OPERATING SYSTEM SUPPORT

1/6/2026

From Chapter 7 of Distributed Systems Concepts and Design,

By G. Coulouris, J. Dollimore, T. Kindberg and G. Blair

Published by Addison Wesley/Pearson Education

Topics

- Introduction
- The Operating System Layer
- Processes and Threads
- Operating System Architecture

Introduction

- We have learned that an important aspect of distributed systems is resource sharing.
- Client applications invoke operations on resources that are often on another node or at least in another process.
- Applications (in the form of clients) and services (in the form of resource managers) use the middleware layer for their interactions.
- Middleware enables remote communication between objects or processes at the nodes of a distributed system.
- The main types of remote invocation found in middleware, such as RPC, Java RMI and ….

Introduction

- Below the middleware layer is the operating system (OS) layer.
- An operating system such as UNIX (and its variants, such as Linux and Mac OS X) or Windows (and its variants, such Windows 10 and Windows 11) provides the programmer with, for example, files rather than disk blocks.
- It takes over the physical resources on a single node and manages them to present these resource abstractions through the system-call interface.
- Two operating system concepts have come about during the development of distributed systems:

network operating systems and distributed operating systems.

Introduction

- Distributed operating systems are operating systems in which users are never concerned with where their programs run, or the location of any resources.
- There is a single system image.
- The operating system has control over all the nodes in the system, and it transparently locates new processes at whatever node suits its scheduling policies.
- Many distributed operating systems have been investigated, but there are none in general/wide use (HarmonyOS, BlueOS, MINIX).

Introduction

- But network operating system those have a networking capability built into them and so can be used to access remote resources.
- Network operating system are in wide use for various reasons both technical and non-technical.
  - Users have much invested in their application software; they will not adopt a new operating system that will not run their applications.
  - The second reason against the adoption of distributed operating system is that users tend to prefer to have a degree of autonomy for their machines, even in an organization.
- Unix and Windows are two examples of network operating systems.

Introduction

- The combination of middleware and network operating systems provides an acceptance balance between the requirement of autonomy and network transparent resource access.
- The network operating systems allows users to run their favorite word processor and other standalone applications.
- Middleware enables users to take advantage of services that become available in their distributed system.

The Operating System Layer

- Figure 1 shows how the operating system layer at each of two nodes supports a common middleware layer in providing a distributed infrastructure for applications and services.
- Kernels and server processes are the components that manage resources and present clients with an interface to the resources.

The Operating System Layer

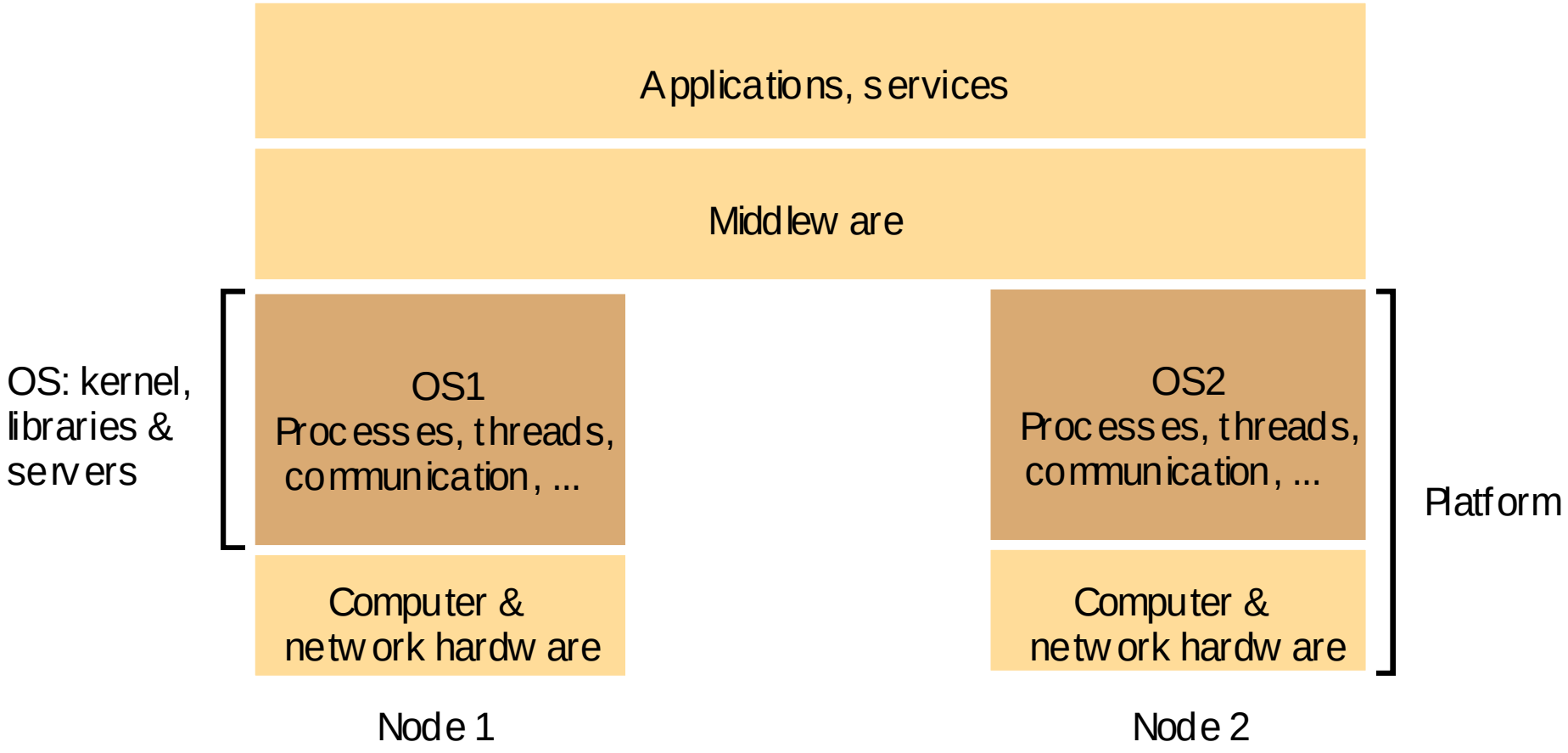

Figure 1. System layers

The Operating System Layer

- The OS facilitates:
  - **Encapsulation:** providing a useful service interface to resources
  - **Protection:** resources require protection from illegitimate accesses
  - **Concurrent processing:** clients may share resources and access them concurrently.
- Clients access resources by making RMI to server object.
- Invocation mechanism is a means of accessing an encapsulated resource.

The Operating System Layer

- Figure 2 shows the core OS components:
  - **Process manager:** handles the creation of and operations upon processes.
  - **Thread manager:** thread creation, synchronization and scheduling.
  - **Memory manager:** management of physical and virtual memory.
  - **Communication manager:** communication between threads
  - **Supervisor:** dispatching of interrupts, system call, and other exceptions.

The Operating System Layer

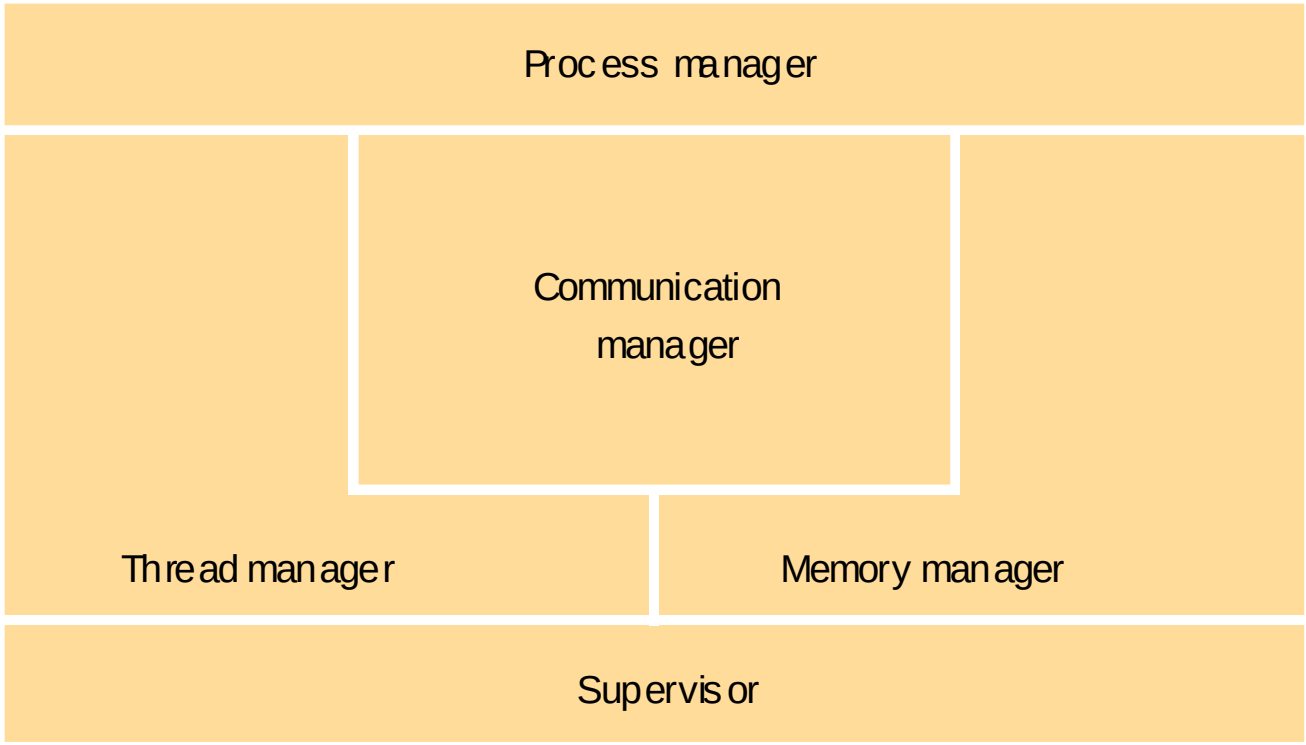

Figure 2. Core OS functionality

Processes and Threads

- Process

- The traditional operating system notion of a process that executes a single activity was found to be unequal to the requirements of distributed systems
- The problem, is that the traditional process makes sharing between related activities awkward and expensive.
- The solution reached was to enhance the notion of a process so that it could be associated with multiple activities (threads).
- A process consists of an execution environment together with one or more threads.
- A thread is the operating system abstraction of an activity.
- An execution environment is the unit of resource management: a collection of local kernel managed resources to which its threads have access.

Processes and Threads

- An execution environment consists of :
  - An address space
  - Thread synchronization and communication resources (e.g., semaphores, sockets)
  - Higher-level resources (e.g., open files and windows).
- Execution environments are normally expensive to create and manage, but several threads can share them

Processes and Threads

- Threads
  - Threads are schedulable activities attached to processes.
  - The aim of having multiple threads of execution is :
    - To maximize degree of concurrent execution between operations
    - To enable the overlap of computation with input and output
    - To enable concurrent processing on multiprocessors.

Processes and Threads

- Threads can be helpful within servers:
  - Concurrent processing of client's requests can reduce the tendency for servers to become bottleneck.
    - E.g. one thread can process a client's request while a second thread serving another request waits for a disk access to complete Figure 3 .
- Alternative server threading architectures
- Thread-per-request
- Thread-per-connection
- Thread-per-object remote

Processes and Threads

- Threads can be useful for clients as well as servers.
- Figure 3 also shows a client process with two threads:
- The first thread generates results to be passed to a server by remote method invocation,
- The second thread, which performs the remote method invocations and blocks while the first thread is able to continue computing further results.

Client and server with threads

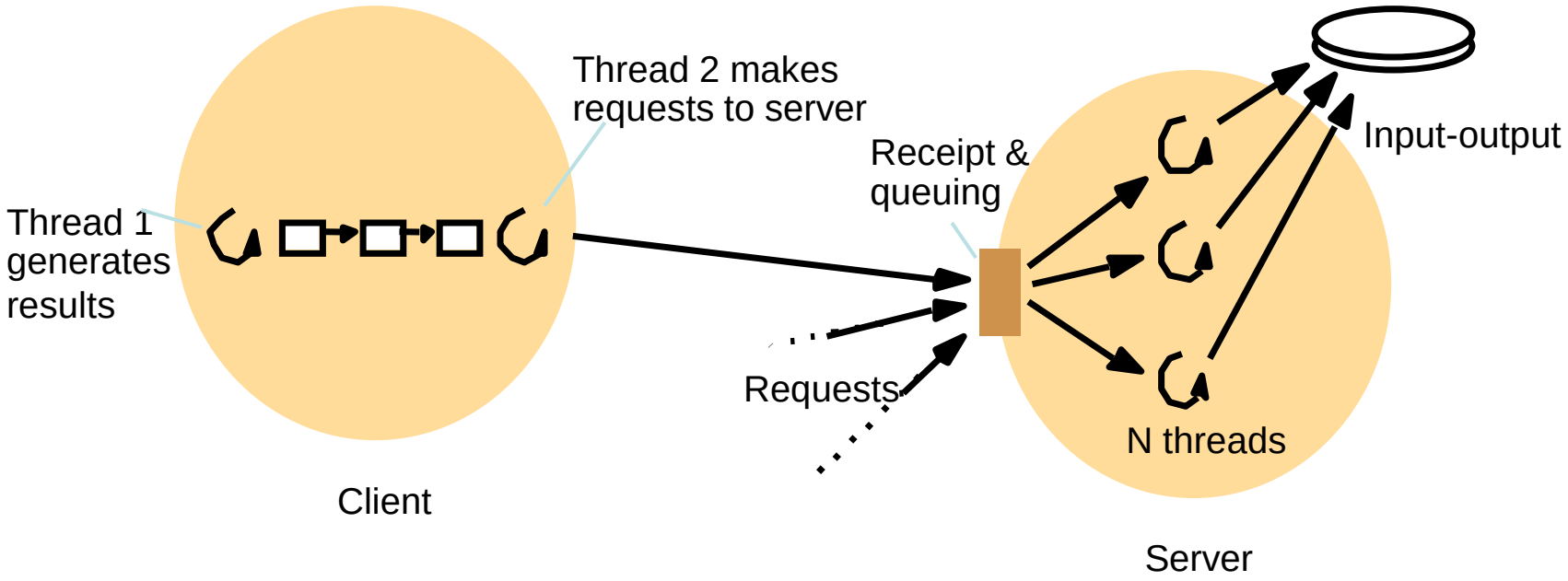

Figure 3. Client and Server with threads

Processes and Threads

- Processes vs. Threads
  - We can summarize a comparison of processes and threads as follows:
    - Creating a new thread within a process is cheaper than creating a process.
    - More importantly, switching to a different thread within the same process is cheaper than switching between threads belonging to different processes.
    - Threads within a process may share data and other resources conveniently and efficiently compared with separate processes.
    - *But, threads within a process are not protected from one another*

Operating System Architecture

- Ideally, the kernel would provide only the most basic mechanisms upon which the general resource management tasks at a node are carried out.
- Server modules would be dynamically loaded as required, to implement the required resource management policies for the currently running applications.
- The major kernel architectures:
  - Monolithic kernels
  - Micro-kernels

These designs differ primarily in the decision as to what functionality belongs in the kernel and what is to be left to server processes that can be dynamically loaded to run on top of it.

Operating System Architecture

- Monolithic Kernels
  - A monolithic kernel can contain some server processes that execute within its address space, including file servers and some networking.
  - The code that these processes execute is part of the standard kernel configuration.

(Figure 4)

Operating System Architecture

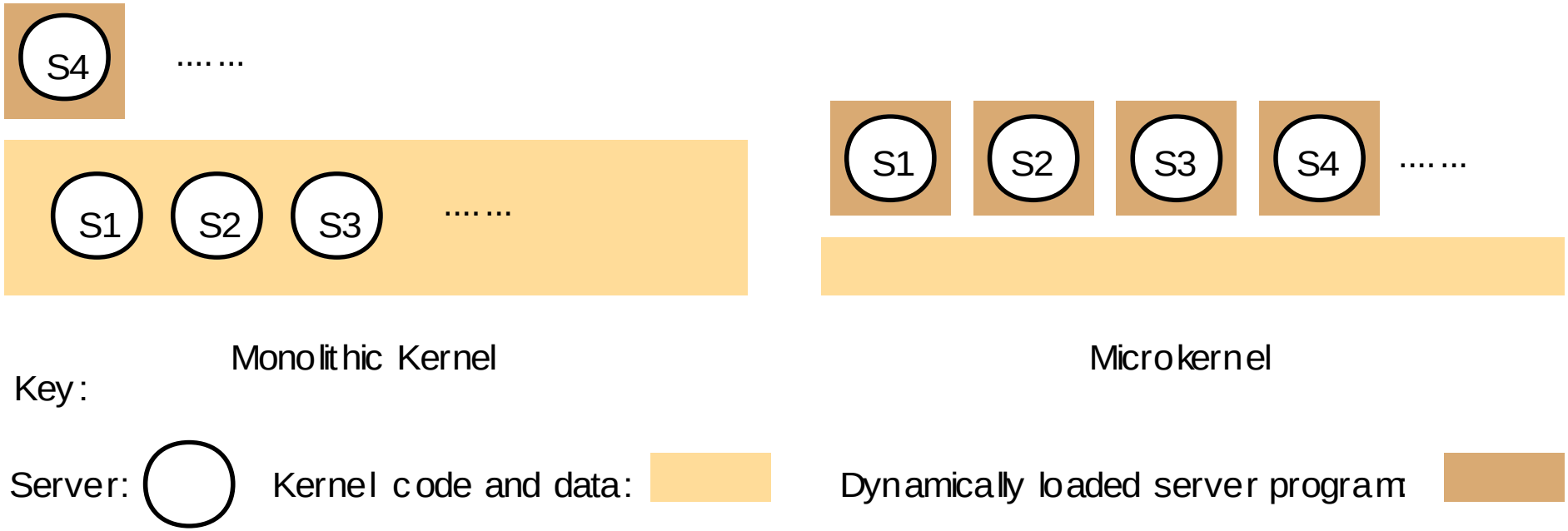

Figure 4. Monolithic kernel and microkernel

Operating System Architecture

- Microkernel
  - The microkernel appears as a layer between hardware layer and a layer consisting of major systems components.
  - the kernel provides only the most basic abstractions, principally address spaces, threads and local interprocess communication; all other system services are provided by servers that are dynamically loaded at precisely those computers in the distributed system that require them
  - If performance is the goal, rather than portability, then middleware may use the facilities of the microkernel directly.