

**Sistemas Operacionais**  
**Curso: Engenharia da computação**  
**Prof.: Maurício Acconcia Dias**  
**Data: 22/09/2025**

**Simulador Sistema Operacional**

**Caroline da Silva Grizante Ra:114105**  
**Emilly Emanuely R. dos Santos Ra:114095**  
**Marcela Lovatto Ra:113626**

**Fundação Hermínio Ometto**  
**Araras-SP**

## Sumário

1. Introdução.....	pág.02
2. Objetivos.....	pág.02
3. Metodologia e Desenvolvimento.....	pág.02
3.1 Estrutura do Projeto.....	pág.02
3.2 Classes Principais.....	pág.03
3.3 Algoritmos de Escalonamento.....	pág.05
3.4 Gerenciamento de Memória.....	pág.05
4. Diagrama UML.....	pág.06
5. Códigos.....	pág.06
6. Imagens.....	pág.22
7. Conclusão.....	pág.27
8. Referências Bibliográficas.....	pág.27

## **1. Introdução**

O projeto SimuladorSO é uma simulação de um sistema operacional simples, implementado em C# com uma arquitetura dividida em camadas. O sistema permite o gerenciamento de instruções, alocação de memória e execução com base em diferentes algoritmos de escalonamento.

A solução é composta por três projetos:

- SimuladorInterface (WPF): Interface gráfica que permite interação com o usuário por meio de botões, menus e formulários.
- SimuladorLogica (Class Library): Contém toda a lógica de negócio do sistema, como modelos, algoritmos de escalonamento e gerenciamento de memória.
- SimuladorSO (Solução): Projeto principal que organiza os dois módulos acima.

O simulador oferece ao usuário funcionalidades para visualizar os processos, adicionar ou remover processos, executar algoritmos de escalonamento, alocar memória e calcular o tempo total de execução do sistema. A proposta é criar um ambiente didático, em que estudantes e interessados em computação possam compreender, de forma prática, o comportamento de um sistema operacional, sem a complexidade de um sistema real.

## **2. Objetivos**

O principal objetivo do SimuladorSO é fornecer uma ferramenta que permita a compreensão dos mecanismos básicos de um sistema operacional. Através da simulação, o usuário consegue observar como diferentes algoritmos de escalonamento impactam a ordem de execução das threads e como a memória é alocada para cada processo. Além disso, o projeto busca demonstrar a importância do gerenciamento de recursos, da organização de processos e da priorização de tarefas em um ambiente controlado.

## **3. Desenvolvimento**

### **3.1 Estrutura do Projeto**

O projeto foi desenvolvido em C# e simula de forma simplificada um sistema operacional. A solução é composta por classes que representam processos, threads, memória e algoritmos de escalonamento, além de um sistema que permite interação com o usuário por meio de um menu.

Essa estrutura segue o paradigma da Programação Orientada a Objetos (POO), onde cada classe possui atributos e métodos que encapsulam seu comportamento.

## 3.2 Classes Principais

### Classe Processo

#### Atributos

- string ProcessId → Identificador do processo.
- List<ThreadSO> Threads → Lista de threads pertencentes ao processo.

#### Métodos

- override string ToString() → Retorna uma representação textual do processo.

### Classe Thread

#### Atributos

- string ThreadId → Identificador da thread.
- int TempoExecucao → Tempo necessário para execução.
- int EnderecoMemoria → Endereço de memória associado.
- int Prioridade → Prioridade da thread.

#### Métodos

- override string ToString() → Retorna uma representação textual da thread.

### Classe GerenciadorProcessos

#### Atributos

- List<Processo> Processos → Lista de processos ativos no sistema.

#### Métodos

- void Adicionar(Processo processo) → Adiciona um processo à lista.
- void Remover(string processId) → Remove processos com o ID informado.
- void Listar() → Exibe todos os processos e suas threads.
- int TempoExecucaoTotal() → Calcula o tempo total de execução de todas as threads.

### Classe Escalonador

#### Atributos

- GerenciadorProcessos \_gerenciador → Referência ao gerenciador de processos.

## Métodos

- void FCFS() → Executa processos no algoritmo First Come, First Served.
- void SJF() → Executa processos no algoritmo Shortest Job First.
- void RR(int quantum) → Executa processos no algoritmo Round Robin.

## Classe Memoria

### Atributos

- int Endereco → Endereço de memória alocado.
- string ProcessId → Identificador do processo que ocupa o endereço.
- string ThreadId → Identificador da thread que ocupa o endereço.

## Classe GerenciadorMemoria

### Atributos

- int TamanhoTotal → Tamanho total disponível de memória.
- List<Memoria> Alocaoes → Lista de endereços alocados.

### Métodos

- void Alocar(Processo processo) → Registra as threads de um processo nos endereços de memória.
- void Mostrar() → Exibe uma tabela com os endereços e suas ocupações.

## Classe SistemaOperacional

### Atributos

- GerenciadorProcessos gerenciador → Responsável por manipular os processos.
- GerenciadorMemoria memoria → Responsável pelo gerenciamento da memória.

### Métodos

- void CarregarArquivo(string caminho) → Carrega processos a partir de um arquivo texto.
- void Menu() → Exibe e controla as opções do menu principal.

## Classe Program

### Métodos

- static void Main(string[] args) → Ponto de entrada da aplicação, inicializa o sistema e exibe o menu.

### 3.3 Algoritmos de Escalonamento

O sistema implementa três políticas de escalonamento de threads:

- FCFS (First Come, First Served): os processos são executados na ordem de chegada, de forma sequencial.
- SJF (Shortest Job First): os processos com menor tempo de execução são priorizados.
- RR (Round Robin): utiliza um quantum de tempo fixo, alternando a execução entre os processos ativos, garantindo justiça no uso do processador.

Cada algoritmo foi implementado como um método da classe Escalonador, permitindo simulação prática.

Exemplo RR:

Ciclo	Execução
1	P1-T1 (3), P2-T1 (3), P3-T1 (3)
2	P1-T1 (2), P2-T1 (1), P3-T1 (3)
3	P3-T1 (1)

### 3.4 Gerenciamento de Memória

A administração do espaço da memória é realizada pela classe GerenciadorMemoria, que recebe como parâmetro o tamanho total da memória durante sua inicialização. Essa classe mantém uma lista de objetos Memoria, representando as alocações feitas ao longo da simulação.

O método Alocar() percorre todas as threads de um processo e registra o endereço de memória ocupado por cada uma delas, criando instâncias da classe Memoria. Já o método Mostrar() exibe o estado atual da memória em forma de tabela, indicando os endereços ocupados e os respectivos processos e threads.

Dessa forma, o gerenciamento de memória oferece uma visão clara da distribuição dos recursos, permitindo que o usuário compreenda de maneira simples como cada thread ocupa uma posição específica na memória. Apesar de simplificada em relação a um sistema operacional real, essa implementação é suficiente para ilustrar os conceitos fundamentais de alocação estática e visualização do uso da memória.

## 4. Diagrama UML

O diagrama UML elaborado para este projeto representa as principais classes que compõem o simulador de sistema operacional, suas responsabilidades e relações. A classe Processo contém o identificador do processo (ProcessId) e uma lista de threads associadas. Ela é responsável por agrupar e organizar as threads que pertencem ao mesmo processo.

A classe ThreadSO representa cada thread individualmente. Seus atributos incluem o identificador (ThreadId), o tempo de execução (TempoExecucao), o endereço de memória (EnderecoMemoria) e a prioridade (Prioridade). Esta classe é fundamental para caracterizar a carga de trabalho de cada processo. O gerenciamento dos processos é realizado pela classe GerenciadorProcessos, que mantém uma lista de processos ativos. Ela oferece métodos para adicionar, remover e listar processos, além de calcular o tempo total de execução do sistema.

Para lidar com as políticas de escalonamento, foi definida a classe Escalonador, que possui referência ao GerenciadorProcessos e implementa três algoritmos: FCFS, SJF e Round Robin (RR). Esta classe é responsável por simular a forma como a CPU organiza a execução das threads.

A classe Memoria abstrai a alocação de um endereço de memória, indicando qual processo e thread ocupam determinada posição. Complementarmente, a classe GerenciadorMemoria administra o espaço total disponível e possui métodos para registrar alocações (Alocar) e exibi-las (Mostrar).

A classe SistemaOperacional integra todos os componentes. Ela realiza a leitura do arquivo de entrada, inicializa os processos, e oferece o menu interativo para que o usuário possa escolher entre visualizar processos, executar algoritmos de escalonamento ou verificar a alocação de memória.

## 5. Código

### Parte Lógica

[//Escalonador.cs](#)

```
namespace SistemaLogica
{
    public class Escalonador
    {
        // Guarda a referência para o GerenciadorProcessos
    }
}
```

```

private readonly GerenciadorProcessos _gerenciador;

// Construtor Escalonador
public Escalonador(GerenciadorProcessos gerenciador)
{
    _gerenciador = gerenciador;
}

// A thread que chega primeiro é escalonada primeiro
public void FCFS()
{
    var todas = _gerenciador.Processos
        .SelectMany(p => p.Threads.Select(t => new { Processo = p.ProcessoId, Thread = t }))
        .ToList();

    // Cabeçalho da tabela
    Console.WriteLine($"{ "Ordem",-7} | { "Processo-Thread",-20} | { "Tempo de Execução",-20}");
    Console.WriteLine(new string('-', 55));

    int ordem = 1;
    // Executa cada thread exatamente na ordem da lista
    foreach (var item in todas)
    {
        Console.WriteLine($"{ordem,-7} | { $"{item.Processo}-{item.Thread.ThreadId",-20} | {
item.Thread.TempoExecucao,-20}");
        ordem++;
    }
}

// A thread com menor tempo de execução é escalonada primeiro
public void SJF()
{
    // Monta uma lista com todas as threads e ordena pelo tempo de execução
    var todas = _gerenciador.Processos
        .SelectMany(p => p.Threads.Select(t => new { Processo = p.ProcessoId, Thread = t }))
        .OrderBy(x => x.Thread.TempoExecucao)
        .ToList();

    // Cabeçalho da tabela
    Console.WriteLine($"{ "Ordem",-7} | { "Processo-Thread",-20} | { "Tempo de Execução",-20}");
    Console.WriteLine(new string('-', 55));

    int ordem = 1;
    // Percorre a lista já ordenada e imprime na ordem
    foreach (var item in todas)
    {

```



```

        Console.WriteLine($"{ordem,-7} | ${"{item.Processo}-{item.Thread.ThreadId}",-20} |
{item.Thread.TempoExecucao,-20}");
        ordem++;
    }
}

// A ideia é dar um "pedaço de tempo" igual para cada processo/thread
public void RR(int quantum = 3) // Foi definido um quantum padrão de 3 mas pode ser alterado
ao chamar o método

{
    // Monta uma lista temporária com todas as threads de todos os processos
    // Cada item da lista tem o ID do processo e a referência da thread
    // Isso permite que o texto de execução total seja preservado
    var fila = _gerenciador.Processos
        .SelectMany(p => p.Threads.Select(t => new ThreadExecucaoRR
        {
            ProcessId = p.ProcessId,
            ThreadId = t.ThreadId,
            TempoRestante = t.TempoExecucao // Copia o tempo de execução original
        }))
        .ToList();

    if (!fila.Any())
    {
        Console.WriteLine("Nenhuma thread para executar.");
        return;
    }

    // Cabeçalho da tabela
    Console.WriteLine($"Execução RR (Quantum = {quantum}):");
    Console.WriteLine($"{{"Processo-Thread",-20} | {"Executou",-10} | {"Restante",-10}}");
    Console.WriteLine(new string('-', 45));

    // Continua enquanto houver alguma thread com tempo restante
    while (fila.Any(t => t.TempoRestante > 0))
    {
        // Percorre a fila de threads
        foreach (var item in fila)
        {
            // Se a thread ainda precisa executar
            if (item.TempoRestante > 0)
            {
                // Calcula o tempo que será executado nesta rodada
                int tempoExecutar = Math.Min(quantum, item.TempoRestante);
            }
        }
    }
}

```

```

        // Subtrai o tempo executado
        item.TempoRestante -= tempoExecutar;

        // Exibe a execução em formato de tabela
        Console.WriteLine($"{item.ProcessoId}-{item.ThreadId}",-20} | {tempoExecutar,-10} | {item.TempoRestante,-10}");
    }
}
}
}
}
}
}
}
}
}
}

```

### //GerenciadorMemoria.cs

```

namespace SistemaLogica
{
    public class GerenciadorMemoria
    {
        // Guarda o tamanho total da memória
        public int TamanhoTotal { get; private set; }
        // Armazena todas as alocações de memória feitas
        public List<Memoria> Alocacoes { get; private set; } = new List<Memoria>();

        // Construtor da classe
        public GerenciadorMemoria(int tamanho)
        {
            TamanhoTotal = tamanho;
        }

        // Método que aloca todas as threads de um processo na memória
        public void Alocar(Processo processo)
        {
            foreach (var t in processo.Threads)
            {
                // Verifica se já existe uma alocação para esta thread específica
                bool jaAlocada = Alocacoes.Any(m => m.ProcessoId == processo.ProcessoId && m.ThreadId == t.ThreadId);

                if (!jaAlocada)
                {
                    // Cria um objeto Memoria e adiciona na lista de alocações
                    Alocacoes.Add(new Memoria
                    {
                        // Endereço da thread
                        Endereco = t.EnderecoMemoria,
                        ProcessoId = processo.ProcessoId,

```

```

        ThreadId = t.ThreadId
    });
}
}

// Método que remove todas as alocações de memória de um processo
public void Desalocar(string processold)
{
    Alocacoes.RemoveAll(m => m.Processold == processold);
}

// Método que mostra a alocação de memória no WPF
// Adicionei uma verificação para o caso de não haver alocações
public void Mostrar()
{
    if (Alocacoes.Count == 0)
    {
        Console.WriteLine("Nenhuma memória alocada.");
        return;
    }

    // Cabeçalho da tabela
    Console.WriteLine($"{ "Endereço",-10} | { "Processo",-10} | { "Thread",-10}");
    Console.WriteLine(new string('-', 36));

    // Ordenado por endereço para uma visualização mais limpa
    foreach (var m in Alocacoes.OrderBy(a => a.Endereco))
    {
        Console.WriteLine($"{m.Endereco,-10} | {m.Processold,-10} | {m.ThreadId,-10}");
    }
}
}
}

```

#### **//GerenciadorProcessos.cs**

```

namespace SistemaLogica
{
    public class GerenciadorProcessos
    {
        public List<Processo> Processos { get; private set; } = new List<Processo>();

        // Adiciona um processo somente se ele ainda não existe
        public void Adicionar(Processo processo)
        {
            if (!Existe(processo.Processold))

```

```

    {
        Processos.Add(processo);
    }
    else
    {
        // Se o processo já existe, apenas adiciona as threads novas
        var existente = Buscar(processo.ProcessoId);
        foreach (var t in processo.Threads)
        {
            if (!existente.Threads.Any(th => th.ThreadId == t.ThreadId))
                existente.Threads.Add(t);
        }
    }
}

// Remove processo pelo ID
public void Remover(string processId)
{
    Processos.RemoveAll(p => p.ProcessoId == processId);
}

// Busca um processo pelo ID
public Processo? Buscar(string processId)
{
    return Processos.FirstOrDefault(p => p.ProcessoId == processId);
}

// Verifica se o processo existe
public bool Existe(string processId)
{
    return Processos.Any(p => p.ProcessoId == processId);
}

// Lista todos os processos e suas threads
public void Listar()
{
    if (Processos.Count == 0)
    {
        Console.WriteLine("Nenhum processo cadastrado.");
        return;
    }

    // Cabeçalho da tabela
    Console.WriteLine($"{{"Processo",-10} | {"Thread",-10} | {"Tempo",-7} | {"Memória",-10} | {"Prioridade",-10}}");
    Console.WriteLine(new string('-', 60)); // Linha separadora

```

```

foreach (var p in Processos)
{
    if (p.Threads.Count == 0)
    {
        Console.WriteLine($"{p.ProcessId,-10} | Nenhuma thread.");
        continue;
    }
    foreach (var t in p.Threads)
    {
        // Corpo da tabela com colunas alinhadas
        Console.WriteLine($"{p.ProcessId,-10} | {t.ThreadId,-10} | {t.TempoExecucao,-7} |
{t.EnderecoMemoria,-10} | {t.Prioridade,-10}");
    }
}

// Calcula o tempo total de execução de todas as threads de todos os processos
public int TempoExecucaoTotal()
{
    return Processos.Sum(p => p.Threads.Sum(t => t.TempoExecucao));
}

public List<Processo> ObterTodos()
{
    return Processos; // supondo que você guarda numa List<Processo> chamada processos
}
}
}

```

#### //Memoria.cs

```

namespace SistemaLogica
{
    public class Memoria
    {
        // Endereço da memória (ex: posição 1000)
        public int Endereco { get; set; }

        // ID do processo que está ocupando o endereço de memória
        public string ProcessId { get; set; }

        // Identificador da thread que está usando o endereço
        public string ThreadId { get; set; }
    }
}

```

```
//MyThread.cs
namespace SistemaLogica
{
    public class MyThread
    {
        // ID da thread
        public string ThreadId { get; set; } = string.Empty;

        // Tempo de execução da thread
        public int TempoExecucao { get; set; }

        // Endereço de memória onde a thread está alocada
        public int EnderecoMemoria { get; set; }

        // Prioridade da thread (quanto menor o número, maior a prioridade)
        public int Prioridade { get; set; }

        public override string ToString()
        {
            return $"Thread {ThreadId} | Tempo: {TempoExecucao} | Endereço: {EnderecoMemoria} | Prioridade: {Prioridade}";
        }
    }
}
```

#### **//Processo.cs**

```
namespace SistemaLogica
{
    public class Processo
    {
        // ID do processo (ex: P1, P2, etc.)
        public string ProcessId { get; set; } = string.Empty;

        // Lista de threads associadas a este processo
        public List<MyThread> Threads { get; set; } = new List<MyThread>();

        public override string ToString()
        {
            // Retorna uma string representando o processo e a quantidade de threads que ele possui
            return $"Processo {ProcessId} - {Threads.Count} threads";
        }
    }
}
```

## //SistemaOperacional.cs

```
namespace SistemaLogica
{
    public class SistemaOperacional
    {
        // Gerenciador de processos (adicionar, remover, listar)
        public readonly GerenciadorProcessos GerenciadorProcessos = new GerenciadorProcessos();

        // Gerenciador de memória, aqui inicializado com 5000 blocos/endereço
        public readonly GerenciadorMemoria GerenciadorMemoria = new GerenciadorMemoria(5000);

        // Escalonador de processos
        public readonly Escalonador Escalonador;

        public SistemaOperacional()
        {
            Escalonador = new Escalonador(GerenciadorProcessos);
        }

        public void CarregarArquivo(string caminho)
        {
            // Lê todas as linhas do arquivo
            var linhas = File.ReadAllLines(caminho);
            foreach (var linha in linhas)
            {
                // Ignora comentários (#) e linhas vazias
                if (linha.StartsWith("#") || string.IsNullOrEmpty(linha)) continue;

                // Divide a linha pelos ; para extrair os dados
                var partes = linha.Split(';');
                string pld = partes[0];
                string tld = partes[1];
                int tempo = int.Parse(partes[2]);
                int endereco = int.Parse(partes[3]);
                int prioridade = int.Parse(partes[4]);

                // Procura se o processo já existe na lista
                var processo = GerenciadorProcessos.Processos.Find(p => p.Processold == pld);
                if (processo == null)
                {
                    // Se não existe, cria um novo processo e adiciona ao gerenciador
                    processo = new Processo { Processold = pld };
                    GerenciadorProcessos.Adicionar(processo);
                }

                // Adiciona a thread ao processo
            }
        }
    }
}
```

```

        processo.Threads.Add(new MyThread
        {
            ThreadId = tld,
            TempoExecucao = tempo,
            EnderecoMemoria = endereco,
            Prioridade = prioridade
        });
    }

    // Após carregar todos os processos, aloca memória para cada um
    foreach (var p in GerenciadorProcessos.Processos)
        GerenciadorMemoria.Alocar(p);
    }
}

```

#### //ThreadExecucaoRR.cs

```

namespace SistemaLogica
{
    // Classe auxiliar para Escalonador.cs
    public class ThreadExecucaoRR
    {
        // Ela guardará o estado de cada thread durante a execução do RR
        // Permtindo que o Escalonador saiba quanto tempo resta para cada thread ser executada
        public string ProcessId { get; set; }
        public string ThreadId { get; set; }
        public int TempoRestante { get; set; }
    }
}

```

## Parte da Interface Visual

#### //MainWindow.xaml

```

<Window x:Class="SimuladorSO.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:SimuladorSO"
        mc:Ignorable="d"
        Title="Simulador Sistema Operacional" Height="700" Width="1000"
        Icon="/Imagem/Icone.ico">
    <Grid Background="#E0E0E0">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="250"/>
            <ColumnDefinition Width="*/>

```



```

</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="*/>
</Grid.RowDefinitions>

<StackPanel Grid.Column="0" Grid.Row="0" Grid.RowSpan="2" Margin="10">
    <Border BorderBrush="Gray" BorderThickness="1" CornerRadius="5" Padding="10"
Margin="0,0,0,10">
        <StackPanel>
            <TextBlock Text="Gerenciamento de Processos" FontWeight="Bold" Margin="0,0,0,5"/>
            <Button Content="Ver Processos e Threads" Name="btnVerProcessos"
Click="btnVerProcessos_Click" Margin="0,5,0,0"/>
            <Separator Margin="0,10"/>
            <TextBlock Text="Adicionar Novo Processo" FontWeight="Bold" Margin="0,0,0,5"/>

            <Grid Margin="0,5,0,0">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="Auto"/>
                    <ColumnDefinition Width="*/>
                </Grid.ColumnDefinitions>
                <Grid.RowDefinitions>
                    <RowDefinition Height="Auto"/>
                    <RowDefinition Height="Auto"/>
                    <RowDefinition Height="Auto"/>
                    <RowDefinition Height="Auto"/>
                    <RowDefinition Height="Auto"/>
                </Grid.RowDefinitions>

                <TextBlock Text="ID do Processo:" Grid.Row="0" Grid.Column="0"
VerticalAlignment="Center" Margin="0,0,5,0"/>
                <TextBox Name="txtProcessoIdAdd" Grid.Row="0" Grid.Column="1"/>

                <TextBlock Text="ID da Thread:" Grid.Row="1" Grid.Column="0"
VerticalAlignment="Center" Margin="0,5,5,0"/>
                <TextBox Name="txtThreadIdAdd" Grid.Row="1" Grid.Column="1" Margin="0,5,0,0"/>

                <TextBlock Text="Tempo de Execução:" Grid.Row="2" Grid.Column="0"
VerticalAlignment="Center" Margin="0,5,5,0"/>
                <TextBox Name="txtTempoExecucaoAdd" Grid.Row="2" Grid.Column="1"
Margin="0,5,0,0"/>

                <TextBlock Text="Endereço:" Grid.Row="3" Grid.Column="0"
VerticalAlignment="Center" Margin="0,5,5,0"/>
                <TextBox Name="txtEnderecoMemoriaAdd" Grid.Row="3" Grid.Column="1"
Margin="0,5,0,0"/>
            </Grid>
        </StackPanel>
    </Border>
</StackPanel>

```

```

        <TextBlock Text="Prioridade:" Grid.Row="4" Grid.Column="0"
VerticalAlignment="Center" Margin="0,5,5,0"/>
        <TextBox Name="txtPrioridadeAdd" Grid.Row="4" Grid.Column="1" Margin="0,5,0,0"/>
    </Grid>
    <Button Content="Adicionar Processo" Name="btnAdicionarProcesso"
Click="btnAdicionarProcesso_Click" Margin="0,10,0,0"/>
    <Separator Margin="0,10"/>
    <TextBlock Text="Remover Processo" FontWeight="Bold" Margin="0,0,0,5"/>
    <TextBox Name="txtProcessoIdRemover" Tag="ID do Processo" Margin="0,5,0,0"/>
    <Button Content="Remover Processo" Name="btnRemoverProcesso"
Click="btnRemoverProcesso_Click" Margin="0,10,0,0"/>
</StackPanel>
</Border>

    <Border BorderBrush="Gray" BorderThickness="1" CornerRadius="5" Padding="10"
Margin="0,0,0,10">
    <StackPanel>
        <TextBlock Text="Escalonamento" FontWeight="Bold" Margin="0,0,0,5"/>
        <Button Content="FCFS" Name="btnFCFS" Click="btnFCFS_Click" Margin="0,5,0,0"/>
        <Button Content="SJF" Name="btnSJF" Click="btnSJF_Click" Margin="0,5,0,0"/>
        <StackPanel Orientation="Horizontal" Margin="0,5,0,0">
            <TextBlock Text="RR Quantum: " VerticalAlignment="Center"/>
            <TextBox Name="txtQuantumRR" Width="50" Text="3"/>
            <Button Content="RR" Name="btnRR" Click="btnRR_Click" Margin="5,0,0,0"/>
        </StackPanel>
    </StackPanel>
</Border>

    <Border BorderBrush="Gray" BorderThickness="1" CornerRadius="5" Padding="10"
Margin="0,0,0,10">
    <StackPanel>
        <TextBlock Text="Memória e Tempo" FontWeight="Bold" Margin="0,0,0,5"/>
        <Button Content="Alocação de Memória" Name="btnAlocacaoMemoria"
Click="btnAlocacaoMemoria_Click" Margin="0,5,0,0"/>
        <Button Content="Tempo de Execução Total" Name="btnTempoExecucao"
Click="btnTempoExecucao_Click" Margin="0,5,0,0"/>
    </StackPanel>
</Border>
</StackPanel>

<Grid Grid.Column="1" Grid.Row="0" Grid.RowSpan="2" Margin="10">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*/>
    </Grid.RowDefinitions>

```

```

        <TextBlock Text="Saída do Simulador" FontWeight="Bold" FontSize="16" Margin="0,0,0,5"
Grid.Row="0"/>
        <ScrollView Grid.Row="1" VerticalScrollBarVisibility="Auto">

            <!-- FontFamily="Consolas" a formatação de tabela com espaçamento fixo -->
            <TextBox Name="txtSaida"
                IsReadOnly="True"
                TextWrapping="NoWrap"
                VerticalScrollBarVisibility="Auto"
                HorizontalScrollBarVisibility="Auto"
                Background="White"
                MinHeight="400"
                FontFamily="Consolas"/>
        </ScrollView>
    </Grid>
</Grid>
</Window>

```

#### **//MainWindow.xaml.cs**

```

using SistemaLogica;
using System;
using System.IO;
using System.Windows;

namespace SimuladorSO
{
    public partial class MainWindow : Window
    {
        private readonly SistemaOperacional so;

        public MainWindow()
        {
            InitializeComponent();
            so = new SistemaOperacional();

            // Carrega processos do arquivo na inicialização
            CarregarProcessosDoArquivo();

            txtSaida.Text += "Simulador de SO iniciado.\n";
        }

        private void CarregarProcessosDoArquivo()
        {
            string caminho = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "processos.txt");

            if (!File.Exists(caminho))

```

```

    {
        txtSaida.Text += $"Arquivo de processos '{caminho}' não encontrado.\n";
        return;
    }

    try
    {
        // Usa o método centralizado em SistemaOperacional para carregar e alocar
        so.CarregarArquivo(caminho);
        txtSaida.Text += "Processos carregados com sucesso do arquivo.\n";
    }
    catch (Exception ex)
    {
        txtSaida.Text += $"Erro ao carregar o arquivo de processos: {ex.Message}\n";
    }
}

private void btnVerProcessos_Click(object sender, RoutedEventArgs e)
{
    txtSaida.Text += "\n--- Processos e Threads ---\n";
    StringWriter sw = new StringWriter();
    Console.SetOut(sw);
    so.GerenciadorProcessos.Listar();
    txtSaida.Text += sw.ToString();
    ResetConsoleOutput();
}

private void btnAdicionarProcesso_Click(object sender, RoutedEventArgs e)
{
    try
    {
        string processold = txtProcessoldAdd.Text;
        if (string.IsNullOrEmpty(processold))
        {
            throw new Exception("O ID do Processo é obrigatório.");
        }

        // Busca o processo, se não existir, cria um novo
        var processo = so.GerenciadorProcessos.Buscar(processold);
        if (processo == null)
        {
            processo = new Processo { Processold = processold };
            so.GerenciadorProcessos.Adicionar(processo);
        }

        // Cria a nova thread com os dados da UI

```

```

var novaThread = new MyThread
{
    ThreadId = txtThreadIdAdd.Text,
    TempoExecucao = int.Parse(txtTempoExecucaoAdd.Text),
    EnderecoMemoria = int.Parse(txtEnderecoMemoriaAdd.Text),
    Prioridade = int.Parse(txtPrioridadeAdd.Text)
};

// Adiciona a thread ao processo e aloca sua memória
processo.Threads.Add(novaThread);
so.GerenciadorMemoria.Alocar(processo); // O método Alocar já previne duplicatas

txtSaida.Text += $"Thread {novaThread.ThreadId} adicionada ao Processo
{processo.ProcessId} com sucesso.\n";

// Limpa os campos de texto
txtProcessIdAdd.Clear();
txtThreadIdAdd.Clear();
txtTempoExecucaoAdd.Clear();
txtEnderecoMemoriaAdd.Clear();
txtPrioridadeAdd.Clear();
}
catch (FormatException)
{
    txtSaida.Text += "\nErro ao adicionar: verifique se os campos numéricos (Tempo, Endereço,
Prioridade) contêm apenas números.\n";
}
catch (Exception ex)
{
    txtSaida.Text += $"Erro ao adicionar processo: {ex.Message}\n";
}
}

private void btnRemoverProcesso_Click(object sender, RoutedEventArgs e)
{
    string id = txtProcessIdRemover.Text;
    if (string.IsNullOrEmpty(id))
    {
        txtSaida.Text += "\nPor favor, insira um ID de processo para remover.\n";
        return;
    }

    // Desaloca a memória associada ao processo
    so.GerenciadorMemoria.Desalocar(id);

    // Remover o processo da lista de processos

```

```

so.GerenciadorProcessos.Remover(id);

        txtSaida.Text += $"Processo {id} e suas alocações de memória foram removidos (se
existiam).\n";
        txtProcessoRemover.Clear();
    }

private void btnFCFS_Click(object sender, RoutedEventArgs e)
{
    txtSaida.Text += "\n--- Escalonamento FCFS ---\n";
    StringWriter sw = new StringWriter();
    Console.SetOut(sw);
    so.Escalonador.FCFS();
    txtSaida.Text += sw.ToString();
    ResetConsoleOutput();
}

private void btnSJF_Click(object sender, RoutedEventArgs e)
{
    txtSaida.Text += "\n--- Escalonamento SJF ---\n";
    StringWriter sw = new StringWriter();
    Console.SetOut(sw);
    so.Escalonador.SJF();
    txtSaida.Text += sw.ToString();
    ResetConsoleOutput();
}

private void btnRR_Click(object sender, RoutedEventArgs e)
{
    try
    {
        int quantum = int.Parse(txtQuantumRR.Text);
        txtSaida.Text += $"Escalonamento RR (Quantum = {quantum}) ---\n";
        StringWriter sw = new StringWriter();
        Console.SetOut(sw);
        so.Escalonador.RR(quantum);
        txtSaida.Text += sw.ToString();
        ResetConsoleOutput();
    }
    catch (FormatException)
    {
        txtSaida.Text += "\nErro: O valor do Quantum deve ser um número inteiro.\n";
    }
    catch (Exception ex)
    {
        txtSaida.Text += $"Erro no Escalonamento RR: {ex.Message}\n";
    }
}

```

```

    }
}

private void btnAlocacaoMemoria_Click(object sender, RoutedEventArgs e)
{
    txtSaida.Text += "\n--- Alocação de Memória ---\n";
    StringWriter sw = new StringWriter();
    Console.SetOut(sw);
    so.GerenciadorMemoria.Mostrar();
    txtSaida.Text += sw.ToString();
    ResetConsoleOutput();
}

private void btnTempoExecucao_Click(object sender, RoutedEventArgs e)
{
    txtSaida.Text += "\n--- Tempo de Execução Total ---\n";
    int tempoTotal = so.GerenciadorProcessos.TempoExecucaoTotal();
    txtSaida.Text += $"Tempo total de todas as threads: {tempoTotal}\n";
}

private void ResetConsoleOutput()
{
    var standardOutput = new StreamWriter(Console.OpenStandardOutput())
    {
        AutoFlush = true
    };
    Console.SetOut(standardOutput);
}
}
}

```

## 6. Imagens

- ❶ **Imagem:** Interface visual do simulador ao compilar.
- ❷ **Imagem:** Visualizando os processos e threads que estão no arquivo de texto.
- ❸ **Imagem:** Inserindo um novo processo.
- ❹ **Imagem:** Removendo um processo e atualizando a lista.
- ❺ **Imagem:** Execução do escalonador FCFS.
- ❻ **Imagem:** Execução do escalonador SJF.
- ❼ **Imagem:** Execução do escalonador RR com Quantum = 3.
- ❽ **Imagem:** Alocação de memória e tempo de execução finais do simulador.

## 1ª Imagem

**Gerenciamento de Processos**

Ver Processos e Threads

**Adicionar Novo Processo**

ID do Processo:

ID da Thread:

Tempo de Execução:

Endereço:

Prioridade:

Adicionar Processo

**Remover Processo**

ID do Processo:

Remover Processo

**Escalonamento**

FCFS

SJF

RR Quantum: 3

RR

**Saída do Simulador**

Processos carregados com sucesso do arquivo.  
Simulador de Sistema Operacional iniciado!

## 2ª Imagem

**Gerenciamento de Processos**

Ver Processos e Threads

**Adicionar Novo Processo**

ID do Processo:

P5

ID da Thread:

T1

Tempo de Execução:

5

Endereço:

500

Prioridade:

4

Adicionar Processo

**Remover Processo**

ID do Processo:

Remover Processo

**Escalonamento**

FCFS

SJF

RR Quantum: 3

RR

**Saída do Simulador**

Processos carregados com sucesso do arquivo.  
Simulador de Sistema Operacional iniciado!

--- Processos e Threads ---

Processo	Thread	Tempo	Memória	Prioridade
P1	T1	5	100	2
P1	T2	3	101	1
P2	T1	8	200	3
P2	T2	4	201	2
P3	T1	2	300	1
P3	T2	6	301	2
P3	T3	7	302	3
P4	T1	10	400	1



**Simulador Sistema Operacional**

**Gerenciamento de Processos**

Ver Processos e Threads

**Adicionar Novo Processo**

ID do Processo:

ID da Thread:

Tempo de Execução:

Endereço:

Prioridade:

Adicionar Processo

**Remover Processo**

ID do Processo:

Remover Processo

**Escalonamento**

FCFS

SJF

RR Quantum: 2 RR

**Saída do Simulador**

Processos carregados com sucesso do arquivo.  
Simulador de Sistema Operacional iniciado!

--- Processos e Threads ---

Processo	Thread	Tempo	Memória	Prioridade
P1	T1	5	100	2
P1	T2	3	101	1
P2	T1	8	200	3
P2	T2	4	201	2
P3	T1	2	300	1
P3	T2	6	301	2
P3	T3	7	302	3
P4	T1	10	400	1

Thread T1 adicionada ao Processo P5 com sucesso.

--- Processos e Threads ---

Processo	Thread	Tempo	Memória	Prioridade
P1	T1	5	100	2
P1	T2	3	101	1
P2	T1	8	200	3
P2	T2	4	201	2
P3	T1	2	300	1
P3	T2	6	301	2
P3	T3	7	302	3
P4	T1	10	400	1
P5	T1	5	500	4

The screenshot displays the 'Simulador Sistema Operacional' (Operating System Simulator) application window. The interface is divided into several functional areas:

- Gerenciamento de Processos (Process Management):** Located at the top left, it includes a button labeled 'Ver Processos e Threads'.
- Adicionar Novo Processo (Add New Process):** Below process management, this section contains input fields for 'ID do Processo:', 'ID da Thread:', 'Tempo de Execução:', 'Endereço:', and 'Prioridade:', followed by an 'Adicionar Processo' button.
- Remover Processo (Remove Process):** This section features an input field for 'ID do Processo:' and a 'Remover Processo' button.
- Escalonamento (Scheduling):** At the bottom left, it shows two selectable scheduling algorithms, 'FCFS' and 'SJF', each with a corresponding button. Below these is a label 'RR Quantum:' followed by a numeric input field set to '3' and a small icon.

The main area of the simulator, titled 'Saída do Simulador' (Simulator Output), provides real-time feedback:

- A message states: 'Thread T1 adicionada ao Processo P5 com sucesso.' (Thread T1 added to Process P5 successfully).
- A table titled '--- Processos e Threads ---' lists the current state of processes and threads. The table has five columns: 'Processo', 'Thread', 'Tempo', 'Memória', and 'Prioridade'.

Processo	Thread	Tempo	Memória	Prioridade
P1	T1	5	100	2
P1	T2	3	101	1
P2	T1	8	200	3
P2	T2	4	201	2
P3	T1	2	300	1
P3	T2	6	301	2
P3	T3	7	302	3
P4	T1	10	400	1
P5	T1	5	500	4
- Below the first table, another message states: 'Processo P2 e suas alocações de memória foram removidos (se existiam).' (Process P2 and its memory allocations were removed if they existed).
- A second table, also titled '--- Processos e Threads ---', shows the state after the removal of Process P2:

Processo	Thread	Tempo	Memória	Prioridade
P1	T1	5	100	2
P1	T2	3	101	1
P3	T1	2	300	1
P3	T2	6	301	2
P3	T3	7	302	3
P4	T1	10	400	1
P5	T1	5	500	4

## 5ª Imagem

ID da Thread:

Tempo de Execução:

Endereço:

Prioridade:

Adicionar Processo

Remover Processo

ID do Processo:

Remover Processo

Escalonamento

FCFS

SJF

RR Quantum: 3

RR

Memória e Tempo

Alocação de Memória

Tempo de Execução Total

Processo

Thread

Tempo

Memória

Prioridade

P2	T2	4	200	2
P3	T1	2	300	1
P3	T2	6	301	2
P3	T3	7	302	3
P4	T1	10	400	1
P5	T1	5	500	4

Processo P2 e suas alocações de memória foram removidos (se existiam).

--- Processos e Threads ---

Processo	Thread	Tempo	Memória	Prioridade
P1	T1	5	100	2
P1	T2	3	101	1
P3	T1	2	300	1
P3	T2	6	301	2
P3	T3	7	302	3
P4	T1	10	400	1
P5	T1	5	500	4

--- Escalonamento FCFS ---

Ordem	Processo-Thread	Tempo de Execução
1	P1-T1	5
2	P1-T2	3
3	P3-T1	2
4	P3-T2	6
5	P3-T3	7
6	P4-T1	10
7	P5-T1	5

## 6ª Imagem

Ver Processos e Threads

Adicionar Novo Processo

ID do Processo:

ID da Thread:

Tempo de Execução:

Endereço:

Prioridade:

Adicionar Processo

Remover Processo

ID do Processo:

Remover Processo

Escalonamento

FCFS

SJF

RR Quantum: 3

RR

Processo

Thread

Tempo

5	P3-T2	6
6	P3-T3	7
7	P2-T1	8
8	P4-T1	10

Thread T1 adicionada ao Processo P5 com sucesso.

Processo P2 e suas alocações de memória foram removidos (se existiam).

--- Escalonamento FCFS ---

Ordem	Processo-Thread	Tempo de Execução
1	P1-T1	5
2	P1-T2	3
3	P3-T1	2
4	P3-T2	6
5	P3-T3	7
6	P4-T1	10
7	P5-T1	5

--- Escalonamento SJF ---

Ordem	Processo-Thread	Tempo de Execução
1	P3-T1	2
2	P1-T2	3
3	P1-T1	5
4	P5-T1	5
5	P3-T2	6
6	P3-T3	7
7	P4-T1	10

## 7ª Imagem

Simulador Sistema Operacional

ID da Thread:

Tempo de Execução:

Endereço:

Prioridade:

**Adicionar Processo**

**Remover Processo**

ID do Processo:

**Remover Processo**

**Escalonamento**

**FCFS**

**SJF**

RR Quantum:  **RR**

**Memória e Tempo**

**Alocação de Memória**

**Tempo de Execução Total**

**Saída do Simulador**

--- Escalonamento FCFS ---

Ordem	Processo-Thread	Tempo de Execução
1	P1-T1	5
2	P1-T2	3
3	P3-T1	2
4	P3-T2	6
5	P3-T3	7
6	P4-T1	10
7	P5-T1	5

--- Escalonamento RR (Quantum = 3) ---

Execução RR (Quantum = 3):

Processo-Thread	Executou	Restante
P1-T1	3	2
P1-T2	3	0
P3-T1	2	0
P3-T2	3	3
P3-T3	3	4
P4-T1	3	7
P5-T1	3	2
P1-T1	2	0
P3-T2	3	0
P3-T3	3	1
P4-T1	3	4
P5-T1	2	0
P3-T3	1	0
P4-T1	3	1
P4-T1	1	0

## 8ª Imagem

Simulador Sistema Operacional

ID da Thread:

Tempo de Execução:

Endereço:

Prioridade:

**Adicionar Processo**

**Remover Processo**

ID do Processo:

**Remover Processo**

**Escalonamento**

**FCFS**

**SJF**

RR Quantum:  **RR**

**Memória e Tempo**

**Alocação de Memória**

**Tempo de Execução Total**

**Saída do Simulador**

Processo-Thread	Executou	Restante
P1-T1	3	2
P1-T2	3	0
P3-T1	2	0
P3-T2	3	3
P3-T3	3	4
P4-T1	3	7
P5-T1	3	2
P1-T1	2	0
P3-T2	3	0
P3-T3	3	1
P4-T1	3	4
P5-T1	2	0
P3-T3	1	0
P4-T1	3	1
P4-T1	1	0

--- Alocação de Memória ---

Endereço	Processo	Thread
100	P1	T1
101	P1	T2
300	P3	T1
301	P3	T2
302	P3	T3
400	P4	T1
500	P5	T1

--- Tempo de Execução Total ---

Tempo total de todas as threads: 38

## 7. Conclusão

O projeto SimuladorSO alcançou os objetivos propostos, proporcionando uma simulação clara e didática do funcionamento básico de um sistema operacional. A implementação dos algoritmos de escalonamento demonstrou de forma prática como a ordem de execução das threads pode variar dependendo da estratégia escolhida, enquanto o gerenciamento de memória mostrou a alocação de recursos de forma organizada.

Além disso, o simulador é flexível, permitindo futuras expansões, como a inclusão de novos algoritmos de escalonamento, verificações de conflitos de memória ou até simulações mais complexas de prioridades e interrupções. Este projeto serve como uma ferramenta educacional eficiente, auxiliando no aprendizado de conceitos essenciais de Sistemas Operacionais e programação orientada a objetos.

## 8. Referências bibliográficas

**CURSO EM VÍDEO.** *Curso de Sistemas Operacionais – Playlist.* YouTube, 2020. Disponível em: <https://www.youtube.com/playlist?list=PL6i520yqwpsT-Zjj6jf9PA8M2oPk80ypf>. Acesso em: 17 ago. 2025.

**CURSO EM VÍDEO.** *Sistemas Operacionais – Playlist.* YouTube, 2023. Disponível em: [https://www.youtube.com/playlist?list=PL866\\_LrQxNVhjUCo5b9iQLvgTtC9RoyO1](https://www.youtube.com/playlist?list=PL866_LrQxNVhjUCo5b9iQLvgTtC9RoyO1). Acesso em: 15 ago. 2025.

**DRIVE STORAGE DASHBOARD UI.** *Aplicação de WPF.* YouTube, 2020. Disponível em: [https://www.youtube.com/watch?v=LbO\\_EeYi7zk](https://www.youtube.com/watch?v=LbO_EeYi7zk). Acesso em: 6 set. 2025.

**MAZIERI, José Carlos.** *Sistemas Operacionais: Conceitos e Modelos.* Curitiba: UFPR, 2021. Disponível em: <https://wiki.inf.ufpr.br/maziero/lib/exe/fetch.php?media=socm:socm-livro.pdf>. Acesso em: 25 ago. 2025.

**SISTEMA OPERACIONAL SIMULADO.** *SOSIM – Simulador de Sistema Operacional.* 2025. Disponível em: <https://www.training.com.br/sosim/>. Acesso em: 31 a