Operating Systems - Chapter 13 Summary

Early Systems

• Early machines had no abstraction of memory: OS loaded at low addresses, program loaded directly in physical memory.
• Multiprogramming allowed multiple processes to reside in memory, switching between them improved CPU utilization.
• Time sharing extended this idea to multiple interactive users, creating the need for protection among processes.

The Address Space Abstraction

• OS provides each process with its own address space, an abstraction of physical memory.
• Address space typically includes:
– Code segment (program instructions).
– Heap (dynamic data allocated with malloc/new).
– Stack (local variables, function calls, parameters).
• Heap grows upward, stack grows downward, meeting in the middle.
• Every address a program sees is a virtual address; the OS and hardware translate it to a physical address.

The Crux: Virtualizing Memory

• Problem: how to give each process a large, private address space on limited physical memory.
• Solution: virtual memory system with hardware + OS support translates virtual to physical addresses.

Goals of Virtual Memory

• Transparency: processes are unaware memory is virtualized; they behave as if they own the whole memory.
• Efficiency: virtualization should have low time and space overhead (e.g., hardware features like TLBs).
• Protection: processes must be isolated; one process cannot read or write another's memory.
• Isolation ensures reliability: failure of one process does not affect others or the OS.

Summary

Virtual memory provides each process with the illusion of a private, large address space. It includes code, heap, and stack segments, all accessed with virtual addresses. The OS and hardware translate these to physical memory, ensuring transparency, efficiency, and protection. This abstraction is essential for multiprogramming, isolation, and reliable system operation.