Resumo Detalhado: Sistemas Operacionais (Capítulos 1 a 6)

Este documento resume os conceitos fundamentais dos primeiros seis capítulos do livro "Sistemas Operacionais: Conceitos e Mecanismos" de Carlos A. Maziero.

Capítulo 1: Conceitos Básicos

O primeiro capítulo introduz o papel e os objetivos de um Sistema Operacional (SO).

1.1. Objetivos de um SO
O SO atua como uma camada intermediária entre o hardware e os aplicativos, com dois objetivos principais:

- **Abstração de Recursos:** Simplificar o uso do hardware. Em vez de lidar com a complexidade de baixo nível de um disco (controladores, trilhas, setores), os aplicativos usam a abstração de "arquivo", com operações simples como abrir, ler e fechar. Isso torna os aplicativos independentes da tecnologia específica do hardware.
- **Gerência de Recursos:** Organizar e controlar o uso dos recursos de hardware (CPU, memória, impressora, etc.) entre múltiplos aplicativos e usuários de forma justa e eficiente, resolvendo conflitos e evitando que um programa monopolize o sistema.

1.2. Funcionalidades Principais

- **Gerência do Processador (ou de Tarefas):** Distribuir o tempo da CPU entre as várias tarefas em execução.
- **Gerência de Memória:** Fornecer a cada aplicação uma área de memória própria e isolada, protegendo-a de outras.
- **Gerência de Dispositivos (E/S):** Interagir com os periféricos através de *drivers*, oferecendo interfaces de acesso padronizadas.
- **Gerência de Arquivos:** Criar, organizar e controlar o acesso a arquivos e diretórios.
- **Gerência de Proteção:** Definir e garantir que usuários só acessem os recursos aos quais têm permissão.

1.3. Categorias de SOs
Os SOs podem ser classificados como:

- **Batch (de Lote):** Processam tarefas em sequência, sem interação do usuário.
- **Multiusuário:** Permitem que vários usuários utilizem o sistema simultaneamente, com controle de acesso.
- **Desktop:** Voltados para o usuário final, com forte apelo à interface gráfica e interatividade (Windows, macOS, Linux).
- **Móvel:** Otimizados para dispositivos com bateria e conectividade (Android, iOS).
- **Embarcado:** Compactos, para hardware com recursos limitados (sistemas de automação, eletrônicos).

- **Tempo Real:** Onde o tempo de resposta a eventos é crítico e previsível (controle de voo, freios ABS).

Capítulo 2: Estrutura de um SO

Este capítulo descreve os componentes de um SO e o suporte de hardware necessário.

2.1. Elementos do SO

- **Núcleo (Kernel):** O coração do SO, responsável pela gerência de recursos. Executa em modo privilegiado.
- **Drivers:** Módulos que permitem ao SO comunicar-se com dispositivos de hardware específicos.
- **Programas Utilitários:** Ferramentas que auxiliam no uso do sistema (formatadores de disco, interpretadores de comando, etc.).

2.3. Arquitetura do Computador e Suporte de Hardware

- **Níveis de Privilégio:** Para proteção, os processadores modernos têm pelo menos dois modos:
  - **Modo Núcleo (Kernel Mode):** Modo privilegiado, onde o kernel do SO executa e tem acesso total ao hardware.
  - **Modo Usuário (User Mode):** Modo restrito, onde as aplicações executam. Instruções perigosas (como acesso direto a hardware) são proibidas.
- **Interrupções e Exceções:** Mecanismos que transferem o controle da execução para o núcleo. Uma **interrupção** é um sinal de um dispositivo de hardware (ex: tecla pressionada). Uma **exceção** é um evento gerado pela própria CPU (ex: divisão por zero).
- **Chamadas de Sistema (System Calls):** A forma como uma aplicação em modo usuário solicita um serviço ao núcleo (que está em modo núcleo). Funciona como uma "interrupção de software" (trap), garantindo que o pedido seja mediado pelo SO de forma segura.

Capítulo 3: Arquiteturas de SOs

Descreve as diferentes maneiras de organizar a estrutura interna de um núcleo de SO.

- **Sistemas Monolíticos:** Todo o SO (gerência de processos, memória, drivers, etc.) executa como um único programa grande no modo núcleo.
  - **Vantagem:** Alto desempenho, pois a comunicação entre componentes é direta (chamadas de função).
  - **Desvantagem:** Menos robusto (um erro em um driver pode travar todo o sistema) e mais difícil de manter.
  - **Exemplos:** UNIX original, MS-DOS, Linux (embora seja modular).
- **Sistemas Micronúcleo (Microkernel):** Apenas as funções mais essenciais (gerência básica de tarefas, comunicação) rodam no núcleo. Outros serviços, como drivers e sistemas de arquivos, rodam como processos em modo usuário.
  - **Vantagem:** Mais robusto e modular. Uma falha em um serviço não derruba o sistema.

    - **Desvantagem:** Pior desempenho, pois a comunicação entre serviços exige trocas de contexto e passagem de mensagens, que são mais lentas.
    - **Exemplos:** Minix 3, QNX.
- **Sistemas Híbridos:** Uma mistura dos dois modelos anteriores. É a abordagem mais comum hoje. Componentes críticos para o desempenho rodam no núcleo, enquanto outros rodam como serviços.
    - **Exemplos:** Windows, macOS.

Capítulo 4: O Conceito de Tarefa

Este capítulo aprofunda a gerência do processador, focando na unidade de execução.

4.2. Programa vs. Tarefa

- **Programa:** Uma entidade estática. É um conjunto de instruções armazenado em disco (ex: notepad.exe).
- **Tarefa (Task):** Uma entidade dinâmica. É um programa em execução, com um estado que evolui (valores de variáveis, próxima instrução a executar).

4.3. Evolução da Gerência de Tarefas

- **Sistemas Monotarefa:** Executavam uma tarefa por vez. O processador ficava ocioso durante operações de E/S.
- **Sistemas Multitarefas:** Para evitar ociosidade, o SO suspende a tarefa que está esperando por E/S e coloca outra para executar.
- **Sistemas de Tempo Compartilhado (Time-Sharing):** Para garantir a interatividade e evitar que uma tarefa monopolize a CPU, foi introduzida a **preempção**. O SO define uma fatia de tempo (*quantum*) e usa um temporizador de hardware para interromper a tarefa em execução quando seu *quantum* acaba, passando a CPU para a próxima tarefa na fila.

4.4. Ciclo de Vida das Tarefas (Diagrama de 5 Estados)
Uma tarefa passa por vários estados durante sua vida:

1. **Nova (New):** A tarefa está sendo criada.
2. **Pronta (Ready):** A tarefa está na memória, pronta para executar, apenas esperando a CPU.
3. **Executando (Running):** A tarefa está utilizando a CPU.
4. **Suspensa (Suspended/Blocked):** A tarefa está esperando por um evento (ex: término de uma leitura de disco).
5. **Terminada (Terminated):** A execução da tarefa foi concluída.

Capítulo 5: Implementação de Tarefas

Explica como o conceito de tarefa é implementado na prática.

5.1. Contexto e TCB
O contexto de uma tarefa é o seu estado completo em um dado momento (valores dos registradores da CPU, ponteiro de instrução, etc.). O SO armazena o contexto de cada tarefa

em uma estrutura de dados chamada Bloco de Controle de Tarefa (TCB) ou Bloco de Controle de Processo (PCB).
5.2. Troca de Contexto
É o mecanismo de salvar o contexto da tarefa atual no seu TCB e carregar o contexto da próxima tarefa a partir do TCB dela. É a operação que permite a multitarefa preemptiva.
5.3. Processos
Um processo é a implementação clássica de uma tarefa. É um "contêiner" de recursos: possui um espaço de memória isolado e protegido, descritores de arquivos abertos, etc. A criação de processos é custosa (exige alocação de memória e estruturas no SO).
5.4. Threads
Uma thread é um fluxo de execução dentro de um processo.

- Múltiplas threads dentro do mesmo processo **compartilham** o mesmo espaço de memória e os mesmos recursos (arquivos abertos, etc.).
- Cada thread tem seu próprio **contexto de execução mínimo**: seus próprios registradores e sua própria pilha de execução.
- **Vantagens:** A criação e a troca de contexto entre threads do mesmo processo são muito mais rápidas que entre processos. A comunicação é fácil, pois elas compartilham memória.
- **Desvantagens:** Menos robustez. Um erro em uma thread pode derrubar todas as outras no mesmo processo.

Modelos de Threads:

- **N:1 (Nível de Usuário):** Múltiplas threads de usuário são gerenciadas por uma biblioteca e mapeadas para uma única thread de núcleo. Rápido, mas uma chamada de sistema bloqueante para todo o processo.
- **1:1 (Nível de Núcleo):** Cada thread de usuário tem uma thread correspondente no núcleo. É o modelo usado pelos SOs modernos (Windows, Linux), pois permite paralelismo real em sistemas com múltiplos processadores.

Capítulo 6: Escalonamento de Tarefas

O **escalonador** é o componente do SO que decide qual tarefa da fila de prontas irá executar.

6.2. Objetivos e Métricas
O objetivo é otimizar métricas como:

- **Tempo de Resposta:** Tempo entre um evento e a resposta do sistema (importante para sistemas interativos).
- **Tempo de Execução (Turnaround Time):** Tempo total desde a criação até o término da tarefa.
- **Tempo de Espera:** Tempo total que a tarefa passa na fila de prontas.

6.4. Algoritmos de Escalonamento Clássicos

- **First-Come, First-Served (FCFS):** O primeiro que chega é o primeiro a ser atendido. Simples, mas não preemptivo. Uma tarefa longa pode fazer tarefas curtas esperarem

muito.

- **Shortest Job First (SJF):** Seleciona a tarefa com o menor tempo de execução estimado. Ótimo para minimizar o tempo de espera médio, mas pode causar **inanição** (starvation) de tarefas longas e requer prever o futuro.
- **Round-Robin (RR):** Versão preemptiva do FCFS. Cada tarefa recebe um *quantum* da CPU. Se não terminar, volta para o final da fila. Bom para interatividade.
- **Escalonamento por Prioridades:** Cada tarefa tem uma prioridade, e a CPU é alocada para a tarefa de maior prioridade. Pode ser preemptivo. Também pode causar inanição de tarefas de baixa prioridade.
    - **Solução para inanição: Aging (envelhecimento)**. A prioridade de uma tarefa aumenta gradualmente enquanto ela espera na fila.

6.5. Escalonadores Reais

SOs modernos usam algoritmos complexos, geralmente com múltiplas filas para diferentes tipos de tarefas (ex: uma para interativas, outra para batch). O Linux, por exemplo, usa o Completely Fair Scheduler (CFS), que tenta dar a cada tarefa uma porção justa do processador.