Projeto de Fundamentos de Sistemas Operacionais: Pseudo-SO Multiprogramado

Luana Cruz Silva (202033543)
Lucas de Oliveira Silva (200022857)
Regina Emy da Nóbrega Kamada (190037351)

[1]Departamento de Ciência da Computação – Universidade de Brasília (UnB)
CiC - Organização e Arquitetura de Computadores

Resumo. *Este trabalho apresenta a implementação de um pseudo-sistema operacional multiprogramado em Python, desenvolvido para simular os principais mecanismos de gerenciamento de um SO moderno, incluindo escalonamento de processos, gerenciamento de memória, alocação de recursos de E/S e operações em sistema de arquivos. O sistema foi projetado seguindo especificações rigorosas que incluem múltiplas filas de prioridade com realimentação, alocação contígua de memória e disco, e controle de acesso a recursos compartilhados.*

1. Introdução

Os sistemas operacionais modernos precisam gerenciar eficientemente recursos computacionais limitados entre múltiplos processos concorrentes. Este projeto implementa um simulador didático que reproduz quatro subsistemas essenciais de um SO: gerenciamento de processos, memória, dispositivos de E/S e sistema de arquivos. A implementação em Python permite demonstrar claramente os conceitos teóricos através de um código modular e bem estruturado, seguindo especificações do projeto.

2. Descrição das Ferramentas e Linguagens

O sistema foi desenvolvido utilizando:

- **Python**: Escolhido por sua sintaxe clara, tipagem dinâmica e rica biblioteca padrão, ideal para prototipagem rápida e código legível.
- **VSCode**: Ambiente de desenvolvimento com excelente suporte para Python e ferramentas de depuração.
- **Git/GitHub**: Para controle de versão e colaboração em equipe.
- **Sistema Operacional**: Desenvolvido em Windows e Ubuntu, mas compatível com qualquer plataforma que execute Python 3.

3. Solução Proposta: Arquitetura e Implementação

3.1. Arquitetura Geral

A solução foi construída a partir de uma arquitetura modular, em que cada componente representa um subsistema fundamental de um sistema operacional. Os módulos principais são:

- **Dispatcher**: Núcleo do sistema, responsável pela coordenação da execução dos processos e controle global da simulação.

- **Gerenciador de Processos**: Mantém os dados de cada processo, como tempo de execução, prioridade e recursos utilizados.
- **Gerenciador de Filas**: Implementa a política de escalonamento por múltiplas filas com realimentação e controle de aging.
- **Gerenciador de Memória**: Realiza a alocação contígua de blocos de memória, com separação entre memória de tempo real e de usuários.
- **Gerenciador de Recursos**: Gerencia a alocação exclusiva de dispositivos de entrada e saída.
- **Gerenciador de Arquivos**: Simula um sistema de arquivos com alocação contígua em disco e controle de permissões.

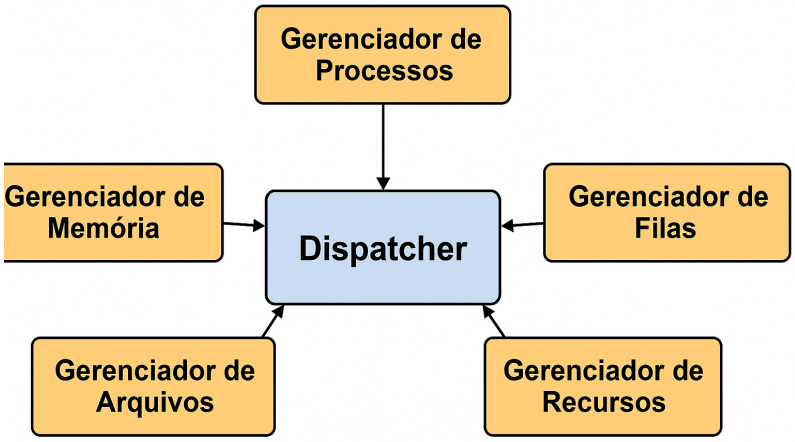

Figura 1. Diagrama de arquitetura do sistema

3.2. Descrição Teórica da Solução

3.2.1. Escalonamento de Processos

A política de escalonamento adotada busca equilibrar a eficiência e a justiça entre processos:

- **FIFO não-preemptivo** para processos de tempo real (prioridade 0), garantindo execução imediata e exclusiva.
- **Múltiplas Filas com Realimentação** para processos de usuário (prioridades 1, 2 e 3), com rebaixamento automático em caso de não conclusão.
- **Aging**: Processos que permanecem por 5 unidades de tempo em filas de menor prioridade são promovidos, evitando starvation.

3.2.2. Gerenciamento de Memória

O gerenciamento de memória é baseado em partição fixa e alocação contígua:

- A memória total é dividida em 64 blocos para processos de tempo real e 960 para processos de usuário.
- Utiliza o algoritmo **first-fit** para localizar o primeiro espaço contíguo disponível.
- Os limites de acesso são verificados para garantir proteção de memória.

3.2.3. Gerenciamento de E/S

Os dispositivos são compartilhados entre processos por meio de alocação exclusiva e não-preemptiva:

- Recursos disponíveis: 1 scanner, 2 impressoras, 1 modem e 2 discos SATA.
- A alocação ocorre apenas se todos os recursos solicitados estiverem disponíveis.
- Processos de tempo real são restritos a não utilizarem E/S.

3.2.4. Sistema de Arquivos

O sistema de arquivos simula blocos de disco com controle de permissões:

- Alocação contígua baseada em **first-fit**.
- Processos de tempo real têm permissão para deletar qualquer arquivo.
- Exibição de mapa textual do disco ao final da simulação.

3.3. Descrição Prática da Implementação

3.3.1. Módulo de Processos

A classe `Processo` encapsula as principais características de um processo, incluindo:

- Identificador (PID), prioridade, tempo de CPU e momento de inicialização.
- Requisitos de memória e dispositivos de E/S.
- Estado de execução e recursos alocados.

3.3.2. Gerenciador de Filas

Implementado com `collections.deque` para representar filas de prioridades distintas. O dispatcher executa o seguinte ciclo:

1. Verifica a fila de processos de tempo real (prioridade 0).
2. Executa o processo de maior prioridade dentre os usuários.
3. Se não finalizado, rebaixa sua prioridade.
4. Aplica o mecanismo de aging após cada ciclo.

3.3.3. Gerenciador de Memória

```
class GerenciadorMemoria:
    def __init__(self):
        self.memoria_tempo_real = [False] * 64
        self.memoria_usuario = [False] * 960
        self.processo_por_bloco = {}
```

Esse módulo realiza a alocação contígua e rastreia os blocos ocupados por processo.

3.3.4. Gerenciador de Recursos

```
recursos = {
    'scanner': 1,
    'impressora': [1, 1],
    'modem': 1,
    'disco': [1, 1]
}
```

Os recursos são verificados e alocados de forma atômica, garantindo que o processo só execute se todos os dispositivos solicitados estiverem disponíveis.

3.3.5. Gerenciador de Arquivos

- Suporte a operações atômicas de criação e exclusão.
- Verificação de permissões conforme a prioridade do processo.
- Mapa do disco impresso ao final da simulação.

4. Saída da Simulação

Abaixo está um trecho da saída da simulação com dois processos de tempo real, utilizando os arquivos de exemplo (`processes.txt, files.txt`) das especificações de implementação do projeto:

```
=== Iniciando Simulacao do Pseudo-SO ===

Tempo 2:
dispatcher =>
    PID: 0
    offset: 0
    blocks: 64
    priority: 0
    time: 3
    printers: 0
    scanners: 0
    modems: 0
    drives: 0
process 0 =>
P0 STARTED
P0 instruction 1

Tempo 3:
P0 instruction 2

Tempo 4:
P0 instruction 3
P0 return SIGINT
```

```
Tempo 8:
dispatcher =>
    PID: 1
    offset: 0
    blocks: 64
    priority: 0
    time: 2
    printers: 0
    scanners: 0
    modems: 0
    drives: 0
process 1 =>
P1 STARTED
P1 instruction 1

Tempo 9:
P1 instruction 2
P1 return SIGINT

Sistema de arquivos =>
Operacao 1 => Falha: O processo 0 nao pode criar o arquivo A (falta de
Operacao 2 => Sucesso: O processo 0 deletou o arquivo X
Operacao 3 => Falha: O processo 2 nao existe
Operacao 4 => Sucesso: O processo 0 criou o arquivo D (blocos 0-2)
Operacao 5 => Sucesso: O processo 1 criou o arquivo E (blocos 8-9)

Mapa de ocupacao do disco:
| D | D | D | Y | 0 | Z | Z | Z | E | E |

=== Simulacao Concluida ===
```

5. Principais Dificuldades e Soluções

- **Sincronização de Recursos**: Evitamos inconsistência implementando alocação atômica com verificação prévia.
- **Starvation**: Corrigido por meio da promoção de processos via aging.
- **Fragmentação de Memória**: Limitada devido ao uso de alocação contígua, o que é aceitável neste contexto didático.

6. Divisão de Tarefas

- **Luana**: Implementação do Gerenciador de Processos e Filas.
- **Lucas**: Gerenciador de Memória e Recursos de E/S.
- **Regina**: Gerenciador de Arquivos e integração geral do sistema.

7. Conclusão

Este projeto permitiu a aplicação prática de diversos conceitos fundamentais de sistemas operacionais. A arquitetura modular e a implementação em Python proporcionaram uma

visão clara e acessível dos desafios envolvidos na gerência de memória, escalonamento de processos, controle de E/S e manipulação de arquivos.

Referências

[1] Tanenbaum, A. *Modern Operating Systems*. Pearson, 2015.