2.1. Low-Level Hardware Interaction

- **Direct Access to Resources:**
  System-level programmers write code that interacts with hardware directly (or via the operating system) to control memory, I/O devices, processors, and interrupts.
- **Memory Management:**
  Manual memory management is common in system programming, especially in languages like C and assembly. This contrasts with application programming, where garbage collection is often employed.
- **Performance and Efficiency:**
  Code is typically optimized for speed and minimal resource usage. Even minor inefficiencies can lead to significant performance degradation in system-critical software.

2.2. Use of Low-Level Languages

- **C/C++:**
  The C programming language is a staple of system-level programming due to its efficiency, close-to-hardware capabilities, and low runtime overhead. C++ is also used in areas where object-oriented design can help manage complexity without losing too much performance.
- **Assembly Language:**
  For the most time-critical sections, assembly language is sometimes used to write code that executes directly on the processor.
- **Other Languages:**
  Languages like Rust have recently gained attention in system programming for their memory safety features while retaining performance.

2.3. Operating System Interfaces

- **System Calls:**
  System-level code makes extensive use of system calls (e.g., read, write, open, fork, exec) that the operating system provides for interacting with hardware or managing resources.
- **Kernel Development:**
  Many system-level programs run in privileged mode (kernel mode) where they have unrestricted access to hardware. Developing operating systems or drivers often requires coding in a system-level language.