**UNIVERSIDADE DO OESTE PAULISTA** 

**Faculdade de Informática de Presidente Prudente - FIPP**

Daniel Martins de Almeida Neves

Gabriel Lucas Moreira Mariano

Mariana Fernandes Rezende

**Relatório Técnico Trabalho Prático sobre Ciclo de Vida dos Processos**

Presidente Prudente, São Paulo

2025

**1. Introdução**

O gerenciamento de processos é uma das funções mais críticas de um sistema operacional (SO). Em sistemas multitarefa, o SO deve administrar a execução concorrente de múltiplos processos, alocando recursos de forma justa e eficiente, notadamente o tempo da Unidade Central de Processamento (CPU). O componente responsável por decidir qual processo pronto deve ser alocado na CPU é o escalonador (scheduler).

O presente projeto consiste na implementação de um simulador em linguagem C para demonstrar e visualizar a dinâmica do ciclo de vida dos processos. O simulador implementa um escalonador preemptivo capaz de gerenciar múltiplas filas, simular disputas por recursos de E/S e orquestrar a criação e sincronização entre processos pai e filho.

**2. Modelagem e Estruturas de Dados**

A fundação do simulador reside na correta modelagem dos componentes lógicos de um sistema operacional. Para tal, foram definidas as seguintes estruturas de dados:

**struct processo (Process Control Block - PCB):**

Esta estrutura é a representação abstrata do Bloco de Controle de Processo. Ela encapsula todos os metadados necessários para que o sistema operacional gerencie um processo, incluindo:

* Identificadores: pid (Process ID), ppid (Parent Process ID), uid (User ID), gid (Group ID).
* Informações de Escalonamento: CPU\_Burst (tempo total de CPU requerido), tempo\_exec (tempo de CPU já consumido) e prioridade (valor numérico para o algoritmo de escalonamento).
* Gerenciamento de Hierarquia: FlagFork (sinalizador que identifica um processo como filho) e filhos (contador de descendentes ativos para a sincronização wait).
* Estado de Recursos: Recursos (flags para controle de E/S) e timeblock (contador de tempo em estado de bloqueio).

**struct tpfilac e struct desc (Implementação da Fila):**

A gestão das listas de processos nos estados de Pronto e Bloqueado foi implementada através de uma **fila encadeada com descritor**.

* O struct tpfilac representa um nó da lista, contendo o PCB do processo e um ponteiro para o próximo nó.
* O struct desc atua como o cabeçalho da fila, armazenando ponteiros para o início (inicio) e o fim (fim) da lista, além de um contador (qtde) de elementos. Esta abordagem permite operações eficientes de inserção e remoção, sendo fundamental para a dinâmica do escalonador.

### **3. Arquitetura e Descrição da Implementação**

O fluxo de controle do simulador é orquestrado por um loop principal que simula o passar do tempo em unidades discretas (ticks de clock). A cada tick, o estado dos processos e das filas é reavaliado.

#### **3.1. Fila de Prontos e Escalonamento por Prioridade**

A política de seleção de processos para execução é a de escalonamento por prioridade preemptivo. A Fila de Prontos (descProntos) não é uma fila FIFO (First-In, First-Out) convencional. A função enqueue foi implementada para realizar uma inserção ordenada (fila com prioridade) ao adicionar um novo processo, a fila é percorrida para posicioná-lo de acordo com sua prioridade. Processos com valores de prioridade mais altos (números menores no código) são inseridos mais próximos do início da fila. Consequentemente, a função dequeue sempre removerá o processo de maior prioridade disponível para execução.

#### **3.2. Alocação de CPU e Preempção com Round Robin**

Uma vez que um processo é selecionado da Fila de Prontos, ele é movido para o estado de Execução (representado pela variável CPU). A alocação de tempo de CPU é governada pelo algoritmo Round Robin. Foi definido um *quantum* (ou fatia de tempo) de 10 unidades de tempo. Um processo pode deixar a CPU sob três condições:

1. **Término da Execução:** O processo completa seu CPU\_Burst.
2. **Requisição de E/S:** O processo solicita voluntariamente um recurso de E/S, causando um bloqueio e uma troca de contexto.
3. **Expiração do Quantum:** Se o processo não terminar ou bloquear dentro de 10 unidades de tempo, ele é preemptado (forçadamente removido da CPU) e reinserido na Fila de Prontos para aguardar uma nova oportunidade, garantindo que processos longos não monopolizem a CPU.

#### **3.3. Gerenciamento de Entrada/Saída (E/S) e Estado de Bloqueio**

O simulador modela a contenção por recursos de E/S (HD, Teclado, Mouse) de forma estocástica. Durante a execução, um processo tem uma probabilidade de requisitar um desses recursos.

* **Alocação de Recurso:** Se o recurso solicitado estiver disponível, o processo transita do estado de Execução (running) para o de Bloqueado (waiting), sendo associado ao recurso por um período de tempo aleatório. A CPU é então liberada para o próximo processo da Fila de Prontos.
* **Fila de Espera de Dispositivo:** Se o recurso estiver ocupado, o processo solicitante é movido da CPU para uma fila de espera específica para aquele dispositivo (descHD, descTeclado, descMouse). Essa arquitetura reflete o comportamento real dos SOs, onde cada dispositivo possui sua própria fila de requisições.
* **Desbloqueio:** Ao final de uma operação de E/S, o processo que estava utilizando o recurso é movido de volta para a Fila de Prontos. Se a fila de espera do dispositivo não estiver vazia, o próximo processo daquela fila é então alocado ao recurso.

#### **3.4. Hierarquia de Processos: Simulação das Primitivas fork e wait**

O sistema implementa a criação e sincronização de processos através de mecanismos análogos às chamadas de sistema fork() e wait().

**Fork:**

Um processo em execução pode, probabilisticamente, invocar a função Fork. Esta cria um novo PCB, que é uma cópia do processo pai, atribuindo-lhe um novo pid e configurando seu ppid. O processo filho é então inserido na Fila de Prontos para competir pela CPU. O processo pai incrementa seu contador interno de filhos (filhos).

**Wait:**

A sincronização é imposta no término do processo pai. Se um processo pai finaliza sua execução (tempo\_exec >= CPU\_Burst) mas ainda possui filhos ativos (contador filhos > 0), ele não é removido do sistema. Em vez disso, ele transita para um estado de espera específico, sendo movido para uma fila dedicada (descWait). Ele permanece nesta fila até que todos os seus descendentes diretos terminem suas execuções e o notifiquem (decrementando o contador). Apenas quando filhos atinge zero, o processo pai é finalmente liberado da fila descWait e considerado terminado. Este mecanismo previne a ocorrência de "processos órfãos" e garante a ordem correta de finalização.

### **4. Análise de Resultados**

//acrescentar

### **5. Conclusão**

### Este projeto culminou na implementação bem-sucedida de um simulador para o ciclo de vida de processos, abordando conceitos complexos como escalonamento híbrido, gerenciamento de E/S e hierarquia de processos. A arquitetura modular e as estruturas de dados escolhidas provaram ser eficazes para modelar a dinâmica de um sistema operacional de tempo compartilhado. A visualização em tempo real das filas e do estado da CPU foi uma ferramenta valiosa para a depuração e para a compreensão do impacto de cada algoritmo e evento aleatório no comportamento geral do sistema.

Como possíveis trabalhos futuros, o simulador pode ser estendido para incluir: Gerenciamento de Memória, que consiste na implementação de algoritmos de alocação de memória e simulação de *swapping*; mecanismos de sincronização adicionais, que é a introdução de semáforos ou monitores para simular seções críticas e problemas clássicos de concorrência; e algoritmos de escalonamento alternativos, adicionam a possibilidade de selecionar outros algoritmos (e.g., Shortest Job First - SJF, Múltiplas Filas com Retroalimentação) para comparar o desempenho.