UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

FELIPE TORRES MINORELLI

IMPLEMENTAÇÃO DE UM SIMULADOR MODULAR PARA O ENSINO DE SISTEMAS OPERACIONAIS

CAMPO MOURÃO

2025

versão 11.0 (abr. 25)

FELIPE TORRES MINORELLI

IMPLEMENTAÇÃO DE UM SIMULADOR MODULAR PARA O ENSINO DE SISTEMAS OPERACIONAIS

Implementation of a Modular Simulator for Teaching Operating Systems

Trabalho de Conclusão de Curso de Graduação como requisito para obtenção do título de Bacharel em Ciência da Computação do Curso de Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.
Orientador(a): Prof. Dr. Juliano Henrique Foleis

CAMPO MOURÃO

2025

FELIPE TORRES MINORELLI

IMPLEMENTAÇÃO DE UM SIMULADOR MODULAR PARA O ENSINO DE SISTEMAS OPERACIONAIS

Trabalho de Conclusão de Curso de Graduação como requisito para obtenção do título de Bacharel em Ciência da Computação do Curso de Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 18/junho/2025

Juliano Henrique Foleis
Doutorado
Universidade Tecnológica Federal do Paraná

Marcos Silvano Almeida
Doutorado
Universidade Tecnológica Federal do Paraná

Rogério Aparecido Gonçalves
Doutorado
Universidade Tecnológica Federal do Paraná

CAMPO MOURÃO

2025

AGRADECIMENTOS

Gostaria de expressar minha gratidão ao Prof. Dr. Rogério Aparecido Gonçalves pela orientação, e dedicação durante o desenvolvimento do Trabalho de Conclusão de Curso 1. Sua experiência, paciência e disponibilidade foram cruciais para a fundamentação deste projeto.

Por fim, agradeço profundamente ao Prof. Dr. Juliano Henrique Foleis, que, mesmo diante da minha demora em concluir o trabalho por conta dos imprevistos e desafios da vida, nunca desistiu do projeto. Sua paciência, bom humor, tempo e flexibilidade em me auxiliar, foram essenciais para que eu pudesse dar continuidade e finalizar esta etapa.

RESUMO

O curso de Ciência da Computação oferece ao aluno contato com tecnologias emergentes e consolidadas. As disciplinas abordam o gerenciamento de recursos, o desenvolvimento de sistemas computacionais e temas do meio empresarial. Apesar do incentivo a atividades práticas desde as primeiras disciplinas, muitas grades curriculares não oferecem uma carga horária ampla para esse fim. Diversas disciplinas têm forte conteúdo teórico a ser ministrado em um curto período. Na disciplina de Sistemas Operacionais, a importância das atividades práticas é evidente. Com o objetivo de apoiar o processo de ensino-aprendizagem, este trabalho apresenta o desenvolvimento de um simulador voltado ao ensino de escalonamento de processos. O simulador fornece uma implementação básica de um sistema operacional simulado, permitindo que os alunos implementem e testem diferentes algoritmos de escalonamento, reforçando os conceitos trabalhados em sala de aula. Para facilitar esse processo, foi desenvolvido uma estrutura que define um modelo a ser seguido na implementação das políticas de escalonamento. A implementação das políticas é feita por meio de *callbacks* específicos, acionadas em pontos predefinidos pelo simulador, abstraindo os detalhes dos mecanismos internos e concentrando a atenção nos aspectos conceituais da disciplina. O resultado é uma ferramenta de nível intermediário que permite visualizar o desempenho dos algoritmos de escalonamento no ambiente simulado. Essa abordagem busca tornar o aprendizado mais acessível e atrativo, evitando tanto a complexidade de implementações de baixo nível quanto a superficialidade de simuladores mais simples.

Palavras-chave: simulador; escalonamento; política; sistemas operacionais.

ABSTRACT

The Computer Science undergraduate program provides students with exposure to both emerging and consolidated technologies. Its curriculum includes topics such as resource management, software systems development, and aspects of the business environment. Although practical activities are encouraged from the early stages of the program, many curricula offer limited time for hands-on learning. Several courses present dense theoretical content that must be covered within a short timeframe. In the context of the Operating Systems course, the role of practical activities is particularly significant. This work presents the development of a simulator designed to support the teaching of process scheduling. The tool provides a basic implementation of a simulated operating system, allowing students to implement and test different scheduling algorithms, thereby reinforcing classroom concepts. To support this process, a framework was developed to define a standard model for implementing scheduling policies. These policies are implemented through specific callbacks, which are triggered at predefined points in the simulator. This abstraction enables students to focus on conceptual understanding while avoiding the complexities of internal mechanisms. The resulting tool is of intermediate complexity and offers a visual representation of the performance of different scheduling algorithms within the simulated environment. This approach aims to enhance student engagement by providing a more accessible and meaningful learning experience, bridging the gap between low-level implementation complexity and the superficial treatment found in simpler simulators.

Keywords: simulator; scheduling; policy; operating systems.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

E/S	Entrada e Saída
SO	Sistema Operacional

SUMÁRIO

1 INTRODUÇÃO

Estudos anteriores já mostram a importância das atividades práticas. Hofstein e Lunetta (1982) destacam que essas aulas ajudam a despertar e manter o interesse dos alunos. Elas ainda contribuem para a compreensão e assimilação do conteúdo, pois permitem que os estudantes experimentem e vivenciem o que está sendo proposto.

Por outro lado, disciplinas com alta carga teórica, como Sistemas Operacionais, reduzem o tempo disponível para atividades práticas. Este é um dos obstáculos que buscamos superar com a proposta deste projeto. Devido à limitação de tempo, exigir que o aluno implemente um sistema operacional completo se torna inviável. Por isso, nosso objetivo é fornecer um ambiente em que o aluno possa focar apenas na implementação dos algoritmos de escalonamento discutidos em sala, sem a necessidade de desenvolver os demais mecanismos de um sistema operacional.

Uma vez que alterações em sistemas reais não são triviais ou não permitem ao aluno ver o resultado de maneira rápida, a utilização de simuladores se torna necessária. Dada a necessidade do ensino sobre os algoritmos de escalonamento, lecionar estes conceitos apenas de forma teórica faz com que este tópico da disciplina se torne cansativo e desestimulante. Portanto, o simulador aqui proposto tem como finalidade facilitar a interação entre o aluno e um ambiente controlado, onde os conceitos vistos em sala de aula possam ser aplicados e seus resultados visualizados de maneira fácil sem distanciar o simulador de um Sistema Operacional (SO) real.

O uso de simuladores ajuda quando não temos o sistema real que é o objeto do estudo. Existem diversos simuladores de Sistemas Operacionais, são ferramentas que estão mais próximas de um sistema operacional real como o `Nachos` (Christopher; Procter; Anderson, 1993), o `PortOS` (Atkin; Sirer, 2002) e outros de nível médio como `SOS` (Gonçalves *et al.*, 2004) e `SOIS` (Gonçalves; Gonçalves; Martini, 2006; Cruz; Silva; Gonçalves, 2007) e ferramentas que trazem representações de mais alto nível dos conceitos relacionados à disciplina de Sistemas Operacionais como o `SOsim` (Maia; Pacheco, 2003).

1.1 Objetivos

O objetivo deste trabalho foi implementar um simulador de Sistema Operacional com foco no gerenciamento de processos. O simulador permite que os alunos implementem os algoritmos de escalonamento discutidos em sala e adicionem novos módulos, se necessário. A simulação é baseada em arquivos de entrada que descrevem os eventos relevantes ao escalonamento durante a execução dos processos. Isso possibilita repetir experimentos com diferentes políticas. Além disso, o simulador gera estatísticas que auxiliam na avaliação do desempenho das políticas implementadas.

1.2 Contribuições

Este trabalho tem como base a construção de uma ferramenta que seja capaz de fornecer um ambiente onde o aluno da disciplina de Sistemas Operacionais possa implementar os algoritmos de escalonamento, e visualizar os diferentes resultados de acordo com a politica de escalonamento implementada. Dentre as ferramentas citadas neste capítulo, e no Capítulo 3, algumas permitem a execução de código real, porém nenhuma delas permite que o aluno seja o criador dos algoritmos de escalonamento de maneira direta, esta é uma das funções que propomos para nosso simulador.

1.3 Organização do Texto

Neste capítulo foram apresentados os objetivos que pretendemos alcançar com nosso projeto. No Capítulo 2 são explicados os conceitos abordados em nosso trabalho. No Capítulo 3 apresentamos trabalhos relacionados com nossa proposta. A implementação e a arquitetura utilizadas são apresentadas no Capítulo 4. Por fim, no Capítulo 5 apresentamos as conclusões obtidas no desenvolvimento deste trabalho.

2 GERENCIAMENTO DE PROCESSOS

Um Sistema Operacional é um *software* que tem como objetivo oferecer uma interface entre o usuário e o conjunto de *hardware* encontrado em um sistema computacional. Além de tornar esta comunicação entre as duas partes mais simples, o SO diminui a preocupação do usuário final com o gerenciamento de recursos do sistema, já que dentro de suas funcionalidades se encontram mecanismos para a realização desta tarefa. Dentre estes mecanismos de gerenciamento de recursos, temos os que gerenciam memória, processos e disco. O foco principal do simulador, objeto de nossa proposta, é o módulo de gerenciamento de processos.

Com o propósito de manter todos os dispositivos do sistema ocupados, o SO torna o escalonamento de processos um mecanismo indispensável em sistemas multiprogramados. No decorrer da execução de um processo, o mesmo pode fazer solicitações a vários dispositivos, tais como os mecanismos de Entrada e Saída (E/S). Durante as operações de E/S a *Central Processing Unit* (CPU) se encontra inoperante até que o evento aguardado pelo processo termine. Para evitar esta ociosidade, o escalonador de processos entra em ação, escolhendo um outro processo que esteja pronto para ser executado, efetuando a troca de contexto entre os processos (Figura 1). Desta forma, tanto a CPU quanto os dispositivos de E/S se encontram operantes simultaneamente, maximizando a produtividade e eficiência do sistema (Silberschatz; Galvin; Gagne, 2013).

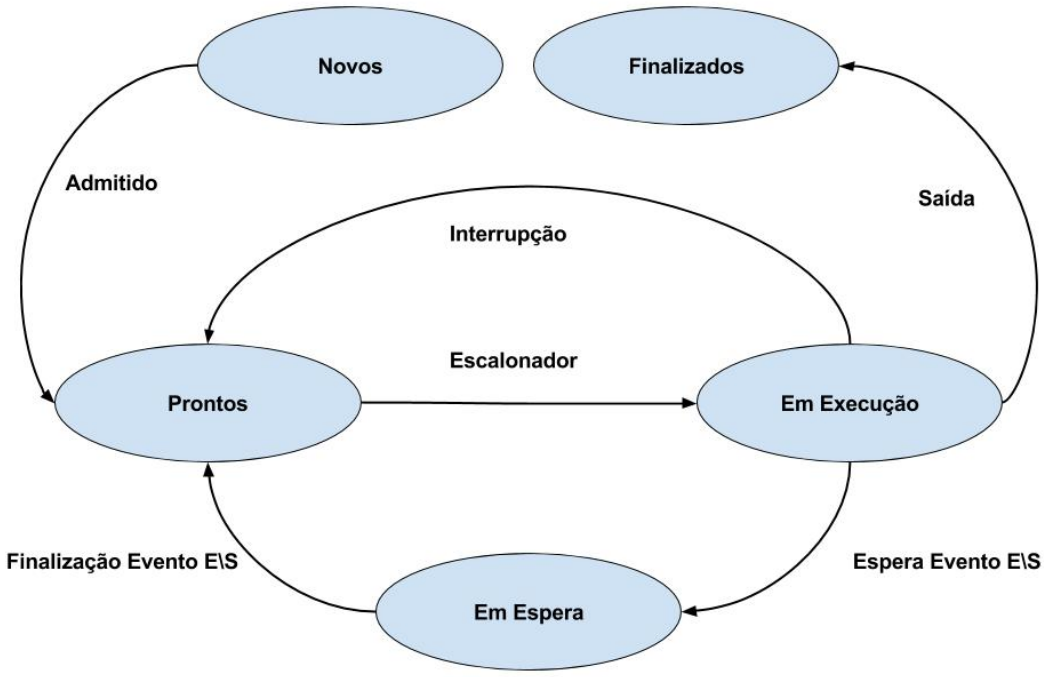

Figura 1 – Diagrama de estados de um processo. Adaptado de Silberschatz, Galvin e Gagne (2013)

O escalonamento deve ser feito com cautela, já que em sistemas interativos a presença de um usuário requer respostas rápidas, e uma troca de contexto ineficiente pode gerar períodos de espera.

2.1 Objetivos de um algoritmos de escalonamento

Uma das principais metas de um algoritmo de escalonamento é manter a CPU sempre em utilização, otimizando o fluxo de processos que possam ser interrompidos por instruções de E/S (Pearson; Tanenbaum, 2008). Para alcançar este objetivo, o sistema precisa manter em memória todos os processos em execução, e os limitados pelas instruções de E/S.

Três métricas que medem o desempenho do escalonador de processos são:

- **Vazão:** Representa a quantidade de processos finalizados em um intervalo de tempo. Quanto maior a quantidade de processos finalizados no intervalo, melhor.
- **Tempo de retorno:** Medida de tempo que processos levaram desde o momento em que chegaram ao sistema até seu termino. Esse tempo inclui períodos de espera, execução e bloqueio. Quanto menor, melhor.
- **Utilização de CPU:** Apesar de não ser uma boa métrica para mensurar o desempenho do sistema, pode ser um bom indicativo de uso de recursos computacionais. Valores como vazão e tempo de retorno são mais valiosos na avaliação de um algoritmo de escalonamento.

Como ferramenta adicional, o simulador proposto neste trabalho, ao final de cada simulação, exibe um diagrama de Gantt. Esse recurso permite ao usuário confirmar visualmente os pontos de entrada, finalização e o tempo de execução de cada processo. Isso ajuda a depurar o algoritmo de escalonamento implementado e auxilia na compreensão de cada política.

2.2 Tipos de algoritmos de escalonamento

Para se adequar às necessidades de cada SO, uma variedade de algoritmos de escalonamento pode ser utilizada. Nesta seção, serão apresentados alguns dos algoritmos mais usados em sistemas interativos.

2.2.1 FCFS (*First-Come, First-Served*)

O algoritmo **primeiro a chegar, primeiro a ser servido (first-come, first-served)** é o mais simples entre os algoritmos de escalonamento. Ele atribui a CPU aos processos na ordem de chegada. Os processos ficam em uma fila única, e o primeiro da fila é executado até ser bloqueado. Nesse momento, o processo bloqueado é movido para o fim da fila, e o próximo processo passa a ser executado. Esse ciclo se repete até que todos os processos sejam finalizados.

2.2.2 SJF (*Shortest-Job-First*)

O algoritmo **tarefa mais curta primeiro (shortest job first)** é usado quando o tempo de execução das tarefas é conhecido antecipadamente. Ele é uma boa escolha quando todas as tarefas estão disponíveis simultaneamente, pois facilita a ordenação dos processos. Essa limitação pode ser superada com a versão preemptiva do SJF, chamada **tempo restante mais curto**. Nessa versão, o algoritmo considera o tempo restante das novas tarefas em comparação com o processo em execução. Se a nova tarefa requer menos tempo para terminar, o processo atual é suspenso. Essa abordagem reduz o tempo de espera das tarefas menores.

2.2.3 *Round-Robin*:

O algoritmo de chaveamento **circular** (round-robin) é um dos mais antigos, justos e amplamente usados. Nesse esquema, cada processo recebe um intervalo de tempo fixo, chamado quantum. Após usar esse quantum, o processo é realocado ao final da fila. Esse método garante uma execução justa para todos os processos. No entanto, o *quantum* deve ser configurado corretamente. Um *quantum* muito curto pode sobrecarregar a CPU com muitos trocas de contexto, enquanto um quantum muito longo reduz a eficiência para solicitações interativas curtas.

2.2.4 Escalonamento por prioridade:

Diferente do escalonamento circular, onde todos os processos têm a mesma importância, no algoritmo de prioridades cada processo recebe uma prioridade. Esse método é ideal para ambientes que exigem considerar fatores externos, como um sistema que exibe vídeo ao usuário enquanto mantém rotinas em segundo plano. Esse algoritmo pode ser implementado de várias formas, por exemplo, reduzindo a prioridade do processo atual a cada *tick*. Também pode ser combinado com a estratégia *round-robin* mencionada anteriormente.

3 TRABALHOS RELACIONADOS

Na literatura, existem diversos trabalhos relacionados ao ensino de Sistemas Operacionais. Há simuladores como o Nachos (Christopher; Procter; Anderson, 1993) e o PortOS (Atkin; Sirer, 2002), que se aproximam mais dos conceitos reais da disciplina. Outros, de nível intermediário, são o SOS (Gonçalves *et al.*, 2004) e o SOIS (Gonçalves; Gonçalves; Martini, 2006; Cruz; Silva; Gonçalves, 2007). Também há ferramentas de nível mais básico, com representações simples dos conceitos de gerenciamento de processos e memória, como o SOsim (Maia; Pacheco, 2003).

Nas seções seguintes, cada um desses simuladores será descrito, e suas principais características, listadas.

3.1 Nachos

O Nachos (Christopher; Procter; Anderson, 1993) é um sistema operacional simulado voltado para o ensino de graduação em Sistemas Operacionais. Ele possui uma interface em modo texto simples e permite simular a CPU, processos, *threads*, *Remote Procedure Call* (RPC), hierarquia de memória, programação orientada a objetos e computação distribuída.

3.2 PortOS

O PortOS (Atkin; Sirer, 2002), direcionado ao ensino de graduação e pós-graduação, é uma plataforma intuitiva que guia os usuários no desenvolvimento de um sistema operacional real em nível de usuário para dispositivos móveis do tipo *Personal Digital Assistant* (PDA). Ele permite o uso de recursos como *threads*, concorrência, sistema de arquivos, escalonamento, sincronização e comunicação ponto a ponto.

3.3 SOsim

Desenvolvido por (Maia; Pacheco, 2003), o SOsim é um simulador que permite a visualização dos conceitos e mecanismos de um sistema operacional multiprogramado por meio de uma interface gráfica. A Figura 2 mostra sua interface. O simulador oferece recursos de gerenciamento de processos e memória, além da seleção de diferentes políticas de escalonamento. Dessa forma, o aluno pode interagir com os conceitos teóricos apresentados durante a disciplina.

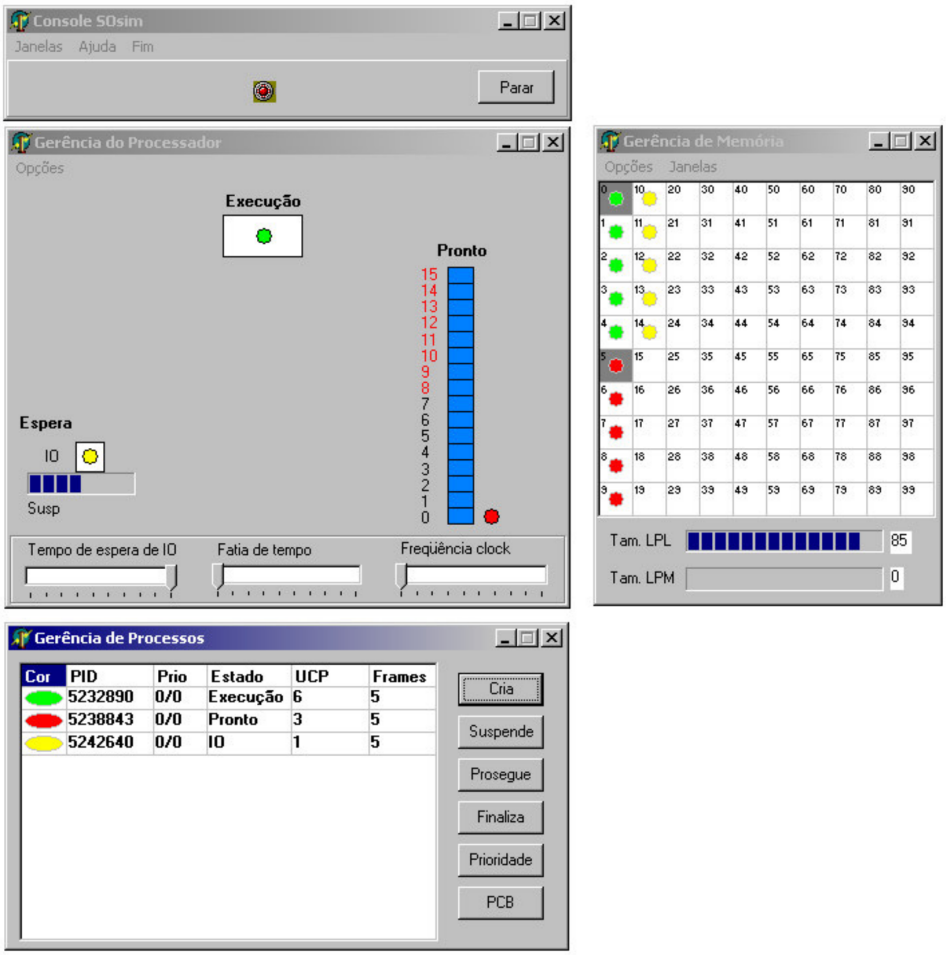

Figura 2 – Interface do `SOsim` (Machado; Maia, 2004)

3.4 SOS

O Sistema Operacional Simulado (SOS) (Gonçalves *et al.*, 2004) simula um ambiente semelhante ao de um SO real. A Figura 3 mostra a interface do SOS. Ele reproduz o funcionamento de um sistema operacional por meio de módulos de gerenciamento de disco e memória. Além disso, possui um conjunto próprio de instruções que permite ao usuário criar programas e executá-los no simulador. O SOS é utilizado como ferramenta de apoio ao ensino de Sistemas Operacionais na graduação. Também oferece a funcionalidade de geração de estatísticas de desempenho, essenciais para a análise da performance dos algoritmos de escalonamento.

3.5 SOIS

O SOIS (Gonçalves; Gonçalves; Martini, 2006; Cruz; Silva; Gonçalves, 2007) é uma continuação do projeto SOS, com o objetivo de permitir a execução de código real pelos processos. Apesar de suportar apenas um subconjunto de instruções da arquitetura IA-32, devido ao uso de um processador simulado, o SOIS é capaz de executar programas reais e exibir sua execução ao usuário. O simulador também inclui um módulo de gerenciamento de entrada/saída,

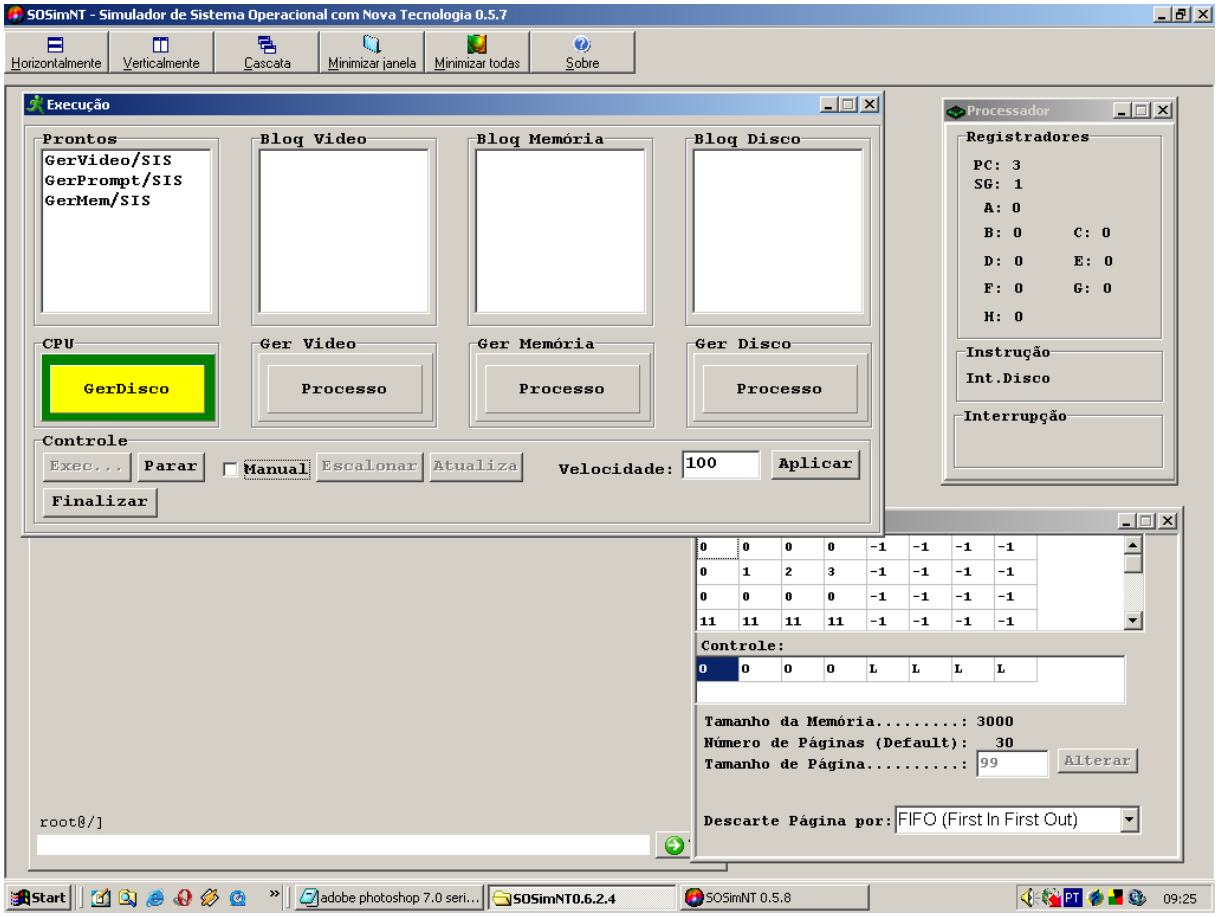

Figura 3 – Interface do SOS (Gonçalves *et al.*, 2004)

que permite o estudo do funcionamento do *hardware* durante essas operações. Isso o torna uma ferramenta útil para a aprendizagem de conceitos de mais baixo nível.

3.6 MINIX

Desenvolvido por Andrew S. Tanenbaum, o `MINIX` é um sistema operacional escrito em `C` e `Assembly`, baseado no `UNIX` e voltado para fins educacionais. Possui suporte a multitarefas, memória estendida e até três usuários simultâneos. O sistema inclui editores de texto, mais de 200 utilitários (como `cat`, `cp`, `grep`, `make`) e mais de 300 bibliotecas (como `fork`, `malloc`, `read`, `stdio`). A Figura 4 mostra a interface do `MINIX`. O `MINIX` conta com ampla documentação e uma comunidade ativa. É amplamente reconhecido por ter inspirado o desenvolvimento do sistema hoje conhecido como `LINUX`.

3.7 Considerações Finais

As ferramentas apresentadas podem ser aplicadas em algum nível do ensino da disciplina de Sistemas Operacionais. Algumas são ferramentas de mais alto nível que simulam partes e conceitos de um Sistema Operacional, outras executam código de processos escri-

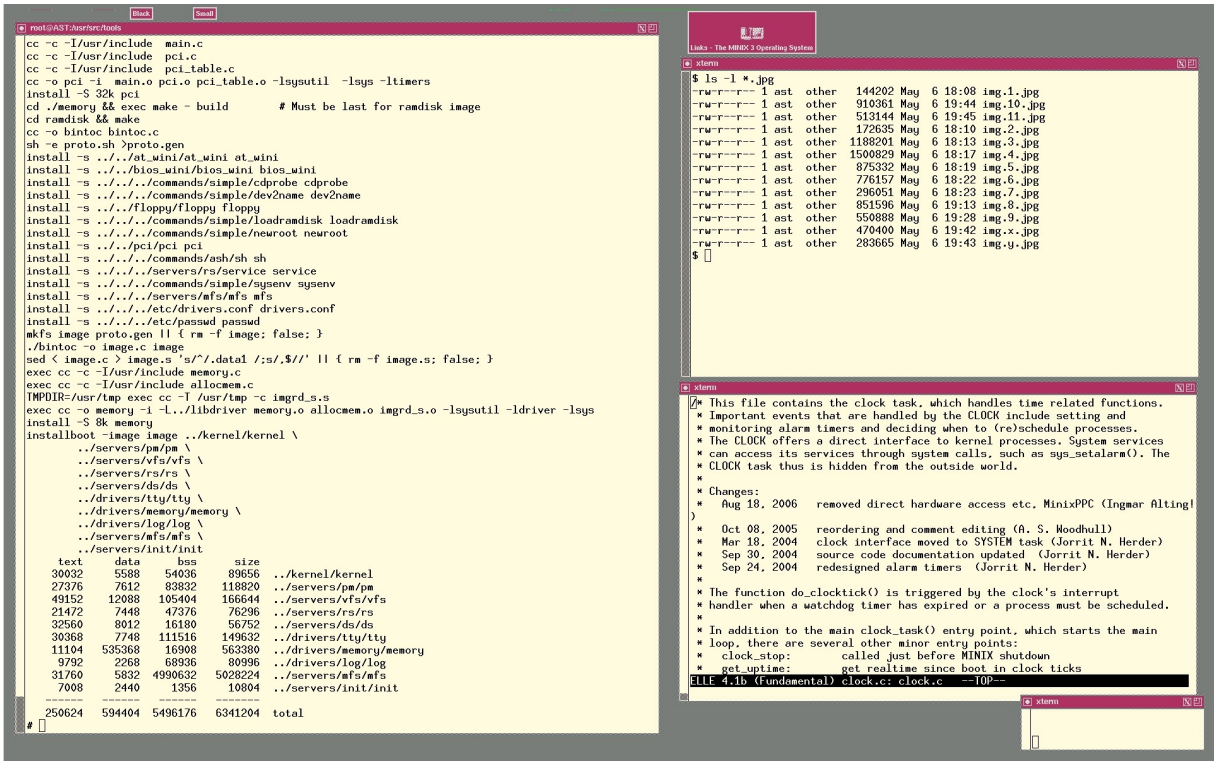

Figura 4 – Interface do `Minix`(Tanenbaum, 2017)

tos em uma linguagem de montagem para uma arquitetura simulada, e outros simuladores são capazes de executar o código de um SO real.

O diferencial deste trabalho é oferecer um simulador que foi projetado para ser estendido com novos algoritmos de escalonamento. Desta forma, é possível estudar os algoritmos existentes e avaliar novas estratégias, sem a necessidade de construir um simulador novo ou alterar de forma significativa o simulador desenvolvido neste trabalho.

4 ARQUITETURA E IMPLEMENTAÇÃO DO SIMULADOR

O objetivo deste trabalho foi oferecer um simulador de sistema operacional que permita ao aluno a implementação em Python da lógica dos algoritmos de escalonamento de processos. O simulador foi projetado de forma que para implementar um novo algoritmo, basta herdar a classe da política de escalonamento e implementar os métodos necessários. Neste capítulo é apresentada a estrutura do simulador, bem como suas principais funcionalidades.

O simulador pode ser dividido em três partes principais: o núcleo da simulação, a política de escalonamento e o módulo de gerenciamento de processos. Essa divisão tem como objetivo simplificar e organizar a estrutura, e futuramente possibilitar a inclusão de novos módulos.

O código do simulador e de todos os utilitários estão disponíveis no Github[1]

4.1 Núcleo do Simulador

O núcleo é responsável pelas rotinas de simulação, tais como a leitura do arquivo de entrada, a preparação da política selecionada, o cálculo de métricas e a geração de estatísticas ao final do processamento. Como ilustrado na Figura 5.

[1] https://github.com/Fminorelli/tcc

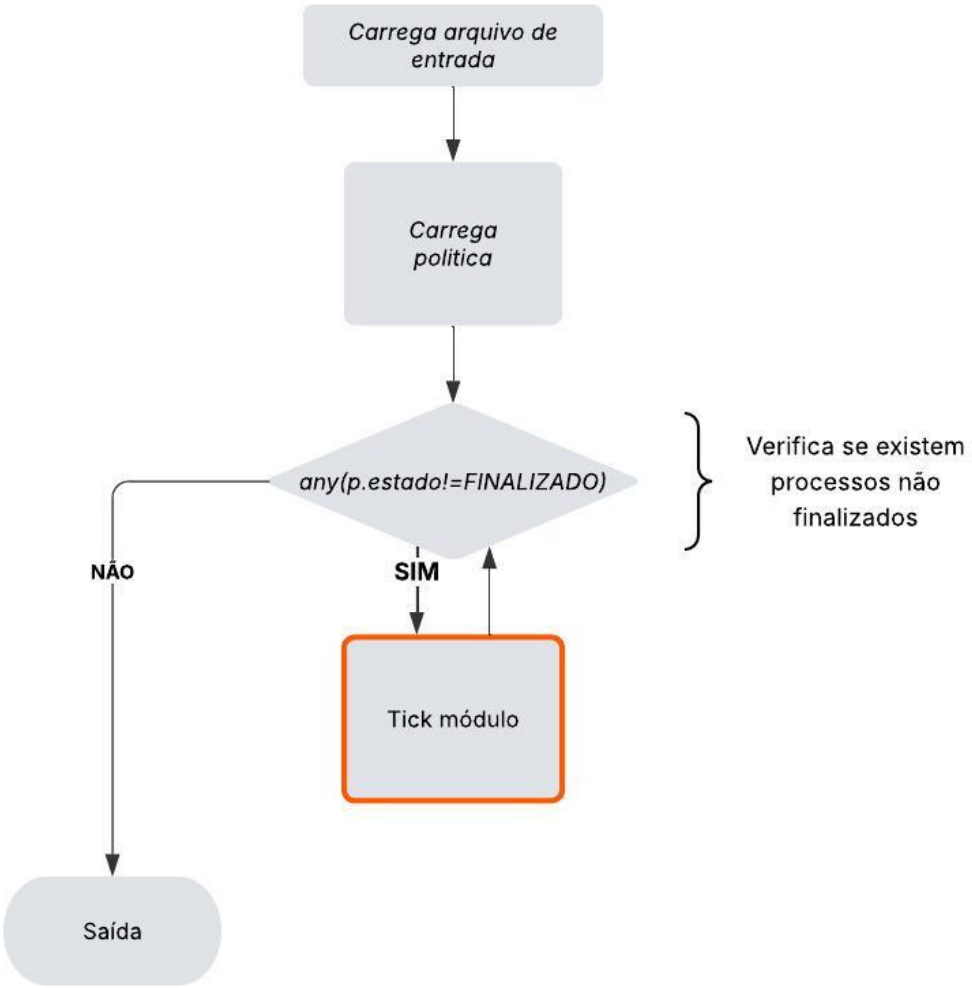

Figura 5 – Fluxo Núcleo do Simulador

4.2 Política de Escalonamento

As políticas de escalonamento devem herdar uma classe abstrata pré-definida pelo simulador denominada `politicaGP`. É obrigatório implementar os métodos a seguir:

- **inicializar**: Utilizada caso a politica necessite criar filas, contadores, ou estruturas de controle internas. Recebe como parâmetro, a lista de processos, e os parâmetros adicionais da politica.
- **iniciar**: Deve ser implementada caso a política precise atualizar sua estruturas de controles internos na chegada de um novo processo.
- **selecionar_proximo**: Invocada pelo simulador sempre que precisa decidir qual processo será executado a seguir. Essa função deve analisar os processos que estão prontos para execução e, com base na política de escalonamento (FIFO, Round Robin, prioridade, etc.), escolher o próximo processo que ocupará a CPU.

- **desbloquear**: Chamada sempre que um processo que estava bloqueado termina seu tempo de bloqueio e pode voltar a competir pela CPU na lista de prontos.
- **bloquear**: É chamada sempre que um processo deve ser bloqueado.
- **finalizar**: Responsável por informar a política sobre o fim do processo, e atualizar estruturas tais como filas ou listas internas da política.
- **tick**: Similar a função *tick* do núcleo do simulador. Utilizada quando a politica precisa atualizar suas listas, filas, contadores, e se necessário os estados dos processos.

Na Figura 6 é possível visualizar a relação de herança entre a classe `politicaGP` e a politica de escalonamento implementada pelo usuário.

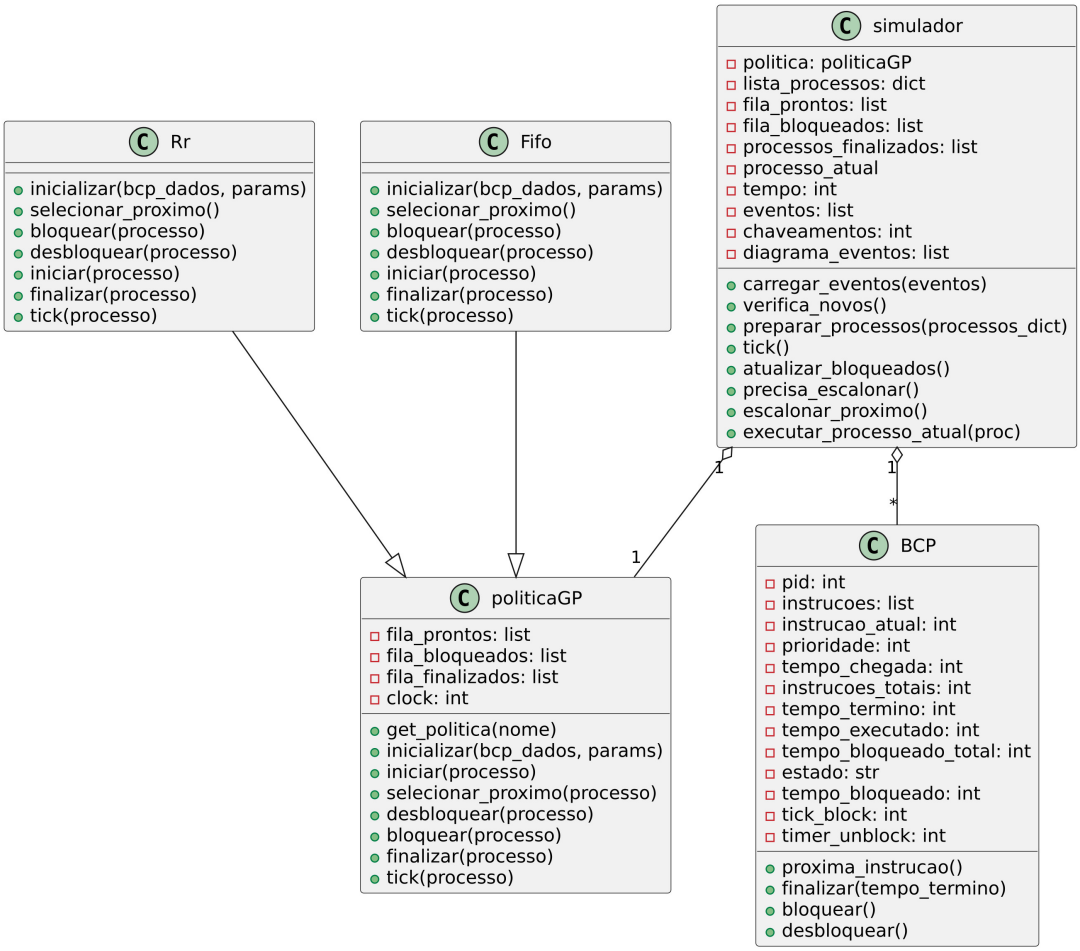

Figura 6 – Fluxo Núcleo do Simulador

No Apêndice A esta presente como executar o simulador, um exemplo da implementação da política FIFO, assim como o arquivo de entrada utilizado, as métricas, estatísticas e o diagrama de Gantt.

4.3 Módulo de Gerenciamento de Processos

Este módulo é responsável por processar os eventos definidos no arquivo de entrada, controlar as filas de processos, acionar os *callbacks* das rotinas de inicialização, bloqueio, desbloqueio e finalização, além de armazenar os dados que serão utilizados na geração das estatísticas finais da simulação. Todas essas tarefas são executadas na função *tick* deste módulo, detalhada a seguir.

4.3.1 Ciclo de Simulação

O ciclo de simulação, definido pela instrução de *tick*, é executado até que todas as instruções sejam processadas, e todos os processos sejam finalizados. Cada chamada representa a passagem de uma unidade de tempo, durante a qual uma série de eventos acontecem:

- **Atualizar o tempo da simulação**: Incrementa em 1 unidade o tempo global da simulação, representando a execução de uma instrução do processo atual.
- **Verificar chegada de novos processos**: Adiciona na fila de prontos qualquer processo com tempo de chegada equivalente ao tempo atual.
- **Atualizar a lista de processos bloqueados**: Incrementa o tempo de bloqueio de cada processo nessa fila. Aqueles que atingirem o tempo de desbloqueio são movidos de volta para a fila de prontos.
- **Executar o processo atual**: Executa a instrução do processo atual. Se a instrução não for de bloqueio, e a política não solicite a troca de processo, o contador de tempo de execução do processo é incrementado em 1 unidade.
- **Verificar a necessidade de escalonamento**: Avalia se a CPU está ociosa e se há processos prontos. Se sim, solicita o escalonamento de um novo processo para a política.
- **Escalonar um novo processo**: Solicita à política de escalonamento a escolha do próximo processo a ser executado, de acordo com os critérios definidos pela mesma.

A Figura 7 demonstra a ordem de execução de cada uma dessas rotinas, bem como os momentos em que os *callbacks* das políticas de escalonamento são invocados.

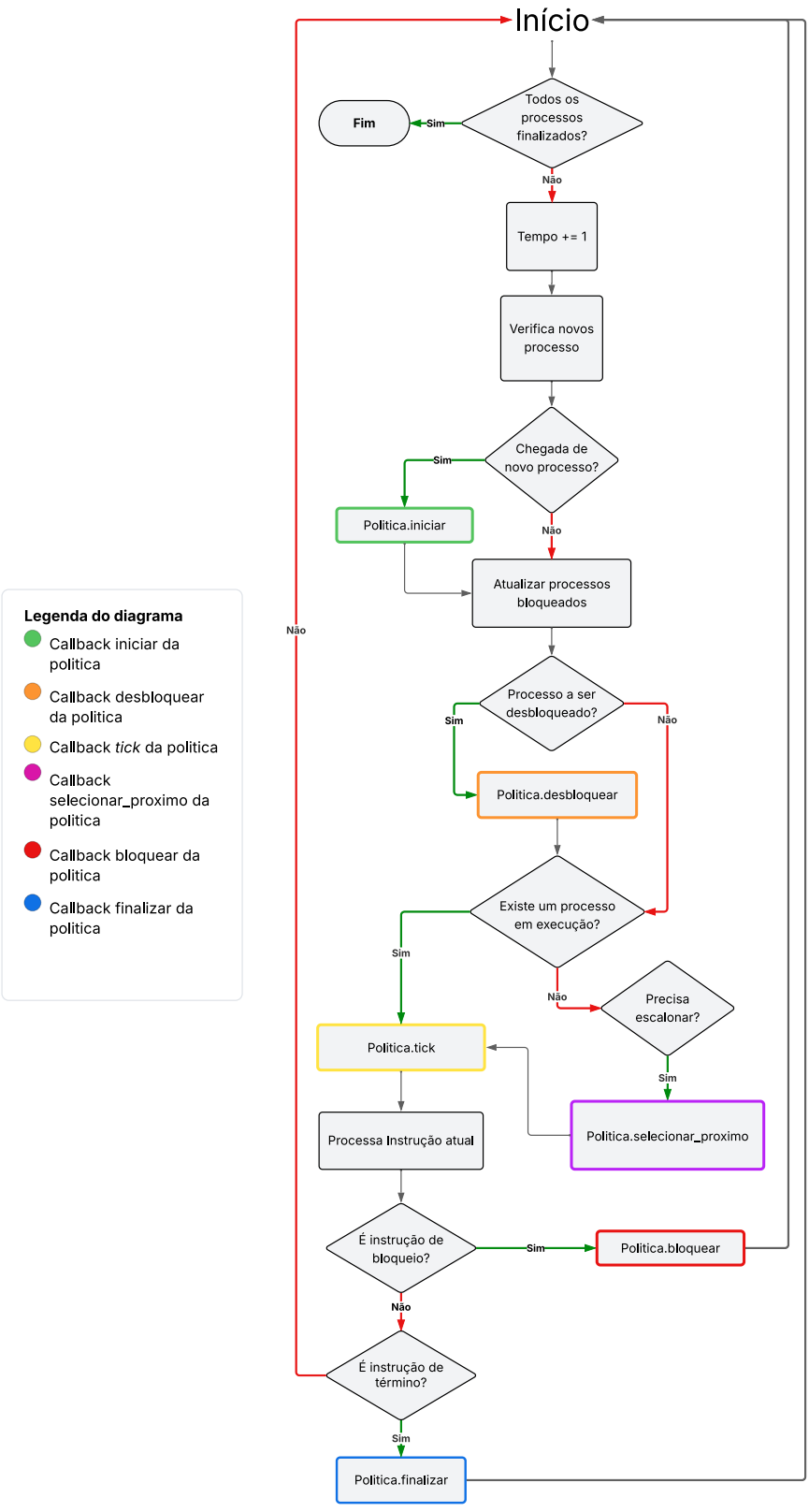

Figura 7 – Fluxo de execução da função *tick*

4.4 Arquivo de Entrada

O arquivo de entrada do simulador é dividido em três partes principais: *a identificação do módulo, os parâmetros de configuração e a lista de processos.* Esses elementos podem ser observados na Listagem 1, e serão discutidos nas próximas subseções.

Listagem 1 – Exemplo de arquivo de entrada

```
gp:
  config:
    nome: gerenciador de processos
    politica:
      nome: rr
      params:
        quantum: 2
  processos:
    pid 1:
      instrucoes:
      - 1 start
      - 11 block 3
      - 15 end
    pid 2:
      instrucoes:
      - 1 start
      - 3 block 2
      - 7 block 1
      - 11 block 2
      - 14 end
    pid 3:
      instrucoes:
      - 1 start
      - 4 block 3
      - 5 block 3
      - 11 block 2
      - 12 end
```

A primeira linha do arquivo de entrada deve indicar o módulo ao qual as demais configurações se referem. No simulador aqui apresentado, o foco está no módulo de gerenciamento de processos (`gp`), porém, a arquitetura foi projetada para ser modular, permitindo a inclusão futura de outros módulos, como gerenciamento de memória e de disco.

Na seção `config` arquivo de entrada são definidos a política de escalonamento a ser utilizada e seus respectivos parâmetros, quando necessário. No exemplo da Listagem 1, a política Round Robin (identificada por `rr`) requer o parâmetro *quantum*. O valor do campo `nome` é usado para carregar o módulo Python que contém a implementação da política. No exemplo da Listagem 1, o simulador irá procurar o arquivo **processos.yaml** que deve conter a implementação da política em uma classe que herda a classe abstrata que define a política de gerenciamento de processos.

4.4.1 Lista de Processos

Nesta parte do arquivo são definidos os processos que serão simulados, bem como suas instruções. O simulador reconhece três tipos de instrução:

- **start**: Indica o tempo global em que o processo entra no sistema.
- **block**: Especifica após quanto tempo em execução o processo deverá ser bloqueado e por quantos *ticks* permanecerá nesse estado.
- **end**: Define por quanto tempo o processo deverá ocupar a CPU antes de ser finalizado.

4.5 Saída e Estatísticas

Durante cada ciclo, o simulador registra o número de instruções executadas em cada processo. Ao final da simulação, ele utiliza esses dados para calcular cinco métricas para análise da eficiência da política de escalonamento escolhida:

- **Chaveamentos**: Chaveamentos ocorrem sempre que a CPU troca o processo em execução por outro. Isso pode acontecer por diversos motivos, como término da execução, bloqueio ou decisão da política de escalonamento.
- **Tempo Médio de Retorno**: Calcula a médio do Tempo de Retorno de todos os processos.
- **Tempo Médio de Espera**: Indica quanto tempo, em média, os processos ficaram esperando na fila de prontos, ou seja, tempo em que não estavam executando nem bloqueados.
- **Vazão**: No simulador, a vazão é calculada a cada 1000 ticks.
- **Ordem de Término**: Mostra a sequência em que os processos foram finalizados.

Além das métricas, o simulador também gera um diagrama de Gantt, que permite visualizar, de forma clara e intuitiva, a ocupação da CPU ao longo do tempo por cada processo. A Figura 8 apresenta o diagrama gerado com base no arquivo de entrada mostrado anteriormente na Listagem 1.

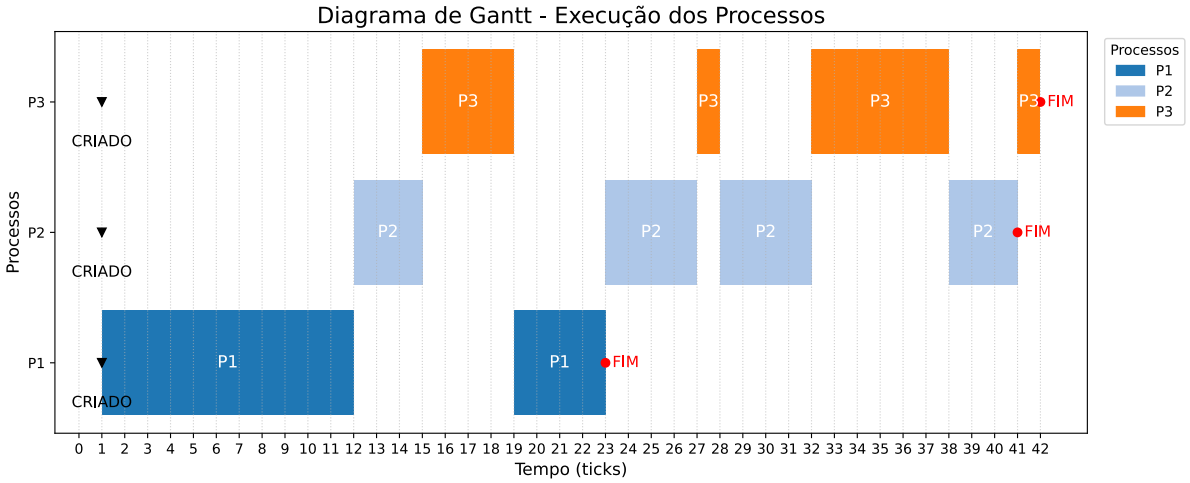

Figura 8 – Exemplo diagrama de Gantt

4.6 Classes Auxiliares

Com o objetivo de facilitar a manutenção e a extensibilidade do sistema, o simulador desenvolvido neste trabalho foi implementado em Python, utilizando o paradigma de programação orientada a objetos. Essa abordagem permite a modularização dos componentes, possibilitando a adição, substituição ou modificação de funcionalidades sem impactar o núcleo do simulador.

Esse aspecto pode ser observado diretamente nas seções dedicadas às políticas de escalonamento, nas quais cada classe implementada deve estender uma classe base fornecida pelo simulador. A estrutura modular do código também foi projetada para permitir a inclusão de funcionalidades adicionais, seja por meio da extensão das classes existentes, seja pela incorporação de novos módulos.

4.6.1 BCP

A classe `BCP` (Bloco de Controle de Processo) é responsável por armazenar e gerenciar os dados internos de cada processo, incluindo informações como tempo de execução, próxima instrução, tempo restante para desbloqueio, número total de instruções, além de métodos responsáveis por alterar o estado do processo. A Listagem 2 mostra a implementação da classe BCP.

Listagem 2 – Classe BCP

```
class BCP:
    def __init__(self, pid, dados):
        self.pid = pid
        self.instrucoes = dados.get('instrucoes', [])
        self.instrucao_atual = 0
        self.prioridade = dados.get('prioridade', 0)
        self.tempo_chegada = dados['start']
        self.instrucoes_totais = dados['end']
        self.tempo_termino = None
        self.tempo_executado = 0   #Tempo real de execução
        self.tempo_bloqueado_total = 0  #Tempo em estado bloqueado total
        self.qtm_block = False
        self.estado = 'EM ESPERA'
        self.tempo_bloqueado = 0      #Tempo em estado bloqueado
        self.tick_block = 0      #Instruções executadas ate bloquear
        self.timer_unblock = 0    #Número de ticks para desbloquear

    def proxima_instrucao(self):
        if self.instrucao_atual < len (self.instrucoes):
            instrucao = self.instrucoes[self.instrucao_atual]
            try:
                _, timer_unblock, tempo_block = instrucao.split()
                self.timer_unblock = timer_unblock
                self.tick_block = tempo_block
                self.instrucao_atual += 1

            except (ValueError, IndexError) as e:
                print(f"Erro ao processar instrução block: {e}")
        else:
            self.tick_block = 0

    def finalizar(self, tempo_termino):
        self.estado = "FINALIZADO"
        self.tempo_termino = tempo_termino

    def bloquear(self):
        self.estado = 'BLOQUEADO'

    def desbloquear(self):
        self.tempo_bloqueado = 0
        self.proxima_instrucao()
        self.estado = 'PRONTO'
```

4.6.2 Construção do Diagrama de Gantt e Cálculo de Estatísticas

O simulador disponibiliza a classe `stats`, responsável pelo cálculo das métricas descritas no Capítulo 4.5, além da geração do diagrama de Gantt. Essa função pode ser customizada para incluir pontos de interesse definidos pelo usuário, como início ou troca de processos.

4.6.3 Gerador de arquivo de entrada

Como parte do projeto, foi desenvolvido um gerador de arquivos de entrada para facilitar a execução de múltiplas simulações e assegurar a integridade dos dados gerados. Esta classe recebe os seguintes parâmetros:

- **qtd_processos**: número de processos a serem criados para a simulação.
- **start_min**: tempo mínimo de início dos processos, valor obrigatório maior que zero.
- **start_max**: tempo máximo de início dos processos; por exemplo, se definido como 10, nenhum processo iniciará após este instante.
- **max_blocks**: limite para o número de instruções de bloqueio do processo. Considerando que o simulador utiliza apenas três tipos de instrução, esse parâmetro define indiretamente o número total de instruções.
- **max_block_dur**: duração máxima, em *ticks*, de uma instrução de bloqueio.
- **max_cpu**: número total de *ticks* de execução do processo antes da finalização.

Exemplo de geração de novos processos na Listagem 3

Listagem 3 – Execução do gerador de processos

```
1 $ python3 .\core\processos\gerador.py
```

5 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho apresentou o desenvolvimento de um simulador modular voltado ao ensino dos conceitos fundamentais da disciplina de Sistemas Operacionais, com ênfase na simulação de políticas de escalonamento.

A arquitetura proposta visa simplificar a implementação e possibilitar a comparação direta de diferentes políticas aplicadas a um mesmo cenário. A utilização de programação orientada a objetos permite a extensão futura do sistema. Com a estrutura desenvolvida, estudantes e docentes podem testar diversas estratégias de escalonamento e analisar seu impacto por meio de métricas e diagramas gerados automaticamente, promovendo uma compreensão prática dos conceitos estudados.

Como trabalhos futuros, destacam-se a implementação de módulos adicionais, como gerenciamento de memória e disco, além da integração de uma interface gráfica para aprimorar a usabilidade em contextos educacionais.

REFERÊNCIAS

ATKIN, B.; SIRER, E. G. Portos: An educational operating system for the post-pc environment. **SIGCSE Bull.**, ACM New York, NY, USA v. 34, n. 1, p. 116–120, fev. 2002. ISSN 0097-8418. Disponível em: http://doi.acm.org/10.1145/563517.563384.

CHRISTOPHER, W. A.; PROCTER, S. J.; ANDERSON, T. E. The nachos instructional operating system. *In*: PROCEEDINGS OF THE USENIX WINTER 1993 CONFERENCE PROCEEDINGS ON USENIX WINTER 1993 CONFERENCE PROCEEDINGS. 1993, Berkeley, CA, USA. **Anais [...]** Berkeley, CA, USA: USENIX Association, 1993. p. 4–4. Disponível em: http://dl.acm.org/citation.cfm?id=1267303.1267307.

CRUZ, E. H. M.; SILVA, V. P.; GONÇALVES, R. A. L. Sistema operacional integrado simulado: Módulo de entrada e saída. *In*: ANAIS DA ERI - ESCOLA REGIONAL DE INFORMÁTICA. 2007, Guarapuava, PR. **Anais [...]** Guarapuava, PR: ERI - Escola Regional de Informática, 2007.

GONÇALVES, R. A.; GONÇALVES, R. A. d. L.; MARTINI, J. A. Sistema operacional integrado simulado: Projeto e implementação do módulo processador. *In*: ANAIS DA XIII ESCOLA REGIONAL DE INFORMÁTICA DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO (SBC) SUL PARANÁ. 1., 2006, Bandeirantes, Paraná. **Anais [...]** Bandeirantes, Paraná: FFALM/SBC, 2006. p. 1–10.

GONÇALVES, R. A. *et al.* Sistema operacional simulado: Ferramenta para o ensino de graduação. *In*: ANAIS DO XXIV CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO / XII WEI - WORKSHOP DE EDUCAÇÃO EM INFORMÁTICA. 1., 2004, Salvador, Bahia. **Anais [...]** Salvador, Bahia: SBC - Sociedade Brasileira de Computação, 2004. p. 1–1.

HOFSTEIN, A.; LUNETTA, V. N. The role of the laboratory in science teaching: Neglected aspects of research. **Review of Educational Research**, v. 52, n. 2, p. 201–217, 1982.

MACHADO, F. B.; MAIA, L. P. Um framework construtivista no aprendizado de sistemas operacionais-uma proposta pedagógica com o uso do simulador sosim. *In*: XII WORKSHOP DE EDUCAÇÃO EM COMPUTAÇÃO (WEI), XXIV CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO (SBC), SALVADOR, BA. 2004. **Anais [...]** [*S.l.*: *s.n.*], 2004.

MAIA, L. P.; PACHECO, A. C. A simulator supporting lectures on operating systems. *In*: 33RD ANNUAL FRONTIERS IN EDUCATION, 2003. FIE 2003. 2., 2003. **Anais [...]** [*S.l.*: *s.n.*], 2003. p. F2C–13–17 Vol.2. ISSN 0190-5848.

PEARSON; TANENBAUM, A. **Modern Operating Systems**. Pearson College Division, 2008. ISBN 9780135013014. Disponível em: https://books.google.com.br/books?id=UoTqRgAACAAJ.

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. **Operating System Concepts Essentials**. 2nd. ed. [*S.l.*]: Wiley Publishing, 2013. ISBN 1118804929, 9781118804926.

TANENBAUM, A. S. **Minix 3**, . 2017. Disponível em: http://www.minix3.org/.

APÊNDICE A – EXEMPLO DE IMPLEMENTAÇÃO DA POLITICA FIFO

Neste apêndice encontram-se:

- Como executar o simulador (Listagem 4);
- A implementação da politica de escalonamento FIFO (Listagem 5);
- Um exemplo de arquivo de entrada (Listagem 6);
- O conteúdo do arquivo de saída da simulação (Listagem 7); e
- Um Diagrama de Gantt, gerado pelo simulador (Figura 9).

Listagem 4 – Execução do módulo

```
$ python3 .\modulos\gerenciador_de_processos\gerenciador_processos.py
```

Na Listagem 5, nota-se que ao estender a classe `politicaGP`, torna-se obrigatória a implementação dos sete métodos definidos na superclasse. No entanto, no caso da política FIFO, nenhuma ação é necessária durante as rotinas de *tick*, e inicializar, sendo suficiente declarar os métodos correspondentes como uma função vazia.

Listagem 5 – Implementação politica FIFO

```
from modulos.gerenciador_de_processos.politicaGP import politicaGP

class Fifo(politicaGP):

    def inicializar(self, bcp_dados, params):
        pass

    def selecionar_proximo(self):
        return self.fila_prontos[0] if self.fila_prontos else None

    def bloquear(self, processo):
        self.fila_bloqueados.append(processo.pid)
        self.fila_prontos.remove(processo.pid)
        return True

    def desbloquear(self,processo):
        self.fila_prontos.append(processo)
        self.fila_bloqueados.remove(processo)

    def iniciar (self, processo):
        self.fila_prontos.append(processo)

    def finalizar (self,processo):
        self.fila_prontos.remove(processo)

    def tick(self,processo):
        pass
```

Listagem 6 – Arquivo de entrada

```
gp:
  config:
    nome: gerenciador de processos
    politica:
      nome: fifo
      params: null
  processos:
    pid 1:
      instrucoes:
      - 1 start
      - 2 block 1
      - 6 end
    pid 2:
      instrucoes:
      - 1 start
      - 4 block 2
      - 7 block 2
      - 9 end
    pid 3:
      instrucoes:
      - 1 start
      - 1 block 2
      - 4 block 2
      - 7 block 2
      - 10 end
```

Listagem 7 – Arquivo de Saída

```
=== Relatório Final ===
Chaveamentos: 9
Tempo Médio de Retorno: 19.0
Tempo Médio de Espera: 7.0
Vazão por intervalo de 1000 ticks:
  0-999: 3 processo(s)
Termino: [1, 2, 3]
```

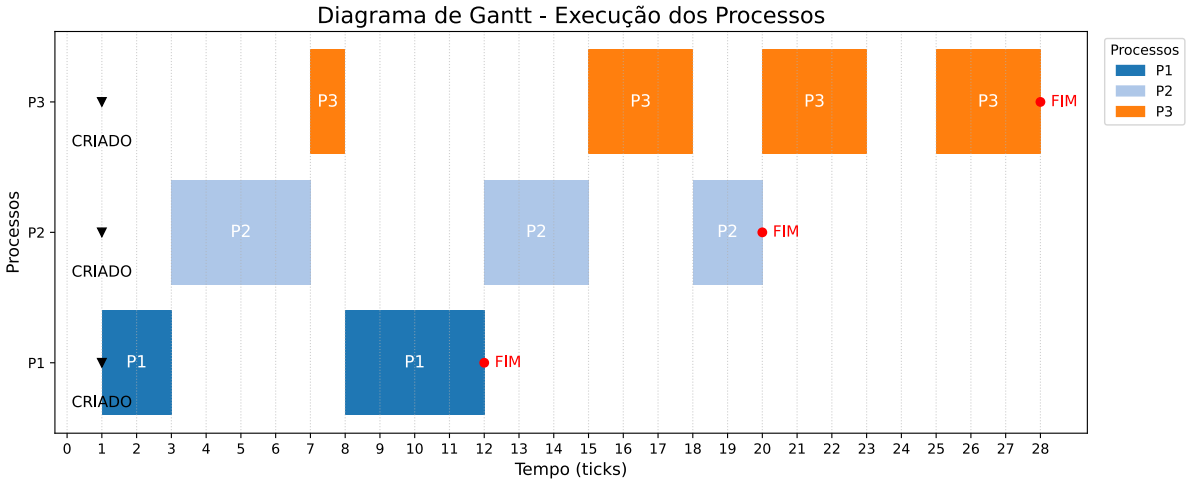

Figura 9 – Diagrama de Gantt