Chapter 3

Real-Time Operating Systems

Real-Time Operating Systems

- The heart of many computerized embedded systems is real-time operating system (RTOS).
- An RTOS is an operating system that supports the construction of applications that must meet real-time constraints in addition to providing logically correct computation results.
- It provides mechanisms and services to carry out real-time task scheduling, resource management, and inter-task communication.

Main Functions of General-Purpose Operating systems

- OS is the software that sits between the hardware of a computer and software applications running on the computer.
- An OS is a resource allocator and manager.
- An OS is a policy enforcer. It defines the rules of engagement between the applications and resources and controls the execution of applications to prevent errors and improper use of the computer.
- An OS is composed of multiple software components, and the core components in the OS form its kernel.
- The kernel of an OS always
- runs in system mode, while other parts and all applications run in user mode.

Main Functions of General-Purpose Operating systems (2)

- OS also provides an application programming interface (API), which defines the rules and interfaces that enable applications to use OS features and communicate with the hardware and other software applications.
- User processes can request kernel services through system call or by message passing.
- With the system call approach, the user process applies traps to the OS routine that determines which function is to be invoked, switches the processor to system mode, calls the function as a procedure, and then switches the processor back to user mode when the function completes and returns control to the user process.

Main Functions of General-Purpose Operating systems (3)

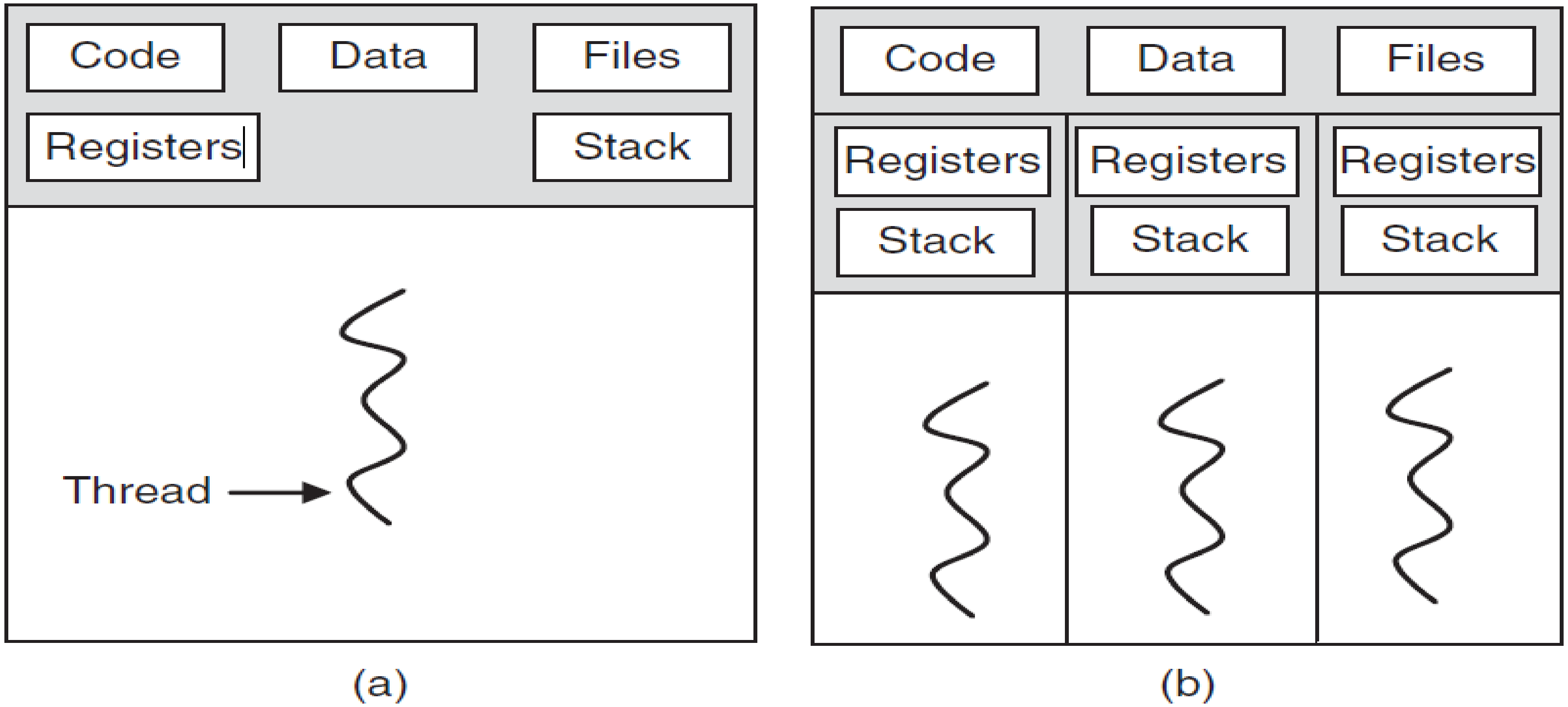
- In the message passing approach, the user process constructs a message that describes the desired service and then uses a send function to pass it to an OS process. The send function checks the desired service specified in the message, changes the processor mode to system mode, and delivers it to the process that implements the service. Meanwhile, the user process waits for the result of the service request with a message receive operation. When the service is completed, the OS process sends a message back to the user process.

Process Management

- A process is an instance of a program in execution. It is a unit of work within the system. A process needs CPU, memory, I/O devices, and files, to accomplish its task.
- A thread is a path of execution within a process and the basic unit to which the OS allocates processor time. A process can be single-threaded or multithreaded.
- Threads within the same process share the same address space, global and static variables, file descriptors, signal bookkeeping, code area, and heap. This allows threads to read from and write to the same data structures and variables and also facilitates communicate between treads. each thread of the same process has its own thread status, program counter, registers, and stack.

Process Management (2)

- Threads are used for small tasks, whereas processes are used for more heavy weight tasks. (lightweight process).
- In RTOSs, the term "tasks" is often used for threads or single-threaded processes.



(a) Single-threaded process; (b) Multithreaded process.

Other Functions

- InterruptsManagement.
- Multitasking
- File System Management.
- I/O Management.

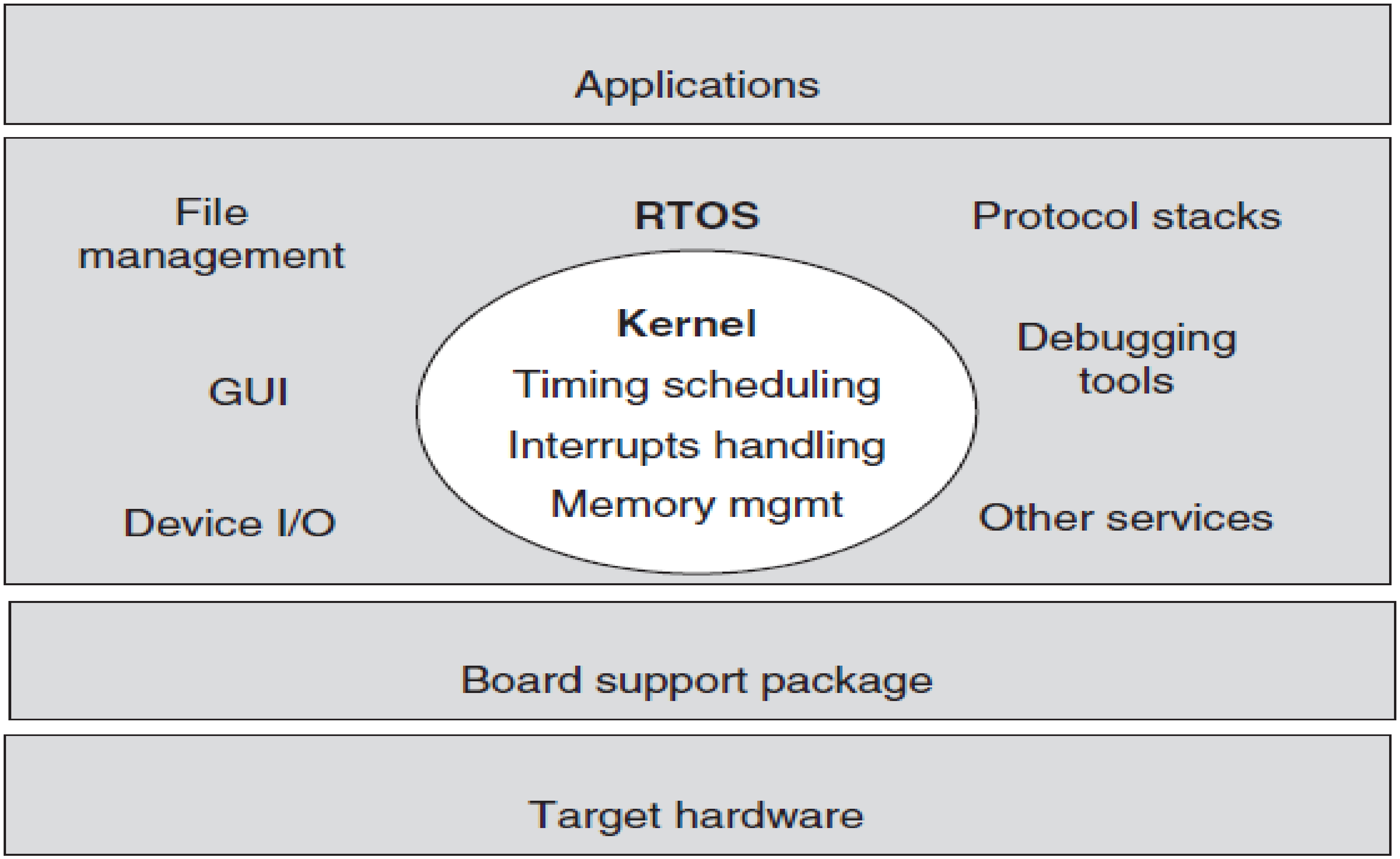
Characteristics of RTOS Kernels

- A general-purpose OS provides a rich set of services that are also needed by real-time systems, it takes too much space and contains too many functions that may not be necessary for a specific real-time application.
- A general-purpose OS is not configurable, and its inherent timing uncertainty offers no guarantee to system response time.

Characteristics of RTOS Kernels (2)

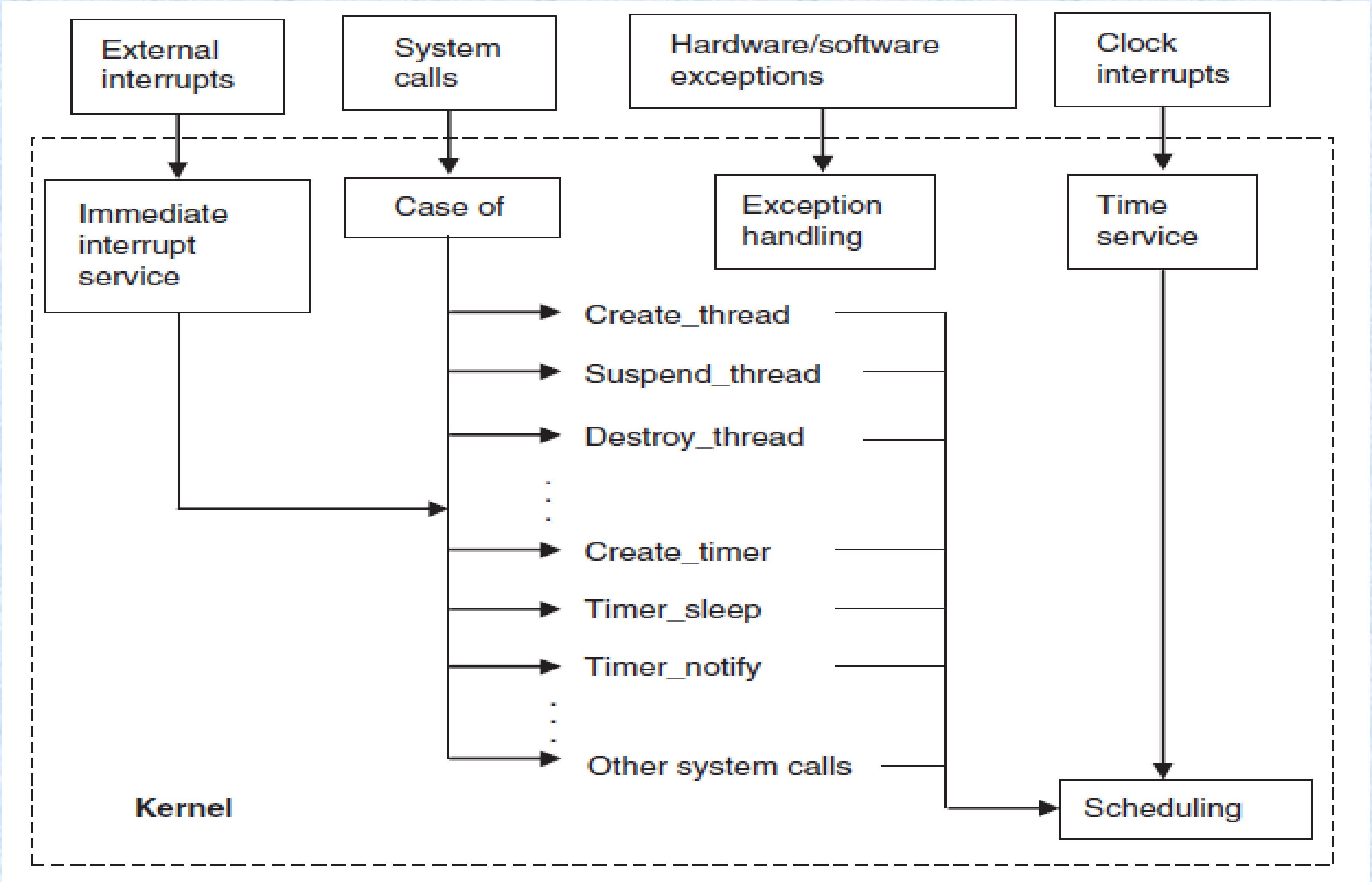
- There are three key requirements of RTOS design.
- Firstly, the timing behavior of the OS must be predictable. For all services provided by the OS, the upper bound on the execution time must be known.
- Secondly, the OS must manage the timing and scheduling, and the scheduler must be aware of task deadlines.
- Thirdly, the OS must be fast. For example, the overhead of context switch should be short.

Components of RTOS



A high-level view of RTOS

Structure of a microkernel



Structure of a microkernel

Components of RTOS (2)

- A real-time kernel and other higher-level services such as file management, protocol stacks.
- A Graphical User Interface (GUI), and other components. Most additional services revolve around I/O devices.
- A real-time kernel is software that manages the time and resources of a microprocessor or microcontroller and provides indispensable services such as task scheduling and interrupt handling to applications.
- In embedded systems, a small amount of code called board support package (BSP) is implemented for a given board that conforms to a given OS.
- It is commonly built with a boot-loader that contains the minimal device support to load the OS and device drivers for all the devices on the board.

Clocks and Timers

- Embedded systems must keep track of the passage of time.
- The length of time is represented by the number of system ticks in most RTOS kernels.
- The RTOS works by setting up a hardware timer to interrupt periodically, say, every millisecond, and bases all timings on the interrupts.
- Some RTOS kernels, define routines that allow the user to set and get the value of system tick.
- The timer is often called a heartbeat timer, and the interrupt is also called clock interrupt.
- Timers improve the determinism of real-time applications. Timers allow applications to set up events at predefined intervals or time.

Priority Scheduling

- Real-time tasks have deadlines, all tasks are not equal in terms of the urgency of getting executed.
- Tasks with shorter deadlines should be scheduled for execution sooner than those with longer deadlines. Therefore, tasks are typically prioritized in an RTOS.
- If a higher priority task is released while the processor is serving a lower priority task, the RTOS should temporarily suspend the lower priority task and immediately schedule the higher priority on the processor for execution. **(preemption.)**
- Task scheduling for real-time applications is typically priority-based and preemptive.

Priority Scheduling (2)

- In priority-driven preemptive scheduling, the scheduler has a clock interrupt task that provides each task with a the time slice.
- Dispatcher, is the module that gives control of the CPU to the task selected by the scheduler and It is responsible for performing context switches.
- The dispatcher should be as fast as possible, during the context switches, the processor is virtually idle; thus, unnecessary context switches should be avoided.
- Choosing priorities for tasks is vital. Priority-driven scheduling may cause low-priority tasks to starve and miss their deadlines.

Inter-task Communication and Resource Sharing

- In an RTOS, a task cannot call another task.
- Tasks exchange information through message passing, memory sharing.
- Tasks Coordinate their execution and access to shared data using real-time signals, mutex, or semaphore objects.

Real-Time Signals

- Signals are similar to software interrupts.
- A signal is automatically sent to the parent when a child process terminates.
- Signals are also used for many other synchronous and asynchronous notifications, such as waking a process when a wait call is performed.
- Signals are the way to inform the processes of the occurrence of asynchronous events such as high-resolution timer expiration, fast inter-process message arrival, asynchronous I/O completion, and explicit signal delivery.

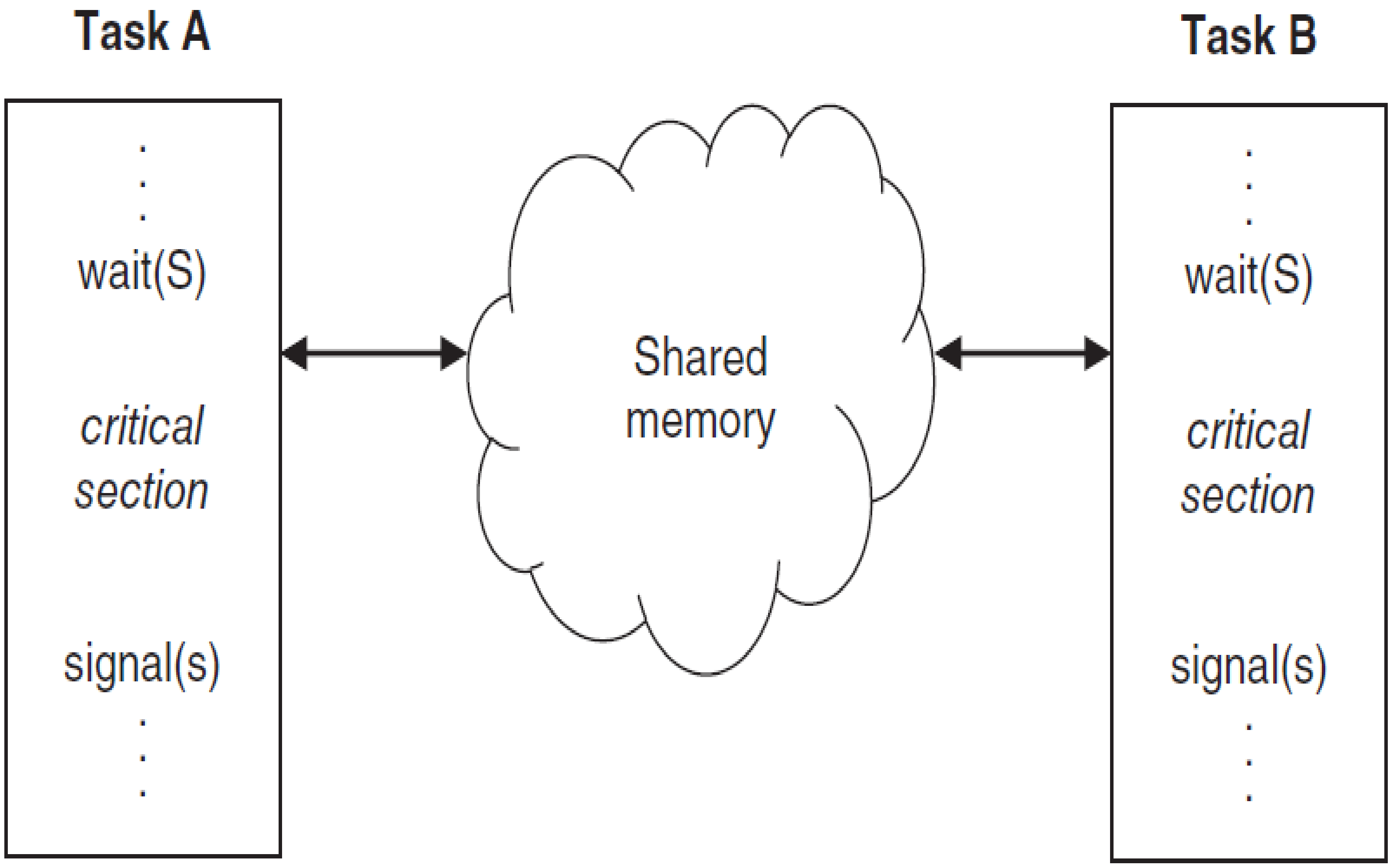
Semaphors

- Semaphores are counters used for controlling access to resources shared among processes or threads.
- The value of a semaphore is the number of units of the resource that are currently available.
- Signal: atomically increment the counter.
- Wait: atomically decrement the counter.
- A semaphore tracks only how many resources are free; it does not keep track of which of the resources are free.
- A binary semaphore acts similarly to a mutex, which is used in the case when a resource can only be used by at most one task at any time.

Message passing

- Tasks can share data by sending messages in an organized message passing scheme.
- Message passing useful for information transfer. It can also be used for synchronization.
- Message contents can be anything that is mutually comprehensible between the two parties in communication.
- Two basic operations are send and receive.
- Message passing can be direct (name the process) or indirect (use mailbox).
- Message passing can be synchronous (sender process is blocked until the message primitive has been performed) or asynchronous (sender process immediately gets control back).

Shared memory



Shared memory and critical section

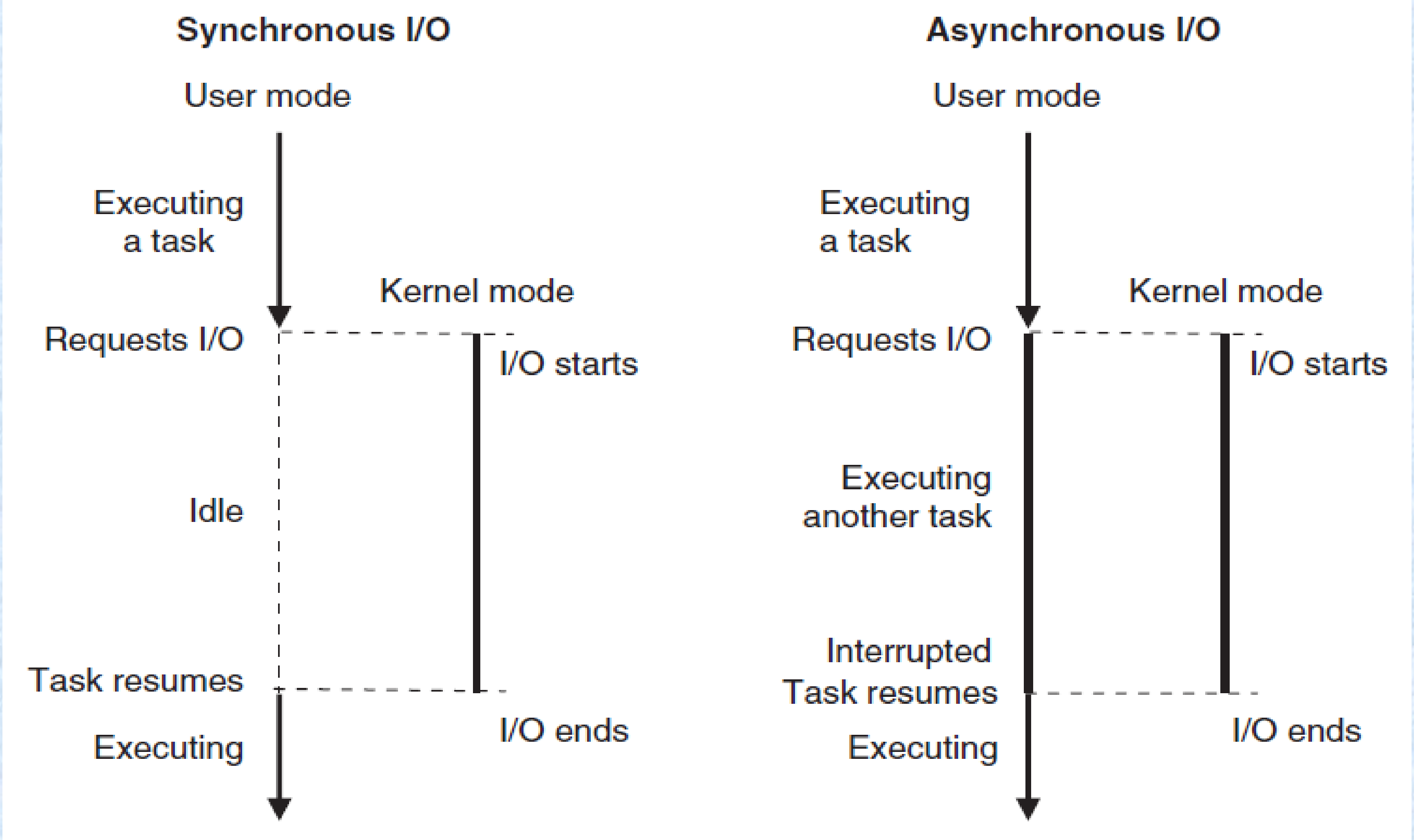
Shared memory (2)

- Shared memory is commonly used to share information (resources) between different processes or threads.
- Shared memory must be accessed exclusively using mutex or semaphores to protect the memory area.
- The code segment in a task that accesses the shared data is called a critical section.
- A side effect in using shared memory is that it may cause priority inversion, a situation that a low-priority task is running while a high-priority task is waiting.

Asynchronous I/O

- In synchronous I/O, when a user task requests the kernel for I/O operation and the requested is granted, the system will wait until the operation is completed before it can process other tasks.
- Synchronous I/O is desirable when the I/O operation is fast. It is also easy to implement.
- In asynchronous I/O, after a task requests I/O operation, while this task is waiting for I/O to complete, other tasks that do not depend on the I/O results will be scheduled for execution. Meanwhile, tasks that depend on the I/O having completed are blocked.
- Asynchronous I/O is used to improve throughput, latency, and/or responsiveness.

Asynchronous I/O (2)



Synchronous I/O versus asynchronous I/O.

RTOS Examples

- LynxOS
- OSE
- QNX
- VxWorks
- Windows Embedded Compact