferramenta proporciona um ambiente educacional prático, permitindo aos alunos interagirem diretamente com o SO, ajustando parâmetros de execução e visualizando os resultados em tempo real.

Fonte: elaborado pelo autor (2025).

Quadro 2 – ESORV: Um simulador para o ensino de sistemas operacionais com a tecnologia da realidade virtual

Referência	Scamati (2017)
Objetivos	Desenvolver um simulador em realidade virtual para o ensino de sistemas operacionais, com foco em escalonamento de processos.
Principais funcionalidades	Simulação imersiva do ciclo de vida dos processos, visualização interativa de operações de CPU e I/O, suporte a múltiplos algoritmos de escalonamento.
Ferramentas de desenvolvimento	Tecnologia de realidade virtual e motores gráficos para simulação tridimensional.
Resultados e conclusões	O uso de realidade virtual melhora o aprendizado de conceitos abstratos, como escalonamento, e aumenta o engajamento dos alunos em comparação com métodos tradicionais.

Fonte: elaborado pelo autor (2025).

Quadro 3 – Um estudo comparativo de simuladores de rotinas de sistemas operacionais

Referência	Cruz, Jesus e Marques Filho (2023)
Objetivos	Realizar uma análise comparativa de simuladores de rotinas de sistemas operacionais, destacando funcionalidades para auxiliar no ensino.
Principais funcionalidades	Análise das características: Suporte a algoritmos de escalonamento, gerenciamento de memória e sistemas de arquivos, facilidade de uso e personalização.
Ferramentas de desenvolvimento	Análise de simuladores desenvolvidos em Java, C++ e outras linguagens, com suporte multiplataforma e interfaces gráficas.
Resultados e conclusões	Simuladores com suporte a personalização e interfaces gráficas intuitivas são mais eficazes no ensino de conceitos de sistemas operacionais.

Fonte: elaborado pelo autor (2025).

Diferentemente das abordagens analisadas, que abstraem a execução de instruções, o ProcSim permite uma compreensão mais autêntica do trabalho da CPU. Enquanto outras ferramentas oferecem algoritmos básicos, o escalonador híbrido preenche a lacuna no ensino de heurísticas complexas, como *aging* e *boost* de I/O, presentes em sistemas operacionais modernos. Por fim, o seu sistema de monitoramento visual oferece uma capacidade de análise em tempo real não encontrada de forma integrada nos trabalhos correlatos.

3 DESCRIÇÃO DO SIMULADOR

Esta seção detalha o processo de desenvolvimento do simulador, desde a concepção de seus requisitos até a arquitetura, modelagem e implementação dos componentes de software que compõem o núcleo da simulação.

3.1 REQUISITOS

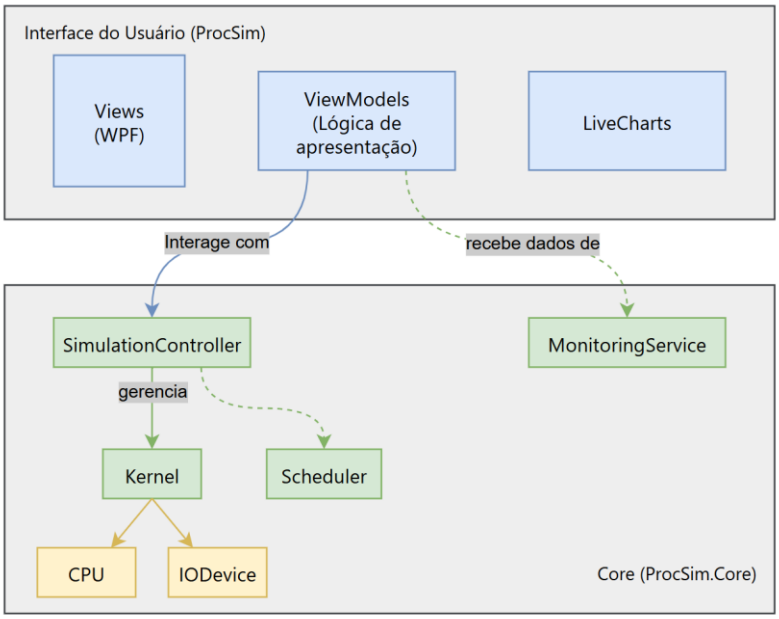
Os principais Requisitos Funcionais (RF) e Requisitos Não Funcionais (RNF) são:

a) simular os algoritmos de escalonamento Round Robin e Prioridades (RF);
b) permitir a configuração da máquina virtual (núcleos, quantum) e dispositivos de I/O (RF);
c) permitir a criação e configuração detalhada de processos e suas operações (RF);
d) exibir gráficos e métricas de desempenho da simulação em tempo real (RF);
e) permitir a interação com processos durante a execução, como alterar prioridade e encerrar (RF);
f) persistir e carregar configurações de simulação em arquivos (RF);
g) adicionar novas instâncias de processos com a simulação em execução (RF);
h) salvar automaticamente a configuração atual ao fechar a aplicação (RNF);
i) possuir uma arquitetura modular e extensível (RNF);
j) apresentar uma interface gráfica intuitiva inspirada no Gerenciador de Tarefas do Windows (RNF);
k) garantir alta fidelidade na simulação através da execução de micro-operações (RNF).

3.2 ARQUITETURA E MODELAGEM DO SISTEMA

Para atender aos requisitos de modularidade e manutenibilidade, a arquitetura do simulador foi dividida em duas camadas principais (Figura 2).

Figura 2 – Diagrama de arquitetura em camadas do ProcSim

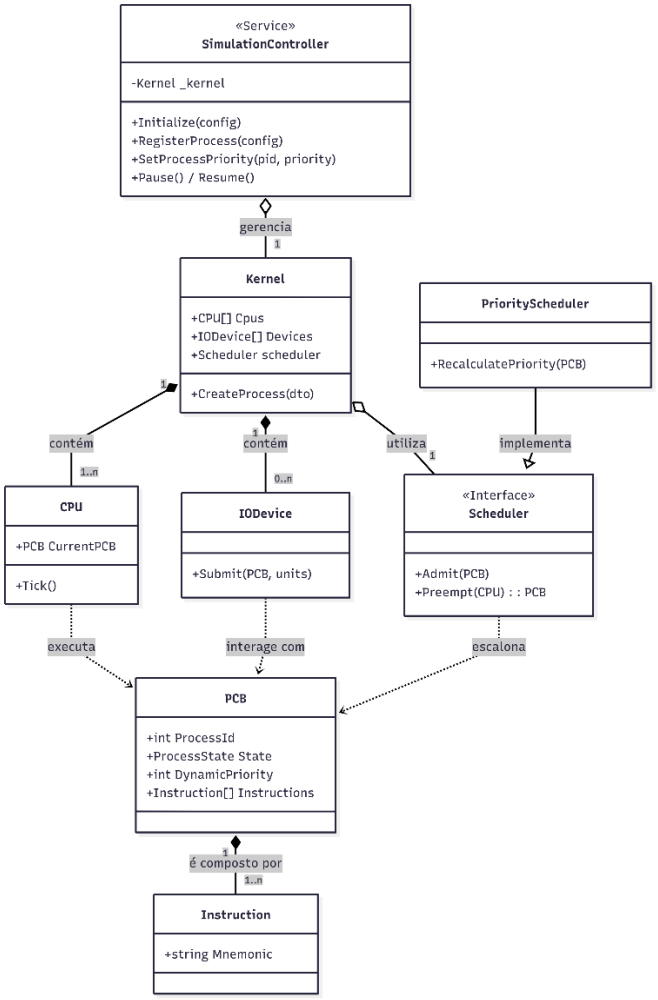


Fonte: elaborado pelo autor (2025).

A arquitetura apresentada evidencia a separação de responsabilidades entre as camadas. A camada Core (`ProcSim.Core`) atua como um motor de simulação autônomo, contendo toda a lógica do `Kernel`, `Scheduler` e dos componentes de hardware simulados (`CPU`, `IODevice`). A camada de UI (`ProcSim`) consome os serviços da camada Core através de um ponto de entrada único, o `SimulationController` (que controla o ciclo de vida dos demais componentes), e recebe atualizações de métricas via `MonitoringService`. Essa estrutura, fundamentada no padrão de projeto MVVM, garante que o núcleo da simulação seja agnóstico à interface, permitindo que, futuramente, seja acoplado a outras tecnologias de UI (como uma aplicação web) com mínima refatoração.

A estrutura interna da camada core é composta por um conjunto de classes que colaboram para emular o comportamento de um sistema operacional. A Figura 3 apresenta um diagrama de classes simplificado com as principais entidades e suas relações.

Figura 3 – Diagrama de classes simplificado da camada core



Fonte: elaborado pelo autor (2025).

O fluxo de funcionamento do sistema, ilustrado na Figura 3, é orquestrado pelo `SimulationController`, que atua como ponto de entrada para a camada de simulação. Ao iniciar uma nova execução, ele instancia a classe `Kernel`, o componente principal do sistema operacional simulado. O `Kernel` é o responsável por criar e gerenciar as instâncias de `CPU` e `IODevice` (dispositivos de I/O). Cada `CPU`, por sua vez, executa as instruções de um processo, cujo estado completo, como identificador, prioridade e valores de registradores, é mantido em uma estrutura PCB. Eventos como a expiração de um *quantum* ou a conclusão de uma operação de I/O geram interrupções que são tratadas pelo `Kernel`, o qual pode invocar uma implementação da interface `IScheduler`, como o `PriorityScheduler`, para tomar uma decisão de escalonamento. O escalonador então seleciona o próximo processo a ser executado, e o `Dispatcher` (uma classe estática) realiza a troca de contexto. Todo este ciclo é observado pelo `MonitoringService`, que coleta e agrega os dados para a exibição na interface.

3.3 IMPLEMENTAÇÃO

Esta seção detalha o processo de implementação do simulador, descrevendo o ambiente de desenvolvimento, a arquitetura de software, o fluxo de dados da aplicação e os detalhes técnicos dos principais componentes que formam as camadas de apresentação e de simulação.

3.3.1 Ambiente e Ferramentas de Desenvolvimento

O projeto foi desenvolvido na plataforma .NET 9 utilizando a linguagem C# 12. A interface gráfica foi construída com o framework Windows Presentation Foundation (WPF), seguindo o padrão de projeto Model-View-ViewModel (MVVM) para garantir o desacoplamento da lógica de negócio. A visualização de dados em tempo real foi implementada com a biblioteca `LiveCharts2`. A ferramenta de desenvolvimento utilizada foi o Microsoft Visual Studio Enterprise 2022, e o controle de versão do código-fonte foi gerenciado com Git, com o repositório hospedado na plataforma GitHub.

3.3.2 Configuração e Validação da Simulação

Antes do início da execução, o simulador oferece ao usuário um ambiente de configuração completo e robusto, dotado de mecanismos de validação para garantir a coerência e a integridade dos dados de entrada.

3.3.2.1 Configuração da Máquina Virtual e Processos

A configuração da simulação é mediada pelas classes `ViewModel` da camada de apresentação, que interagem com o `SimulationController` (ver Figura 2)Figura 3. O `VmConfigViewModel` gerencia os parâmetros da `VM` (número de núcleos, *quantum*, dispositivos), enquanto o `ProcessConfigViewModel` detalha cada processo, incluindo sua lista de operações (`OperationConfigViewModel`). É nesta última camada que a flexibilidade da ferramenta se destaca, permitindo ao usuário definir o comportamento de cada operação.

A propriedade `RepeatCount`, por exemplo, é implementada no `SimulationController` através de um laço de repetição no momento da geração do programa, permitindo que uma única linha de configuração na UI se traduza em múltiplas instruções sequenciais. Da mesma forma, as opções de aleatoriedade para operações de CPU ou durações de I/O são resolvidas durante este passo de "compilação", utilizando a classe `Random` para sortear valores dentro dos parâmetros definidos pelo usuário.

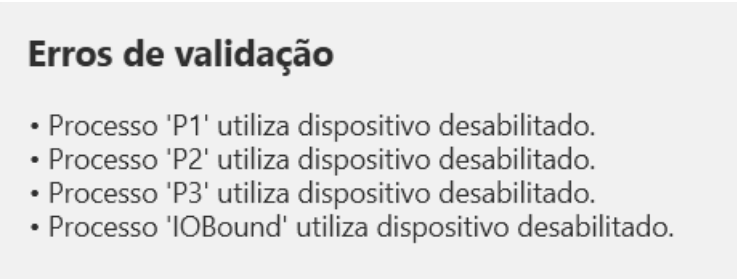
3.3.2.2 Validação de Dados

Para garantir a integridade da simulação, o sistema implementa rotinas de validação ao salvar a configuração de um processo e ao iniciar a execução da simulação. Os `ViewModels` contêm métodos de validação que verificam uma série de regras de negócio, como:

a) a obrigatoriedade e unicidade do nome de um processo;
b) a coerência dos parâmetros de uma operação. Por exemplo, em uma faixa de valores, Min deve ser menor que Max);
c) a obrigatoriedade de ao menos uma operação por processo.

A validação mais importante ocorre antes do início da simulação. O `SimulationControlViewModel` invoca uma rotina que realiza uma validação cruzada entre a configuração dos processos e a da VM. O sistema verifica, por exemplo, se algum processo está configurado para utilizar um dispositivo de I/O que foi desabilitado na configuração da máquina virtual. Caso qualquer inconsistência seja encontrada, a execução é abortada e uma janela de diálogo (`TextDialog`) é apresentada ao usuário com a lista consolidada de todos os erros, impedindo o início de uma simulação inválida, como ilustrado na Figura 4.

Figura 4 – Exemplo de validação ao iniciar a simulação

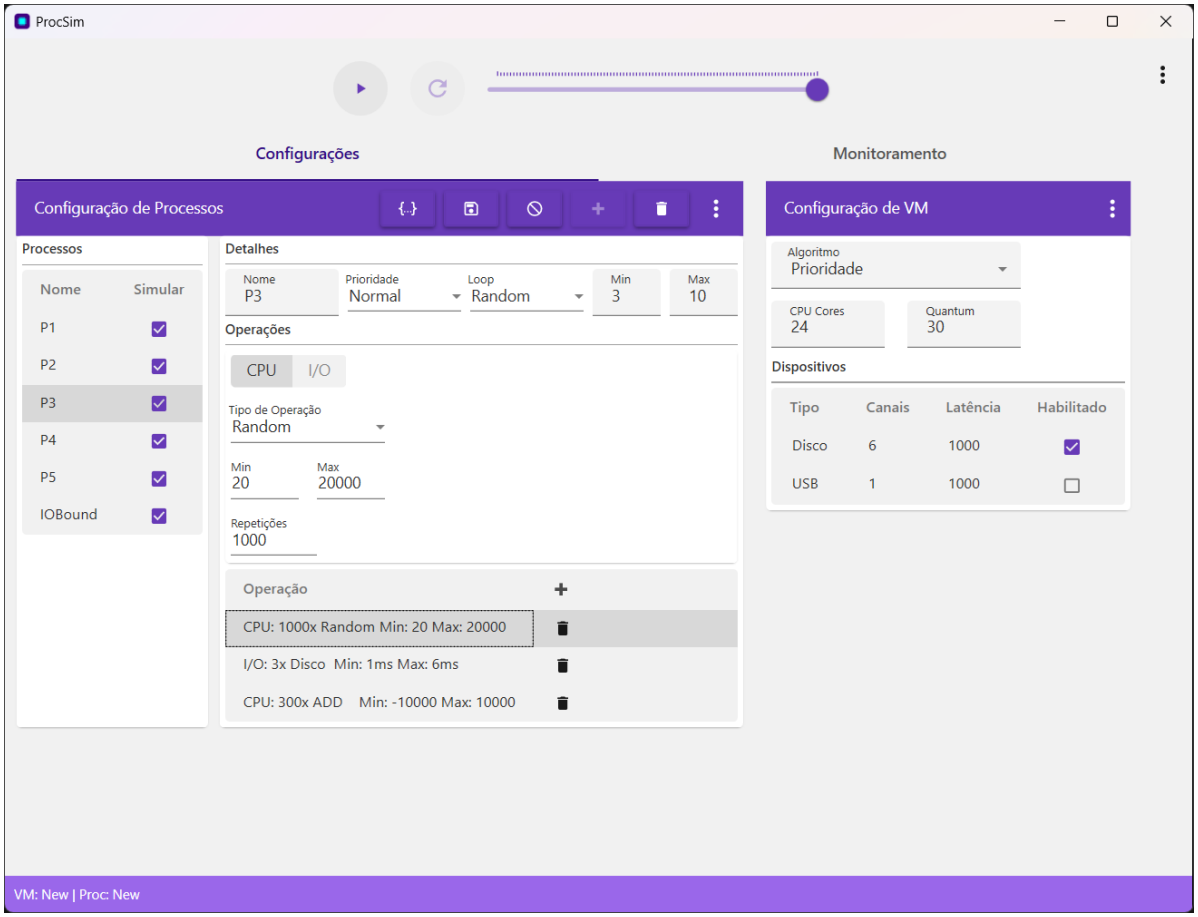


Fonte: elaborado pelo autor (2025).

3.3.3 O Motor de Simulação

Com as configurações validadas, o controle é passado para o motor de simulação, que gerencia o ciclo de vida e a execução da simulação. A Figura 5 apresenta a interface principal do simulador, onde o usuário interage com os controles de execução que comandam o motor, e os mecanismos centrais são detalhados a seguir.

Figura 5 – Tela Principal de Configuração



Fonte: elaborado pelo autor (2025).

3.3.3.1 Controle de Execução e o Ciclo de *Tick*

A interface do provê os controles de Iniciar, Pausar e Reiniciar a simulação, além de um controle deslizante (*slider*) para ajustar a velocidade de execução. A implementação dessa funcionalidade é gerenciada pelo `SimulationControlViewModel`, que atua como um autômato, alterando o estado do `SimulationController` (`Created`, `Running`, `Paused`).

O controle de velocidade é implementado no `SimulationController` através de um `PeriodicTimer`. A propriedade `Clock`, vinculada ao *slider* da UI, define o período (`Period`) deste timer. Este controle permite ao usuário definir o intervalo entre cada *tick* da simulação, variando de um mínimo de 10 ms a um máximo de 1000 ms, com ajustes em incrementos de 10 ms. Um laço assíncrono (`TimerWorker`) aguarda a conclusão de cada intervalo com `_timer.WaitForNextTickAsync`. Ao final de cada período, o *delegate* `_clockTick` é invocado, o que por sua vez dispara o método `Tick` em todas as instâncias de CPU, regendo assim o ritmo de toda a simulação.

Este mecanismo, embora implementado em software, emula funcionalmente o papel do hardware de temporização de um sistema computacional real, como o PIT. Assim como o hardware gera uma interrupção para devolver o controle ao kernel, o `PeriodicTimer` do ProcSim dispara o evento que representa o *tick* do sistema. É este *tick* simulado que permite a implementação da preempção, a contagem do *quantum* no escalonador e o avanço síncrono do tempo em todo o sistema, conectando diretamente a implementação do software aos princípios de hardware.

3.3.3.2 Geração de Programa e Controle de Fluxo

Um dos diferenciais da ferramenta é a sua capacidade de traduzir uma configuração de alto nível, definida pelo usuário, em um programa executável pela CPU simulada. Quando o usuário define uma operação, como uma soma ou uma requisição de I/O, a `InstructionFactory` é utilizada para converter essa abstração em uma ou mais instâncias da classe `Instruction`. Cada `Instruction`, por sua vez, é composta por uma sequência de `MicroOp`, que representam as operações atômicas e indivisíveis que a CPU executa. Essa abordagem garante que operações de diferentes naturezas tenham custos computacionais distintos e realistas. O Quadro 4 demonstra como uma operação simples, definida pelo usuário, é decomposta em instruções e micro-operações.

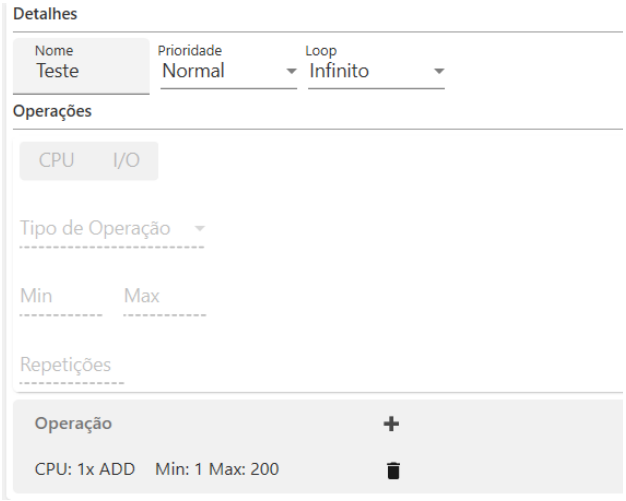
Quadro 4 – Exemplo de Decomposição de uma Operação de CPU

Configuração na Interface	Tipo: CPU Operação: ADD Repetições: 1
Instrução Gerada (Mnemonic)	add r0, {r1}, {r2}
Micro-operações (Name e Description)	SET_R1_{r1}: R1 ← {r1} SET_R2_{r2}: R2 ← {r2} +_R0_R1_R2: R0 ← R1({r1}) + R2({r2})

Fonte: elaborado pelo autor (2025).

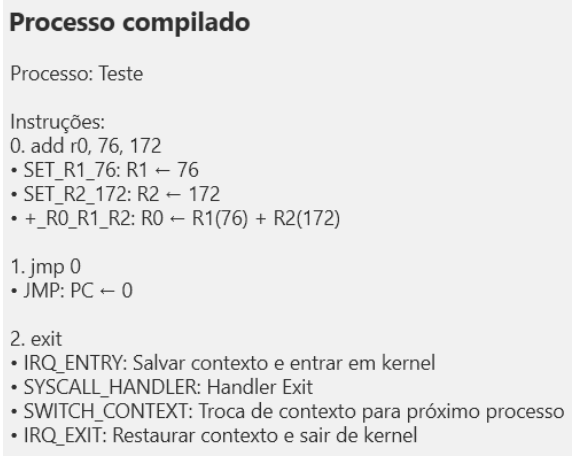
Para tornar esse processo de tradução transparente ao usuário, a interface oferece um botão de "*Build*" ({}) para cada processo configurado. Ao ser acionado, ele exibe uma janela com o programa compilado, mostrando exatamente como as operações de alto nível são decompostas em instruções e micro-operações. A Figura 6 ilustra um processo de exemplo, enquanto a Figura 7 exibe o resultado de sua compilação interna.

Figura 6 – Exemplo de Configuração de um Processo na UI



Fonte: elaborado pelo autor (2025).

Figura 7 – Visualização do Processo Compilado em Instruções



Fonte: elaborado pelo autor (2025).

3.3.3.3 O Ciclo de Execução da CPU

A alta fidelidade da simulação é garantida pelo método `Tick` na classe `CPU`. A cada pulso do relógio do sistema (relógio periódico na `SimulationController`), o método processa uma única `MicroOp` da fila de micro-operações da instrução atual, emulando o ciclo de busca e execução de um processador real. O

Quadro 5 detalha a lógica algorítmica deste ciclo.

Quadro 5 – Algoritmo do Ciclo de Execução da CPU

Passo	Ação
1	Início do Ciclo (*Tick*): O contador de ciclos da CPU (`CycleCount`) é incrementado.
2	Verificação da Fila de Micro-operações: O sistema verifica se a fila de micro-operações (`_ops`) da instrução atual não está vazia. Se sim, a `MicroOp` é executada e o ciclo encerra.
3	Verificação de Interrupção: Se a fila de `_ops` está vazia, o `InterruptController` é consultado. Se houver uma interrupção, a `ISR` correspondente é construída e suas `MicroOps` são enfileiradas.
4	Busca de Nova Instrução: Se não há interrupção, o sistema busca a próxima `Instruction` do processo, incrementa o Contador de Programa (`PC`) e enfileira as `MicroOps` da nova instrução.

Fonte: elaborado pelo autor (2025).

3.3.3.4 Implementação do Escalonador Híbrido

O componente mais complexo do gerenciamento de processos no simulador é o `PriorityScheduler`, que ajusta dinamicamente a prioridade dos processos. Sua lógica reside no método `RecalculatePriority`, apresentado no Quadro 6.

Quadro 6 – Cálculo de Prioridade Dinâmica no `PriorityScheduler`

```
private int RecalculatePriority(PCB pcb, ulong now)
{
    // 1. Heurística de Envelhecimento (Aging)
    ulong wait = now - pcb.LastEnqueueCycle;
    double boost = BETA * (wait / (double)AGING_QUANTUM);

    // 2. Heurística de Perfil de Uso (I/O-Bound vs CPU-Bound)
    double cpuTime = pcb.UserCycles + pcb.SyscallCycles;
    double ioTime = pcb.WaitCycles;
    double total = cpuTime + ioTime + 1;
    double ioCpu = ALPHA * ((ioTime / total) - (cpuTime / total));

    // 3. Cálculo da Prioridade Bruta
    int meanStatic = (int)pcb.StaticPriority;
    int minStatic = meanStatic - 4;
    int maxStatic = meanStatic + 4;
    double raw = meanStatic - boost + ioCpu;

    // 4. Cálculo de penalidade de fila
    int previousPriority = pcb.DynamicPriority;
    int higherCount = _readyQueue.UnorderedItems
                      .Count(item => item.Priority >= previousPriority);

    double queuePenalty = GAMMA * higherCount;
    raw += queuePenalty;

    // 5. Normalização
    int dynamicPriority = (int)Math.Round(raw);
    dynamicPriority = Math.Clamp(dynamicPriority, minStatic, maxStatic);
}
```

Fonte: elaborado pelo autor (2025).

3.3.3.4.1 Heurística de *Aging*

A primeira parte do método implementa o *aging* como uma solução direta para o problema de inanição. Conforme a parte 1 do Quadro 6, a variável `wait` armazena quantos ciclos de simulação o processo está aguardando na fila de prontos. Esse valor, ponderado pela constante `BETA`, é usado para calcular um *boost* que efetivamente aumenta a prioridade do processo quanto mais tempo ele espera, uma técnica clássica para garantir a equidade do escalonador, assegurando que processos de baixa prioridade não sejam indefinidamente ignorados (SILBERSCHATZ; GALVIN; GAGNE, 2015).

3.3.3.4.2 Balanço I/O vs. CPU

A segunda heurística analisa o perfil de uso do processo para melhorar a responsividade. Conforme a parte 2 do Quadro 6, o fator `ioCpu` calcula a diferença proporcional entre o tempo gasto em I/O (`ioTime`) e o tempo consumido pela CPU (`cpuTime`). Processos limitados por I/O (*I/O-bound*), como aplicações interativas, recebem um aumento em sua prioridade, ponderado pela constante `ALPHA`. Esta estratégia é diretamente inspirada em sistemas operacionais comerciais, como o Windows, que concede um bônus de prioridade a *threads* que saem de um estado de espera, para que possam ser executadas rapidamente e oferecer uma resposta mais ágil ao usuário (RUSSINOVICH; SOLOMON; IONESCU, 2012).

3.3.3.4.3 Heurística de Penalidade de Fila

Esta heurística, ponderada pela constante `GAMMA`, ajusta a prioridade com base na contenção da fila de prontos. A parte 4 do Quadro 6 mostra essa implementação, onde o sistema verifica a quantidade de processos na fila (`higherCount`) com prioridade menor (onde um valor numérico maior representa uma prioridade menor). Ao entrar em uma fila já populada com processos de prioridade mais baixa, o novo processo recebe uma leve penalidade. Este mecanismo atua como um regulador de contenção, ajudando a garantir que processos de baixa prioridade não fiquem indefinidamente adiados por um fluxo constante de processos de prioridade ligeiramente superior.

3.3.3.4.4 Normalização e Vínculo à Prioridade Estática

Após a aplicação de todas as heurísticas, o valor final da prioridade dinâmica é restringido (*clamped*) a uma faixa pré-definida em torno da prioridade estática original do processo, conforme a parte 5 do Quadro 6. Este passo é crucial para a estabilidade do sistema, pois garante que, apesar dos ajustes dinâmicos, um processo de baixa prioridade não possa ser impulsionado a um nível de tempo real, e vice-versa. Essa abordagem espelha o conceito de classes de prioridade encontrado em sistemas operacionais reais, onde os ajustes dinâmicos geralmente ocorrem dentro dos limites da classe de prioridade original do processo, mantendo um grau de previsibilidade (TANENBAUM; WOODHULL, 2008).

3.3.3.5 Gerenciamento de I/O e Interrupções

O tratamento de operações de I/O é um processo orientado a eventos que demonstra a interação entre múltiplos componentes da camada core. O ciclo de vida de uma requisição de I/O ocorre da seguinte forma:

1. Um processo em execução invoca uma `Syscall` para uma operação de I/O.
2. O `SystemCallDispatcher` recebe a chamada e encaminha a requisição para o `IODevice` apropriado.
3. O `PCB` do processo tem seu estado alterado para `Waiting` (Bloqueado) e o processo é removido da CPU.
4. Uma troca de contexto é forçada, e o `Scheduler` é invocado para selecionar um novo processo para a `CPU`.
5. O `IODevice` processa a requisição (simulando o tempo de latência) e, ao concluir, dispara o evento `IORequestCompleted`.
6. O `InterruptController` captura este evento e notifica o Kernel, que utiliza o `IoInterruptHandler` para tratar da interrupção. A rotina de tratamento da interrupção move o processo do estado `Waiting` de volta para a fila de prontos do `Scheduler`, tornando-o novamente elegível para execução.

3.3.4 Interface de Usuário e Monitoramento

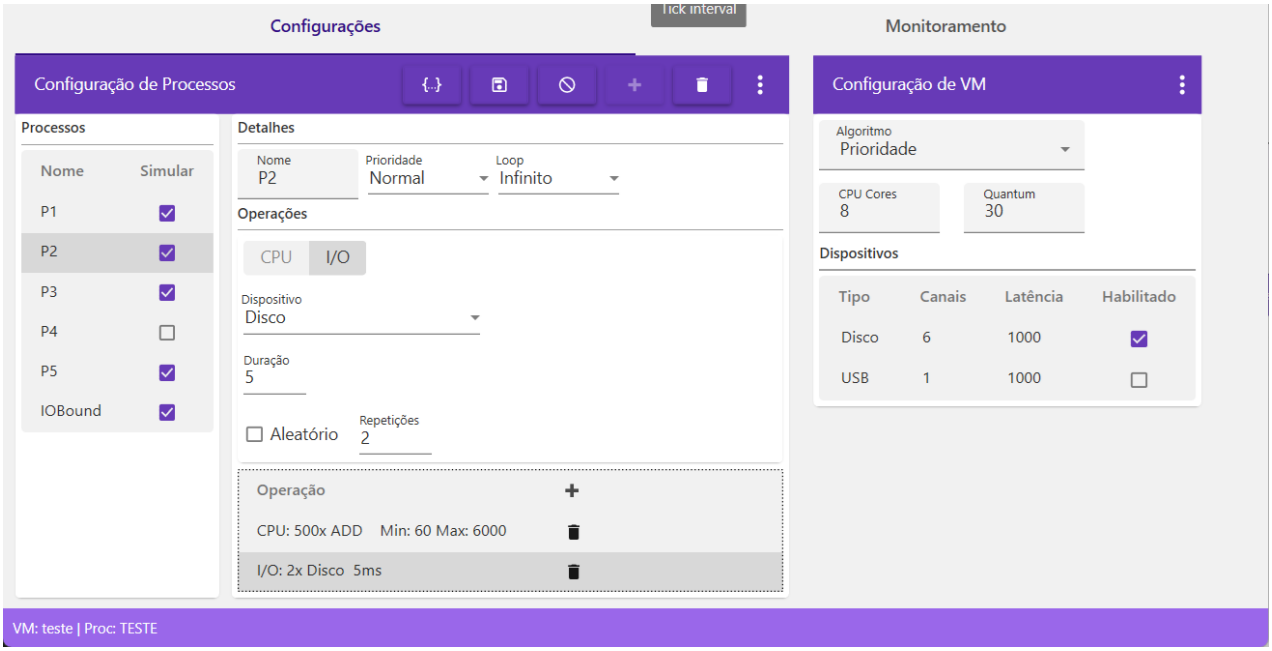
A camada de apresentação é responsável por fornecer ao usuário uma interface clara para configurar e visualizar a simulação.

3.3.4.1 A Interface de Usuário (WPF e MVVM)

A camada de apresentação foi desenvolvida com o objetivo de ser funcional e intuitiva, tendo sua estrutura e design inspirados diretamente no Gerenciador de Tarefas do Windows. A aplicação é organizada em duas áreas principais, "Configurações" e "Monitoramento", permitindo que o usuário primeiramente modele o cenário experimental e, em seguida, observe os resultados em tempo real.

A área de Configurações, ilustrada na Figura 8, é o ponto de partida onde o usuário define todos os parâmetros da simulação. Ela contém painéis para configurar a máquina virtual, como o número de núcleos e o algoritmo de escalonamento, e para criar e gerenciar a lista de processos, detalhando suas operações e prioridades. É possível selecionar quais processos serão incluídos na simulação e, com a execução em andamento, adicionar novas instâncias de um processo configurado, permitindo a análise de sistemas com carga de trabalho dinâmica.

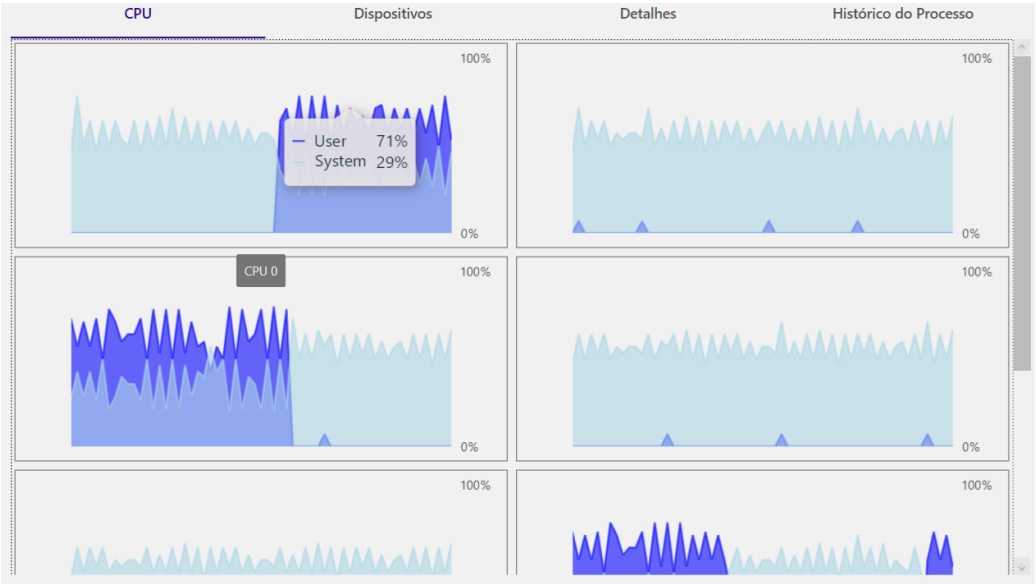
Figura 8 – Tela Principal de Configuração



Fonte: elaborado pelo autor (2025).

A área de Monitoramento é dividida em um conjunto de abas que permitem a observação detalhada da simulação. A Figura 9 exibe a aba "CPU", que apresenta gráficos de desempenho para cada núcleo e uma visão agregada. Esta tela permite a análise da distribuição de carga em um ambiente *multicore* e a visualização do *overhead* do sistema, separando o tempo de execução gasto em modo de usuário (código da aplicação) do tempo gasto em modo de sistema (interrupções e *syscalls*).

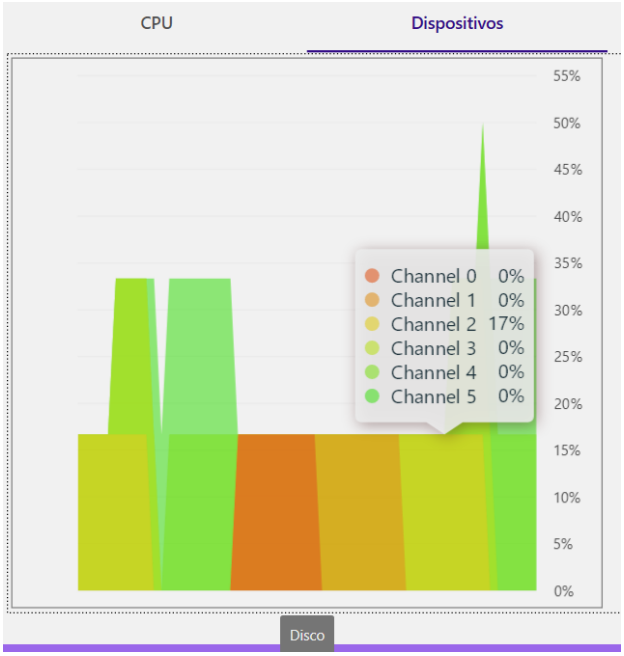
Figura 9 – Aba de Monitoramento de Desempenho de CPU



Fonte: elaborado pelo autor (2025).

A Figura 10 demonstra a aba "Dispositivos", que detalha o uso de cada dispositivo de I/O habilitado. Os gráficos exibem a utilização por canal, tornando visível a contenção de recursos de I/O e permitindo ao estudante observar como múltiplas requisições são enfileiradas e processadas.

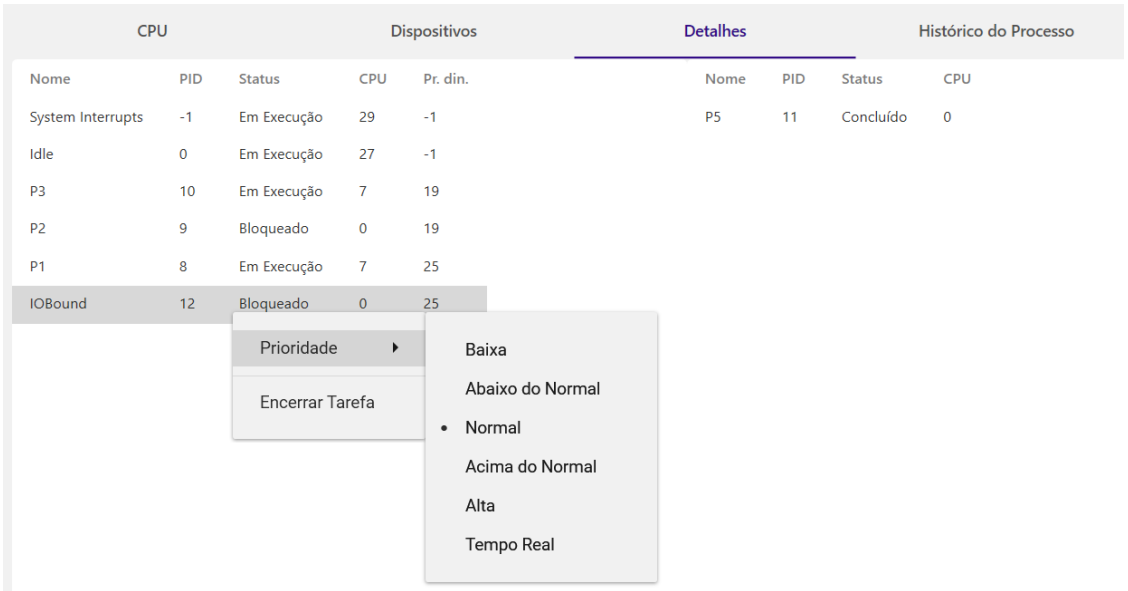
Figura 10 – Aba de Monitoramento de Desempenho de Dispositivos



Fonte: elaborado pelo autor (2025).

A aba "Detalhes", apresentada na Figura 11, funciona como o principal ponto de interação dinâmica do usuário com o sistema em execução. Nela, é possível observar o estado atual de todos os processos (PID, status, uso de CPU, prioridade dinâmica) e intervir diretamente na simulação, alterando a prioridade estática de um processo ou solicitando seu encerramento.

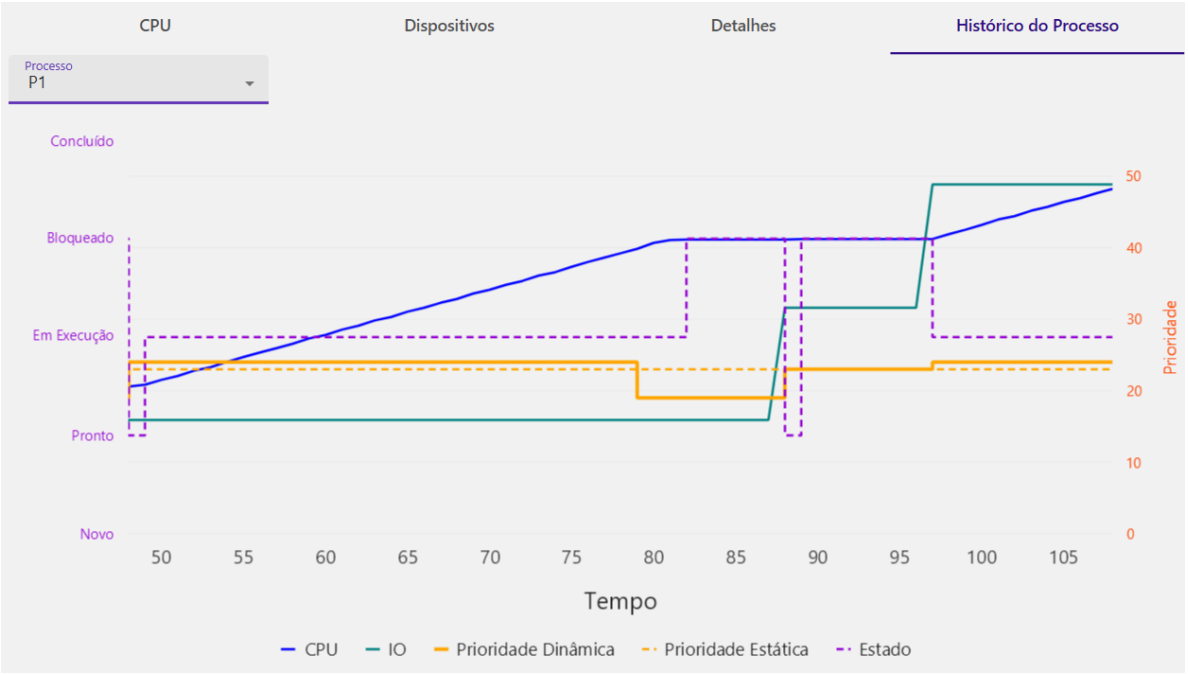
Figura 11 – Aba de Detalhes com Lista de Processos



Fonte: elaborado pelo autor (2025).

Por fim, a Figura 12 ilustra a aba "Histórico do Processo". Esta é uma poderosa ferramenta analítica que permite ao usuário selecionar um processo específico e visualizar um gráfico detalhado que correlaciona o seu uso de CPU, tempo em I/O, e as variações de prioridade dinâmica e estática ao longo do tempo. Esta visualização torna explícitas as decisões tomadas pelo escalonador, como a aplicação de um *boost* de prioridade após uma operação de I/O.

Figura 12 – Aba de Histórico do Processo



Fonte: elaborado pelo autor (2025).

3.3.4.2 O Sistema de Monitoramento por Eventos

O monitoramento em tempo real é realizado por uma arquitetura orientada a eventos, garantindo que a coleta de dados não interfira no desempenho da simulação. O `MonitoringService` é o componente central desta arquitetura. Ele se inscreve em eventos publicados por componentes da camada core. Por exemplo, ao inicializar, ele se registra nos eventos `IORequestStarted` e `IORequestCompleted` de cada `IODevice`. Quando um dispositivo inicia ou termina uma operação de I/O, ele dispara o evento correspondente, e o `MonitoringService` é notificado em tempo real, o permitindo calcular métricas precisas de uso e latência. O Quadro 7 demonstra como o serviço se registra para ser notificado sobre a conclusão de uma operação de I/O.

Quadro 7 – Subscrição de Evento no Serviço de Monitoramento

```
// Dentro do MonitoringService, ao inicializar um dispositivo
private void SubscribeDevice(IODevice device)
{
    // Registra o método OnIoRequestCompleted para ser chamado
    // quando o evento IORequestCompleted do dispositivo for disparado.
    device.IORequestCompleted += OnIoRequestCompleted;
}

// Método que será executado quando o evento ocorrer
private void OnIoRequestCompleted(IoRequestNotification req)
{
    // Lógica para calcular latência e atualizar métricas de I/O...
}
```

Fonte: elaborado pelo autor (2025).

Após coletar e agregar os dados, o `MonitoringService` dispara seu próprio evento, `OnMetricsUpdated`. Os `ViewModels` da camada de apresentação, por sua vez, são inscritos neste evento. Ao recebê-lo, eles atualizam suas coleções de dados, que estão vinculadas aos gráficos da biblioteca `LiveCharts` na interface, fazendo com que a visualização seja atualizada de forma reativa e eficiente. Essa coleta, inspirada no gerenciador de tarefas do Windows, ocorre a cada 1 segundo e independe da velocidade da simulação (*ticks* do sistema).

3.3.5 Persistência de Configurações

Para garantir a reprodutibilidade dos experimentos, a aplicação implementa um sistema de gerenciamento de configurações. A arquitetura utiliza o padrão de repositório, com as classes `VmConfigRepository` e `ProcessesConfigRepository`, para abstrair a lógica de persistência. Essas classes são responsáveis pela serialização dos objetos de configuração para o formato JSON e pelo salvamento em arquivos com extensões customizadas (.vmconfig e .pconfig).

Adicionalmente, para conveniência do usuário, a configuração ativa é salva de forma implícita quando a aplicação é fechada. Este mecanismo é implementado no manipulador de eventos `OnClosing` da classe `MainView`. O evento invoca os métodos de salvamento nos `ViewModels` (`VmConfigViewModel` e `ProcessesConfigViewModel`), que por sua vez utilizam o padrão de repositório (*repository pattern*) para serializar o estado atual da configuração para o formato JSON. O resultado é então persistido nas configurações de usuário do Windows por meio da classe `Settings.Default`, garantindo que o mesmo ambiente de trabalho seja restaurado na próxima inicialização da ferramenta.

4 RESULTADOS

Esta seção apresenta os resultados obtidos com a execução do ProcSim, validando as principais funcionalidades implementadas através de cenários de teste práticos. O objetivo é demonstrar que o simulador se comporta conforme a teoria de sistemas operacionais e que é uma ferramenta eficaz para ilustrar conceitos complexos. Os arquivos de configuração para os cenários descritos encontram-se no Apêndice A.

4.1 CENÁRIO 1: VALIDAÇÃO DO CICLO DE VIDA E BLOQUEIO POR I/O

O primeiro cenário de teste valida a funcionalidade mais fundamental do simulador, que é a capacidade de executar um processo e gerenciar corretamente a transição de seus estados, especialmente o bloqueio por operações de I/O.

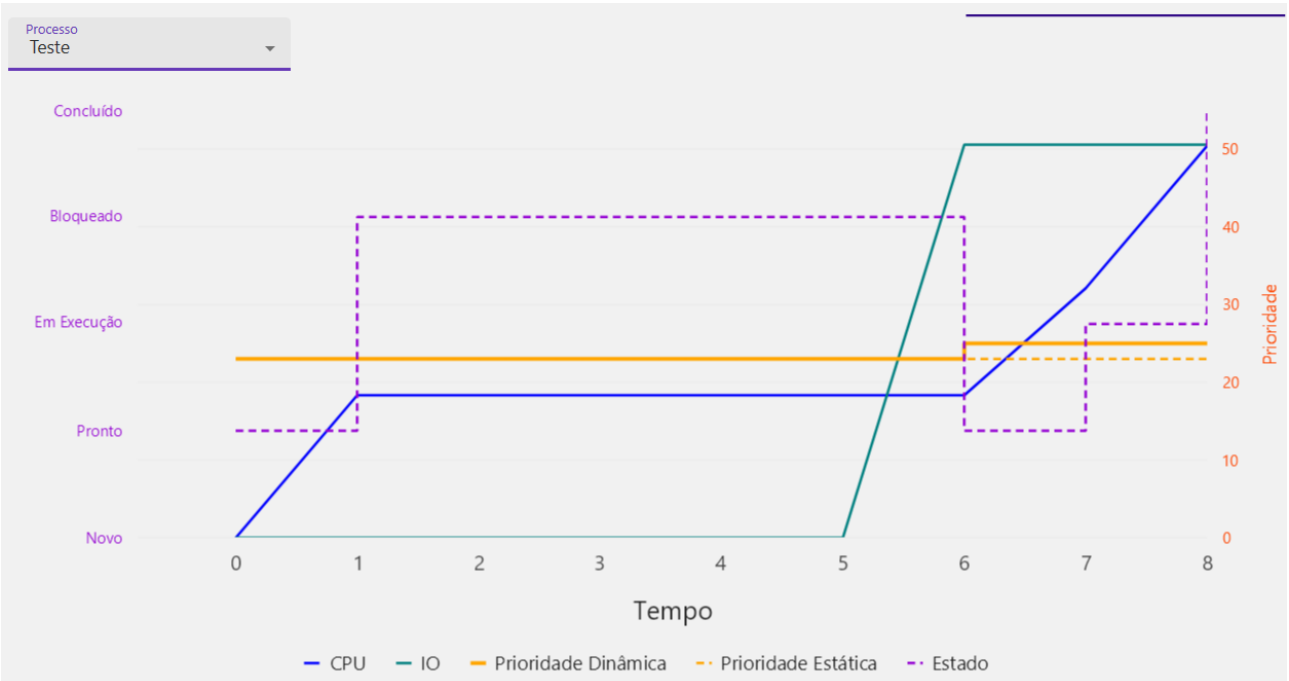
Quadro 8 – Teste de Ciclo de Vida do Processo com Bloqueio de I/O

Objetivo	Validar a correta transição de estados de um processo, incluindo o bloqueio por uma operação de I/O.
Configuração do Cenário	Um único processo é configurado com uma sequência de operações: uma operação de CPU, seguida por uma operação de I/O (leitura de disco) e, por fim, outra operação de CPU.
Método de Verificação	Executar a simulação e observar o estado do processo na aba "Detalhes" e no gráfico da aba "Histórico do Processo".
Resultado Observado	O processo transitou corretamente pelos estados “Em execução” > “Bloqueado” > “Pronto” > “Concluído”, validando o mecanismo de bloqueio e a liberação da CP, conforme evidenciado na Figura 13.

Fonte: elaborado pelo autor (2025).

O resultado do teste confirma que o simulador modela corretamente o ciclo de vida de um processo. A capacidade de visualizar essa transição de estados ao longo do tempo, apresentada na Figura 13, torna o conceito, que é puramente teórico em sala de aula, em um evento concreto e observável. No gráfico, a linha tracejada roxa mostra o processo passando do estado "Pronto" para "Bloqueado" ao iniciar a operação de I/O, e retornando para "Pronto" após sua conclusão, antes de ser escalonado novamente para executar sua instrução final.

Figura 13 – Validação do Estado de Bloqueio por I/O



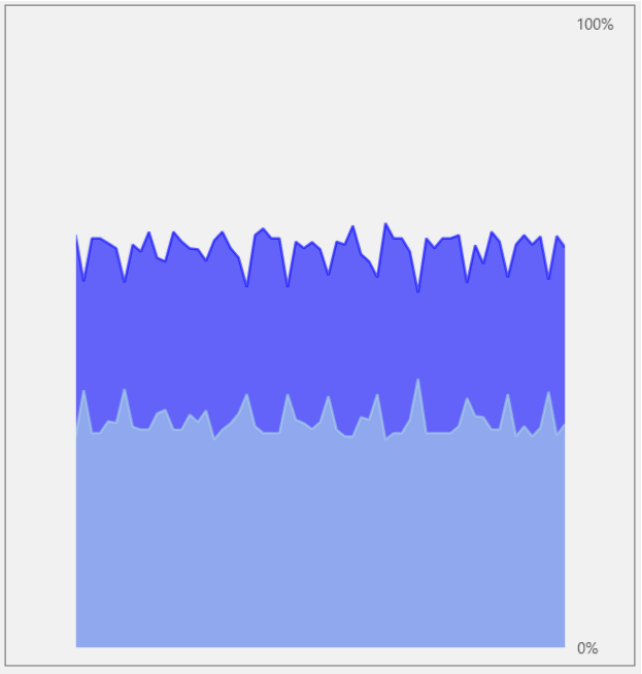
Fonte: elaborado pelo autor (2025).

4.2 CENÁRIO 2: ANÁLISE COMPARATIVA DOS ESCALONADORES

Este cenário demonstra as diferenças comportamentais entre os algoritmos de escalonamento implementados. Um mesmo conjunto de processos, um CPU-bound e um I/O-bound com a mesma prioridade estática, foi executado sob cada um dos escalonadores em uma configuração de CPU com apenas um núcleo.

Sob o escalonador Round Robin, a execução resultou em um padrão de uso de CPU intercalado, onde ambos os processos receberam fatias de tempo de forma alternada, demonstrando a equidade do algoritmo na distribuição de recursos, como pode ser visto na Figura 14.

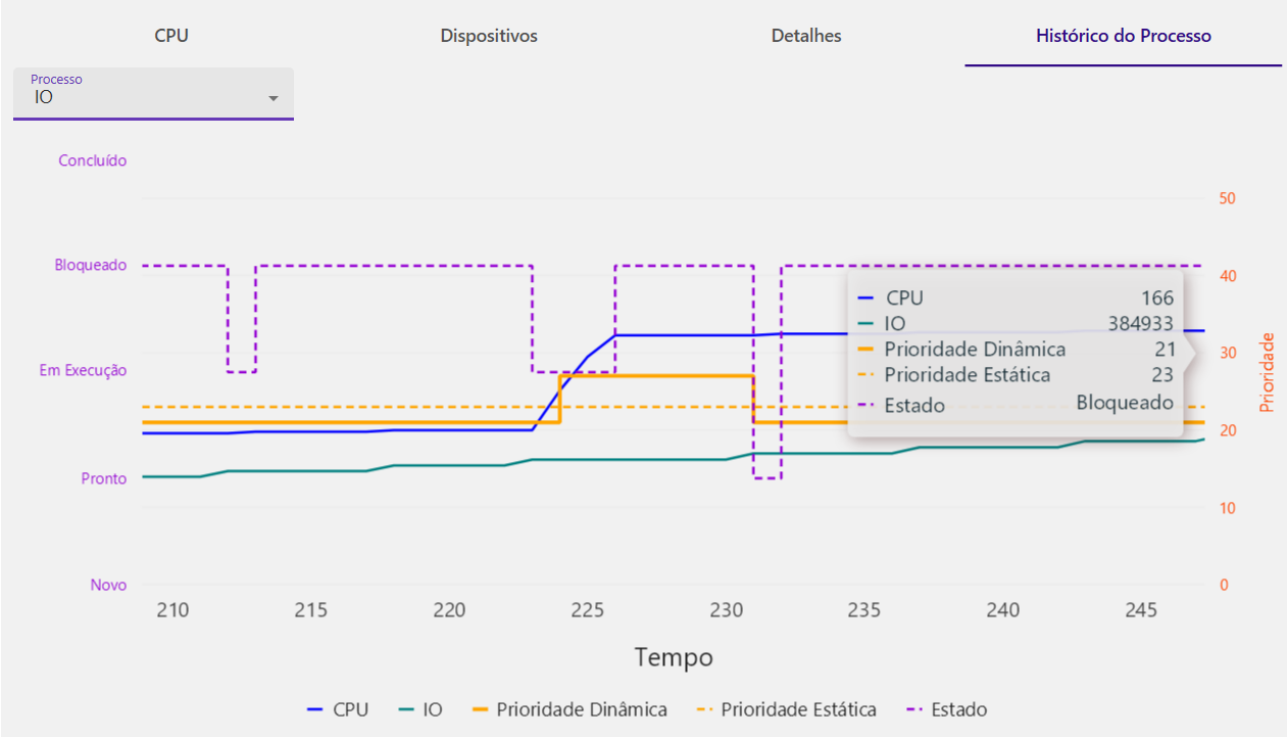
Figura 14 – Execução Intercalada no Escalonador Round Robin



Fonte: elaborado pelo autor (2025).

Em contraste, ao utilizar o escalonador por Prioridades Híbrido, o comportamento do sistema muda para otimizar a responsividade. A análise do histórico do processo *I/O-bound*, apresentada na Figura 15, torna essa otimização explícita. Observa-se que, imediatamente após o processo sair do estado "Bloqueado", sua "Prioridade Dinâmica" (linha laranja contínua) sofre um aumento significativo, ultrapassando a sua "Prioridade Estática" (linha tracejada amarela). Esse *boost* garante que o processo interativo ganhe acesso à CPU rapidamente, validando a implementação da heurística. Com o tempo, à medida que o processo consome ciclos de CPU, sua prioridade dinâmica decai gradualmente, demonstrando o equilíbrio dinâmico do escalonador.

Figura 15 – Análise do *Boost* de Prioridade na Aba "Histórico do Processo"



Fonte: elaborado pelo autor (2025).

4.3 CENÁRIO 3: VALIDAÇÃO DA PERSISTÊNCIA E REPRODUTIBILIDADE

O último cenário valida a funcionalidade de salvar e carregar as configurações da simulação, um requisito essencial para a reprodutibilidade de experimentos em um contexto acadêmico.

Quadro 9 – Teste de Persistência de Configuração

Objetivo	Validar a capacidade do sistema de salvar e carregar uma configuração de simulação completa.
Configuração do Cenário	Uma configuração de VM e um conjunto de processos complexos são criados na interface.
Método de Verificação	1. A configuração é salva em arquivos (.vmconfig e .pconfig) usando a função "Salvar". 2. O simulador é reiniciado para limpar o estado da memória. 3. Os arquivos de configuração salvos são carregados usando a função "Carregar". 4. A configuração exibida na interface é comparada com a original.
Resultado Observado	O sistema restaurou completamente o estado da configuração, com todos os parâmetros da máquina virtual e todos os processos, incluindo suas operações detalhadas e prioridades, confirmando o sucesso da funcionalidade.

Fonte: elaborado pelo autor (2025).

Este teste confirma que os cenários de simulação podem ser facilmente salvos, compartilhados e recarregados, garantindo que estudantes e professores possam trabalhar sobre configurações idênticas, um requisito fundamental para atividades em laboratório.

5 CONCLUSÕES

O objetivo de desenvolver um simulador visual e pedagógico de gerenciamento de processos e escalonamento em sistemas operacionais foi atendido com sucesso. A ferramenta ProcSim foi projetada e implementada para superar as lacunas de fidelidade de simulação e de capacidade de monitoramento identificadas na análise dos trabalhos correlatos. Os objetivos específicos traçados para este trabalho foram cumpridos integralmente, conforme detalhado a seguir:

a) os algoritmos de escalonamento Round Robin e Prioridades Híbrido foram implementados com sucesso, este último incorporando heurísticas de *aging* e *boost* de I/O, cuja eficácia foi validada no Cenário 2 da seção de Resultados;
b) a interface gráfica interativa em WPF, utilizando o padrão MVVM e inspirada no Gerenciador de Tarefas

do Windows, mostrou-se eficaz para o monitoramento detalhado e interação em tempo real, como demonstrado nas figuras do item 3.3.4.1A Interface de Usuário (WPF e MVVM);

c) a simulação detalhada do ciclo de vida dos processos, incluindo a estrutura de PCBs e o tratamento de operações de I/O, foi garantida pela execução de micro-operações, validada no Cenário 1;
d) A arquitetura modular em camadas, que separa o núcleo de simulação (`ProcSim.Core`) da interface (`ProcSim`), foi estabelecida para garantir a manutenibilidade e a extensibilidade da ferramenta, conforme descrito no item 3.2.

Em comparação com os trabalhos correlatos, o ProcSim avança ao se aprofundar nas complexas heurísticas de escalonadores modernos, um tópico frequentemente simplificado em ferramentas educacionais como as analisadas por Cruz et al. (2023). A sua principal contribuição técnica reside na implementação de um motor de simulação de alta fidelidade, baseado na execução de micro-operações, e de uma suíte de monitoramento visual em tempo real que não é encontrada de forma integrada nas soluções existentes. Esses elementos, em conjunto, permitem uma análise do comportamento do sistema mais fiel à encontrada em sistemas operacionais comerciais.

A contribuição fundamental deste trabalho, contudo, é pedagógica. Ao oferecer a estudantes e professores um laboratório virtual onde conceitos teóricos abstratos, como a preempção e o dilema entre responsividade e vazão, se tornam eventos visuais, interativos e mensuráveis, o ProcSim atua como uma ponte eficaz entre a teoria e a prática.

5.1 LIMITAÇÕES E MELHORIAS

Apesar de atingir seus objetivos, o projeto apresenta limitações que abrem caminhos para futuras melhorias. A principal limitação técnica reside na dependência da plataforma Windows, imposta pelo uso do framework WPF para a interface gráfica. Adicionalmente, o escopo da simulação, embora profundo no gerenciamento de processos, não abrange outras áreas essenciais de um SO, como o gerenciamento de memória. Por fim, é importante ressaltar que, embora tecnicamente validado através dos cenários de teste, o trabalho não incluiu uma validação pedagógica formal com estudantes para mensurar o impacto da ferramenta no processo de aprendizagem.

Com base nisso, algumas melhorias podem ser sugeridas para aprimorar a versão atual da ferramenta. Na área de visualização e depuração, seria de grande valia implementar um monitor para as filas de espera de cada dispositivo de I/O e um painel que exiba a lista de instruções de um processo, destacando a que está sendo executada. Do ponto de vista técnico, otimizações de desempenho, como a coleta de métricas sob demanda apenas para os gráficos visíveis, representam melhorias que aumentariam a eficiência da ferramenta.

5.2 EXTENSÕES

A arquitetura modular do ProcSim foi concebida para ser um "motor base", o que permite que trabalhos futuros explorem diversas frentes de expansão para além do escopo atual. A principal proposta de extensão seria no próprio núcleo funcional, com a adição de novos módulos de simulação, como uma Unidade de Gerenciamento de Memória (*Memory Management Unity* - MMU) para explorar algoritmos de alocação e substituição de página (como FIFO ou LRU).

Adicionalmente, a estrutura do escalonador foi projetada para facilitar a implementação de outros algoritmos clássicos e avançados, como Trabalho Mais Curto Primeiro (*Shortest Job First* - SJF) ou Filas de Múltiplos Níveis com Retroalimentação (*Multilevel Feedback Queues*), enriquecendo ainda mais o potencial pedagógico da aplicação. Por fim, a migração da interface para um framework multiplataforma como .NET MAUI ou desenvolvimento de uma interface web aumentaria significativamente o alcance e a acessibilidade do simulador.

Com o intuito de fomentar a colaboração e a evolução contínua da ferramenta, o projeto foi disponibilizado publicamente no repositório GitHub sob uma licença de código aberto. O objetivo é que a comunidade acadêmica e de desenvolvedores possa não apenas utilizar o ProcSim, mas também contribuir com novas funcionalidades, melhorias e extensões, garantindo a longevidade e o aprimoramento do simulador. O código-fonte pode ser acessado em: https://github.com/casuffitsharp/procsim.

REFERÊNCIAS

COSTA, Antonio V.; SILVA, Arthur; FERNANDES, Silvio; MACEDO, Francisco Tailanio de. **MOSS: Uma ferramenta para o auxílio do ensino de sistemas operacionais**. Anais do XXIX Simpósio Brasileiro de Informática na Educação (SBIE 2018), VII Congresso Brasileiro de Informática na Educação (CBIE 2018), p. 755-764, 2018

CRUZ, Adilson Oliveira; JESUS, Átila Alves de; MARQUES FILHO, Fabio Pio. **Um estudo comparativo de simuladores de rotinas de sistemas operacionais para auxílio às aulas teóricas**. Revista Observatorio de la Economia Latinoamericana, Curitiba, v.21, n.1, p.497-507, 2023.

DOWNEY, Allen B. **The Little Book of Semaphores**. 2. ed. [S.l.]: Green Tea Press, 2008.

HERRINGTON, Jan; OLIVER, Ron. **An instructional design framework for authentic learning environments**. Educational Technology Research and Development, v. 48, n. 3, p. 23-48, 2000.

LOVE, Robert. **Linux Kernel Development**. 3. ed. Upper Saddle River, NJ: Addison-Wesley, 2010.

RUSSINOVICH, Mark E.; SOLOMON, David A.; IONESCU, Alex. **Windows Internals, Part 1**. 6. ed. Redmond, WA: Microsoft Press, 2012.

SCAMATI, Vagner. **Um simulador para o ensino de sistemas operacionais com a tecnologia da realidade virtual (ESORV)**. 2017. Dissertação (Mestrado em Ciência da Computação) - Faculdade Campo Limpo Paulista, Campo Limpo Paulista, 2017.

SILBERSCHATZ, Abraham; GALVIN, Peter B.; GAGNE, Greg. **Fundamentos de Sistemas Operacionais**. 9. ed. Rio de Janeiro: Grupo GEN, 2015.

SMITH, Josh. **WPF Apps With The Model-View-ViewModel Design Pattern.** *MSDN Magazine*, fev. 2009. Disponível em: https://learn.microsoft.com/en-us/archive/msdn-magazine/2009/february/patterns-wpf-apps-with-the-model-view-viewmodel-design-pattern. Acesso em: 08 jul. 2025.

TANENBAUM, Andrew S.; WOODHULL, Albert S. **Sistemas Operacionais: Projeto e Implementação**. 3. ed. Porto Alegre: Bookman, 2008.

APÊNDICE A – ARQUIVOS DE CONFIGURAÇÃO DO CENÁRIO DE TESTE

Este apêndice detalha os arquivos de configuração em formato JSON utilizados nos cenários de teste para coleta dos resultados.

Quadro 10 – Conteúdo do Arquivo de Configuração da VM, cenário 1 (cenario1.vmconfig)

```
{
  "devices": [
    {
      "type": "Disk",
      "name": "Disco",
      "baseLatency": 1000,
      "channels": 6,
      "isEnabled": true
    },
    {
      "type": "USB",
      "name": "USB",
      "baseLatency": 1000,
      "channels": 1,
      "isEnabled": false
    }
  ],
  "schedulerType": "Priority",
  "cpuCores": 1,
  "quantum": 30
}
```

Fonte: elaborado pelo autor (2025).

Quadro 11 – Conteúdo do Arquivo de Configuração de Processos, cenário 1 (cenario1.pconfig)

```
[
  {
    "loopConfig": {
      "loopConfig": "finite",
      "iterations": 1
    },
    "name": "Teste",
    "operations": [
      {
        "operationConfig": "cpu",
        "type": "Add",
        "min": 1,
        "max": 2,
        "repeatCount": 1
      },
      {
        "operationConfig": "io",
        "deviceType": "Disk",
        "duration": 5,
        "minDuration": 0,
        "maxDuration": 0,
        "isRandom": false,
        "repeatCount": 1
      },
      {
        "operationConfig": "cpu",
        "type": "Subtract",
        "min": 2,
        "max": 5,
        "repeatCount": 1
      }
    ],
    "priority": "Normal",
    "isSelectedForSimulation": true
  }
]}
```

Fonte: elaborado pelo autor (2025).

Quadro 12 – Conteúdo do Arquivo de Configuração de Processos, cenário 2 (cenario2.pconfig)

```
[
  {
    "loopConfig": {
      "loopConfig": "infinite"
    },
    "name": "CPU",
    "operations": [
      {
        "operationConfig": "cpu",
        "type": "Add",
        "min": 1,
        "max": 2,
        "repeatCount": 10000
      }
    ],
    "priority": "Normal",
    "isSelectedForSimulation": true
  },
  {
    "loopConfig": {
      "loopConfig": "infinite"
    },
    "name": "IO",
    "operations": [
      {
        "operationConfig": "io",
        "deviceType": "Disk",
        "duration": 5,
        "minDuration": 0,
        "maxDuration": 0,
        "isRandom": false,
        "repeatCount": 10000
      }
    ],
    "priority": "Normal",
    "isSelectedForSimulation": true
  }
]
```

Fonte: elaborado pelo autor (2025).

Quadro 13 – Conteúdo do Arquivo de Configuração da VM, cenário 2 Round Robin (cenario2RR.vmconfig)

```
{
  "devices": [
    {
      "type": "Disk",
      "name": "Disco",
      "baseLatency": 1000,
      "channels": 6,
      "isEnabled": true
    },
    {
      "type": "USB",
      "name": "USB",
      "baseLatency": 1000,
      "channels": 1,
      "isEnabled": false
    }
  ],
  "schedulerType": "RoundRobin",
  "cpuCores": 1,
  "quantum": 30
}
```

Fonte: elaborado pelo autor (2025).

Quadro 14 – Conteúdo do Arquivo de Configuração da VM, cenário 2 Prioridade (cenario2Prioridade.vmconfig)

```
{
  "devices": [
    {
      "type": "Disk",
      "name": "Disco",
      "baseLatency": 1000,
      "channels": 6,
      "isEnabled": true
    },
    {
      "type": "USB",
      "name": "USB",
      "baseLatency": 1000,
      "channels": 1,
      "isEnabled": false
    }
  ],
  "schedulerType": "Priority",
  "cpuCores": 1,
  "quantum": 30
}
```

Fonte: elaborado pelo autor (2025).

APÊNDICE B – JORNADA DO USUÁRIO

Este apêndice ilustra um ciclo pedagógico completo de uso do ProcSim, demonstrando como suas funcionalidades permitem a exploração prática de conceitos teóricos de Sistemas Operacionais. A jornada a seguir guia o usuário através da análise comparativa entre os algoritmos de escalonamento Round Robin e Prioridades Híbrido, focando no dilema entre equidade e responsividade.

Cenário de Estudo: Equidade vs. Responsividade

O objetivo deste experimento é visualizar e comparar como dois escalonadores distintos gerenciam dois processos com perfis de uso diferentes: um processo *CPU-bound* (que consome intensivamente a CPU) e um processo *I/O-bound* (que passa a maior parte do tempo esperando por operações de I/O).

1) Configuração da Máquina Virtual

O primeiro passo é configurar o ambiente da simulação. Na área "Configuração de VM", definimos os seguintes parâmetros para garantir que a competição por recursos seja clara (Figura 16):

- Algoritmo: Iniciaremos com Round Robin.
- CPU Cores: 1. O uso de um único núcleo força os processos a competirem diretamente pelo mesmo recurso de processamento.
- Quantum: 30.
- Dispositivos: Habilitar o dispositivo Disco.

Figura 16 – Configuração da Máquina Virtual para testes

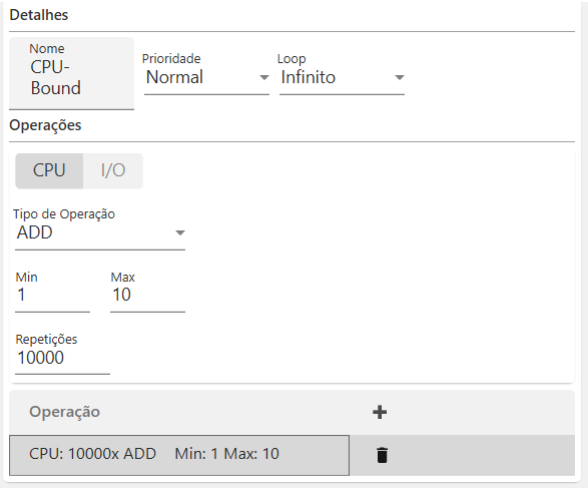
Fonte: elaborado pelo autor (2025).

2) Configuração dos Processos

A seguir, na área "Configuração de Processos", criamos dois processos:

- Processo "CPU-Bound" (Figura 17):
    - Nome: CPU-Bound
    - Loop: Infinito
    - Operações: Uma única operação de CPU do tipo ADD, com um número de repetições alto (ex: 10000). Isso garante que o processo estará sempre pronto para executar trabalho.
    - Prioridade: Normal

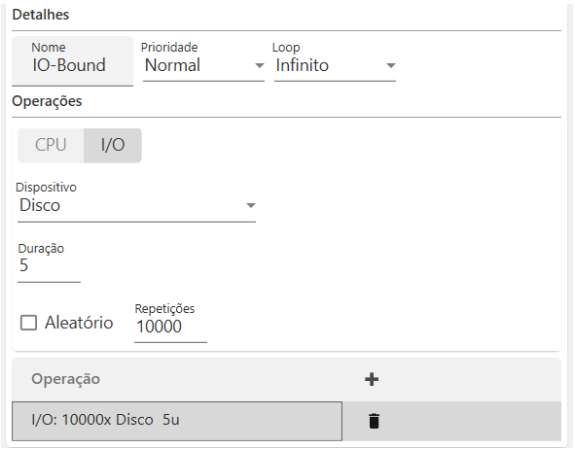
Figura 17 – Configuração do processo *CPU-Bound* para testes



Fonte: elaborado pelo autor (2025).

- Processo "IO-Bound" (Figura 18):
    - Nome: IO-Bound
    - Loop: Infinito
    - Operações: Uma única operação de I/O utilizando o dispositivo Disco, com uma Duração de 5 unidades e um número de repetições alto (ex: 10000). Isso fará com que o processo execute, bloqueie para I/O, e repita o ciclo.
    - Prioridade: Normal

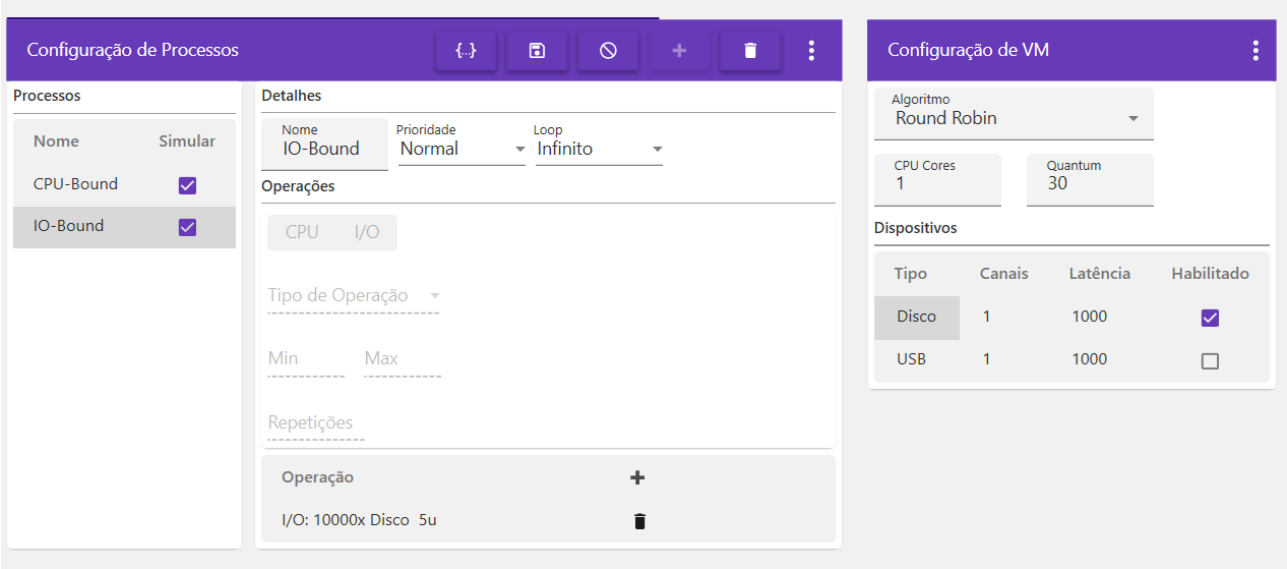
Figura 18 – Configuração do processo *IO-Bound* para testes



Fonte: elaborado pelo autor (2025).

Ao final, ambos os processos devem ser marcados com a opção "Simular". A tela de configuração deve se assemelhar à Figura 19.

Figura 19 – Painel de configuração do ProcSim com os processos do primeiro cenário

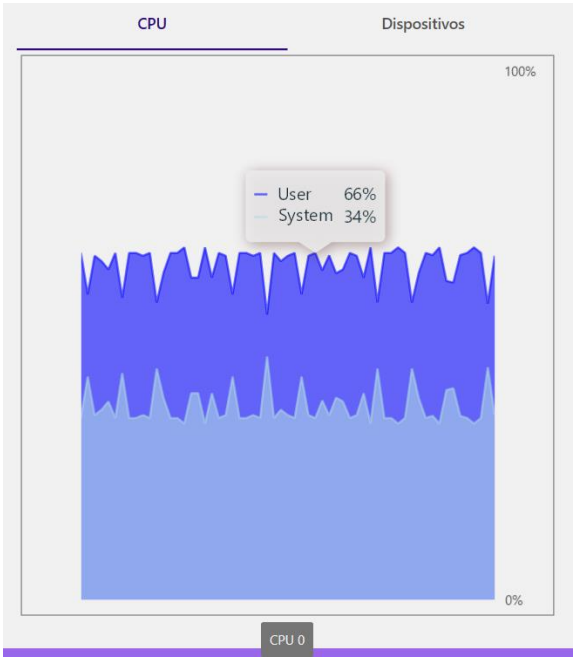


Fonte: elaborado pelo autor (2025).

3) Execução e Análise do Round Robin

Inicie a simulação. Navegue até a aba Monitoramento > CPU. Como esperado do algoritmo Round Robin, que preza pela equidade, o gráfico de uso da CPU mostrará que o tempo é dividido de forma homogênea entre os dois processos (Figura 20). O sistema operacional (tempo de sistema) e os processos "CPU-Bound" e "IO-Bound" (tempo de usuário) se revezarão no uso do processador, mesmo tendo perfis completamente diferentes.

Figura 20 – Gráfico de uso de CPU indicando distribuição homogênea de uso

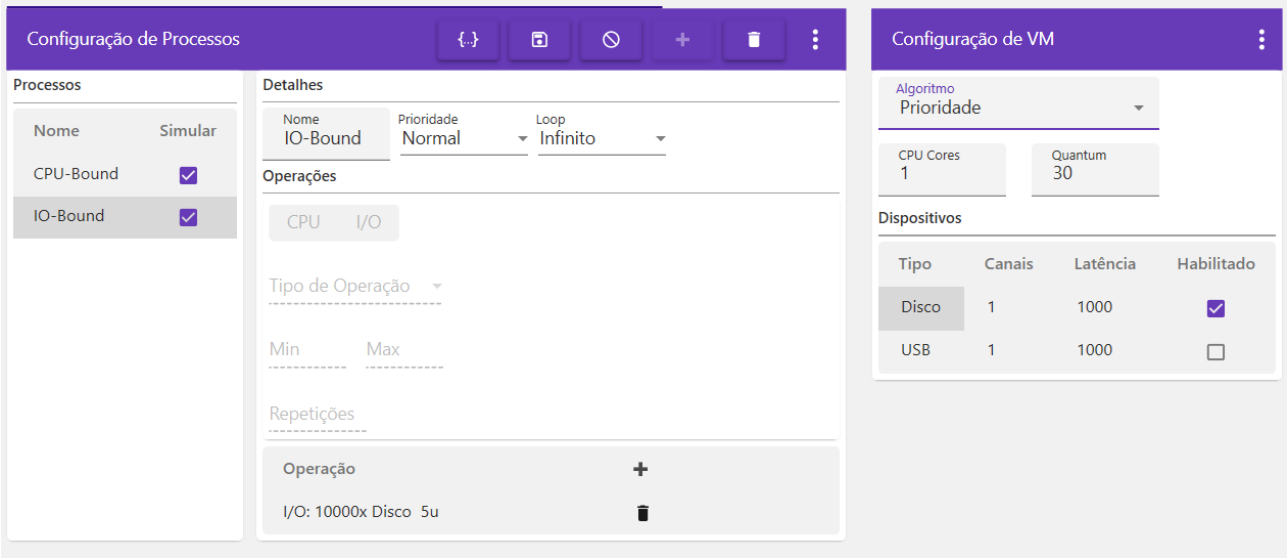


Fonte: elaborado pelo autor (2025).

4) Alteração do Cenário para o Escalonador por Prioridades

Pause e reinicie a simulação. Na "Configuração de VM", altere o Algoritmo para Prioridade e mantenha todas as outras configurações (Figura 21). Inicie a simulação novamente.

Figura 21 – Painel de configuração do ProcSim com os processos do segundo cenário

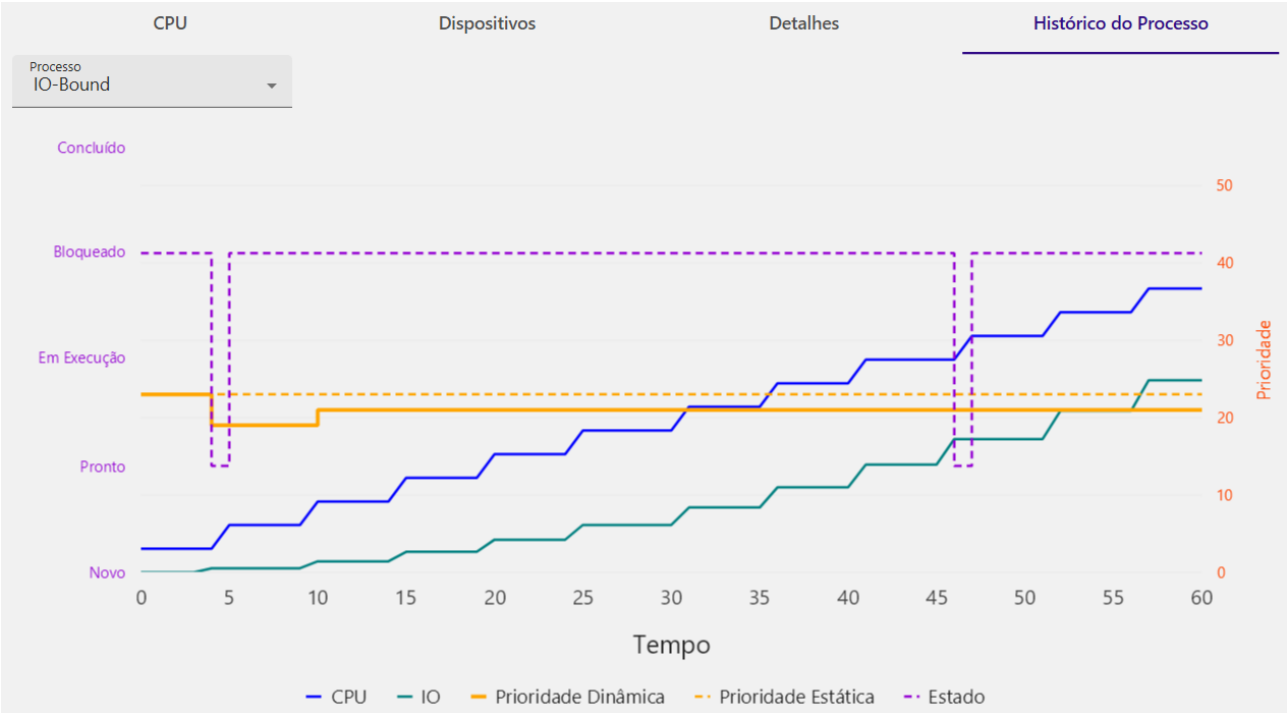


Fonte: elaborado pelo autor (2025).

5) Execução e Análise do Escalonador por Prioridades Híbrido

Com o novo escalonador, o comportamento do sistema muda drasticamente. Inicie a simulação, navegue até a aba Monitoramento > Histórico do Processo e selecione o processo IO-Bound na lista. O gráfico revelará o funcionamento da heurística de *boost* de prioridade (Figura 22).

Figura 22 – Análise do *Boost* de Prioridade para o processo I/O-Bound



Fonte: elaborado pelo autor (2025).

A análise do gráfico permite ao estudante observar que:

- A linha de Estado (roxa tracejada) mostra o processo alternando entre "Em Execução" e "Bloqueado".
- Imediatamente após o processo sair do estado "Bloqueado", a sua Prioridade Dinâmica (linha sólida laranja) recebe um *boost*, elevando-se acima de sua Prioridade Dinâmica inicial.
- Esse aumento garante que o processo IO-Bound seja escalonado rapidamente, passando à frente do processo CPU-Bound, o que melhora a responsividade do sistema.

- Caso possuísse operações de CPU intermediárias, sua prioridade dinâmica decairia gradualmente, até que ele solicitasse uma nova operação de I/O e o ciclo se repita.

Conclusão da Jornada

Através deste experimento prático, o estudante não apenas leu sobre os algoritmos de escalonamento, mas visualizou, interagiu e comparou seus efeitos em um cenário controlado. Ele pôde observar concretamente a dinâmica entre a distribuição justa de tempo do Round Robin e a otimização da responsividade do escalonador por Prioridades Híbrido, solidificando a compreensão de um dos conceitos mais fundamentais de escalonamento de processos.