OS & Networking Guide for System/Backend Roles

A Comprehensive Overview of Key Concepts

Created on July 6, 2025

For System and Backend Engineers

Contents

1 Introduction

This guide provides an in-depth exploration of key operating system and networking concepts critical for system and backend engineering roles. It covers processes and threads, context switching and scheduling, sockets, TCP/IP, HTTP/HTTPS, and RESTful APIs and WebSockets. Each topic is explained with clarity to aid understanding for professionals and learners alike.

2 Process vs Thread

2.1 Definitions

A **process** is an instance of a program in execution, encompassing the program code, data, and system resources (e.g., memory, file handles). Each process operates in its own isolated memory space.

A **thread** is a lightweight unit of execution within a process, sharing the process's memory and resources but maintaining its own stack and program counter.

2.2 Key Differences

- **Memory**: Processes have separate memory spaces; threads within the same process share memory.
- **Resource Usage**: Processes are resource-heavy, requiring significant overhead for creation and management. Threads are lightweight, with lower overhead.
- **Communication**: Inter-process communication (IPC) is slower (e.g., pipes, message queues). Threads communicate faster via shared memory.
- **Isolation**: Process isolation ensures a crash in one process does not affect others. A thread crash may impact the entire process.

2.3 Use Cases

- **Processes**: Ideal for applications requiring isolation, such as web browsers with separate tabs or server processes.
- **Threads**: Suitable for tasks within a single application, like parallel data processing or handling multiple client requests in a server.

3 Context Switching and Scheduling

3.1 Context Switching

Context switching is the process by which a CPU switches from executing one process or thread to another. It involves:

- Saving the state (registers, program counter) of the current process/thread.
- Loading the state of the next process/thread.
- Updating system data structures (e.g., process control blocks).

Context switching incurs overhead, impacting system performance, especially in high-frequency switching scenarios.

3.2 Scheduling

Scheduling determines the order and duration for which processes or threads are executed. Common algorithms include:

- **Round-Robin**: Each process gets a fixed time slice (quantum).
- **Priority-Based**: Higher-priority tasks are executed first.
- **Shortest Job First**: Minimizes average waiting time by prioritizing short tasks.

Modern operating systems use preemptive multitasking, allowing the scheduler to interrupt and switch tasks to ensure responsiveness.

4 Sockets, TCP/IP, HTTP/HTTPS

4.1 Sockets

Sockets are endpoints for communication between devices over a network. They enable data exchange in client-server architectures.

- **Types**: Stream sockets (TCP, reliable) and datagram sockets (UDP, connectionless).
- **Use**: Sockets are used in applications like web servers, chat applications, and file transfers.

4.2 TCP/IP

The **TCP/IP** model is a suite of protocols for network communication:

- **IP**: Handles addressing and routing of packets.
- **TCP**: Ensures reliable, ordered data delivery with error checking.
- **Layers**: Application, Transport, Internet, and Link layers.

TCP/IP forms the backbone of internet communication, enabling robust data transfer.

4.3 HTTP/HTTPS

HTTP (Hypertext Transfer Protocol) is a stateless protocol for transmitting data over the web, typically using TCP. **HTTPS** adds security via SSL/TLS encryption.

- **Methods**: GET, POST, PUT, DELETE, etc., for client-server interactions.
- **Status Codes**: 200 (OK), 404 (Not Found), 500 (Server Error), etc.
- **HTTPS Security**: Encrypts data, ensuring confidentiality and integrity.

5 RESTful APIs and WebSockets

5.1 RESTful APIs

REST (Representational State Transfer) is an architectural style for designing networked applications. Key principles include:

- **Stateless**: Each request contains all necessary information.
- **Resources**: Identified by URLs, manipulated via HTTP methods.
- **Formats**: Typically JSON or XML for data exchange.

RESTful APIs are widely used in web services, e.g., retrieving user data from a server.

5.2 WebSockets

WebSockets enable bidirectional, persistent communication between client and server over a single TCP connection.

- **Use Cases**: Real-time applications like chat, live notifications, or gaming.
- **Advantages**: Low latency compared to HTTP polling.
- **Protocol**: Starts with an HTTP handshake, then upgrades to WebSocket.

5.3 Comparison

- **REST**: Request-response model, stateless, suitable for CRUD operations.
- **WebSockets**: Persistent connection, ideal for real-time, bidirectional data.

6 Conclusion

Understanding OS and networking concepts is essential for system and backend engineers. Processes and threads form the foundation of program execution, while context switching and scheduling optimize resource utilization. Networking protocols like TCP/IP and HTTP/HTTPS, along with sockets, enable robust communication. RESTful APIs and WebSockets provide flexible options for building scalable, real-time applications. This guide serves as a concise reference for these critical topics.