📚 Resumo para Estudo: Chamadas de Sistema, Processos e Threads

Material de estudo baseado em sistemas operacionais

📥 Gerar PDF 🖨️ Imprimir

🔍 PARTE 1: CHAMADAS DE SISTEMA (SYSTEM CALLS)

✅ O que são?

✓ Interface entre programas e o núcleo do sistema operacional.

✓ Permitem operações como leitura/escrita de dados, gerenciamento de processos, etc.

✅ Importância:

✓ Ponte entre software e hardware.

✓ Fundamentais para desempenho, segurança e estabilidade.

✓ Essenciais para programação de sistemas e desenvolvimento de software eficiente.

✅ Impacto na Segurança:

✓ Mecanismos de segurança como permissões e validação de entradas.

✓ Vulnerabilidades podem comprometer o sistema (ex.: escape de contêineres).

✓ Uso de ferramentas como seccomp para restringir chamadas perigosas.

✅ Melhores Práticas:

✓ Minimizar chamadas desnecessárias.

✓ Documentar e testar as chamadas utilizadas.

✓ Usar namespaces e cgroups corretamente em contêineres.

✓ Exemplo: Docker, Kubernetes usam clone(), setns(), chroot().

✅ Desempenho:

✓ Chamadas de sistema impactam a latência e a responsividade.

✓ Otimização pode melhorar significativamente o desempenho.

✓ Uso de buffers e monitoramento contínuo para identificar gargalos.

🔍 PARTE 2: PROCESSOS E THREADS

✅ Processos:

✓ Programas em execução, com seu próprio espaço de memória e recursos.

✓ Programas em execução, com seu próprio espaço de memória e recursos.

✓ Isolados entre si, o que garante segurança.

✓ Comunicam-se via pipes, filas, memória compartilhada.

✓ Maior overhead devido ao contexto separado.

✅ Threads:

✓ Unidades de execução leves dentro de um processo.

✓ Compartilham memória e recursos do processo.

✓ Menor overhead, mas exigem sincronização (mutex, semáforos).

✓ Propensas a condições de corrida e deadlocks.

✅ Diferenças Principais:

Aspecto	Processos	Threads
Isolamento	Alto	Baixo
Overhead	Alto	Baixo
Comunicação	Complexa (IPC)	Direta (memória compartilhada)
Segurança	Alta	Média-Baixa
Escalabilidade	Limitada	Alta

✅ Interação e Uso Prático:

✓ Um processo pode ter múltiplas threads.

✓ Threads melhoram responsividade (ex.: GUIs, servidores web).

✓ Processos são usados em sistemas críticos (ex.: servidores, containers).

✓ Padrões de design: Pool de Threads, Produtor-Consumidor, Fork-Join.

✅ Melhores Práticas com Threads:

✓ Sincronização para evitar condições de corrida.

✓ Uso de pools de threads para reduzir overhead.

✓ Testes de estresse e detecção de deadlocks.

- Testes de estresse e detecção de deadlocks.

🧠 PONTOS-CHAVE PARA LEMBRAR:

- ✓ Chamadas de sistema são a base da interação entre app e SO.
- ✓ Processos são isolados e seguros, mas pesados.
- ✓ Threads são leves e rápidas, mas exigem cuidado com concorrência.
- ✓ Ambos são complementares e usados conforme a necessidade da aplicação.
- ✓ Segurança e desempenho dependem do uso correto de syscalls e da gerência de processos/threads.

🛠️ EXEMPLOS PRÁTICOS MENCIONADOS:

- ✓ Docker / Kubernetes: usam syscalls para isolamento.
- ✓ Servidores Web: multitthreading para lidar com requisições.
- ✓ Interfaces Gráficas: threads para responsividade.
- ✓ Sistemas de Tempo Real: agendamento preciso com threads.
