# **Embedded Systems Essentials with Arm: Get Practical with Hardware**

## Module 3

## KV1: Operating System Overview

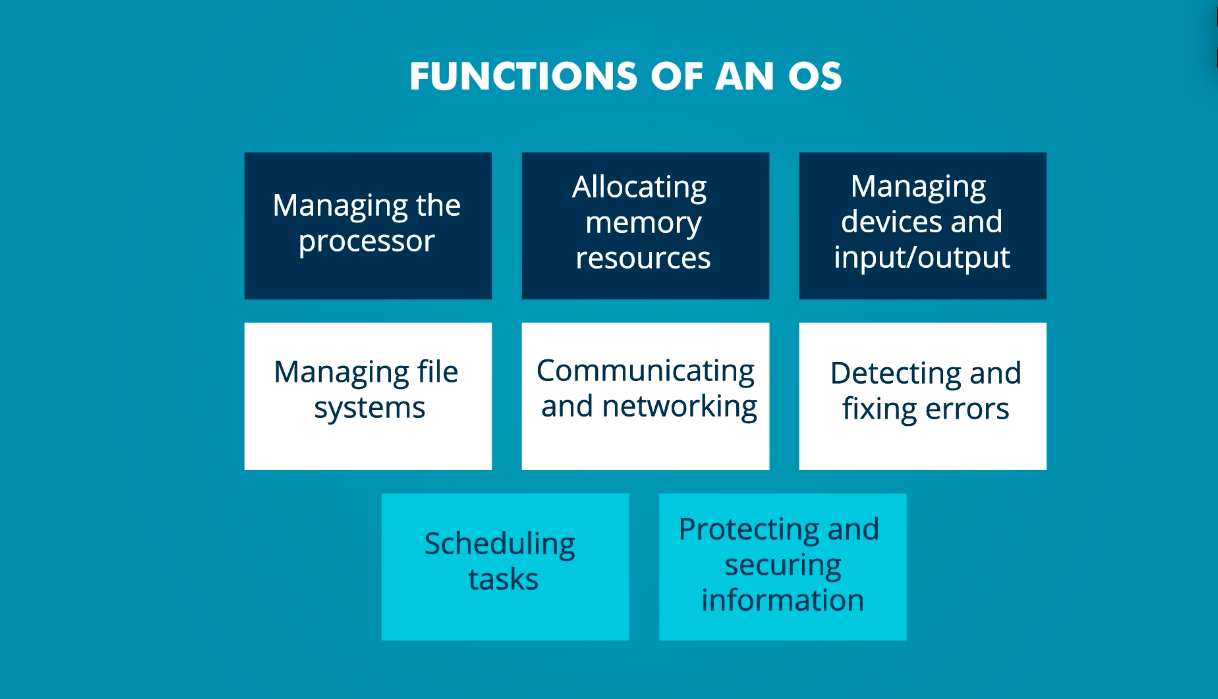
Using mobile phones, laptops, desktops, and other computerized devices has become part of everyday life. When we do this, we engage with familiar operating systems such as Windows, Linux Mac OS, and Android, to name a few. But what exactly is an operating system?

Whether we're working with a laptop or an embedded microcontroller, we’ll have a collection of hardware. It would be challenging for us as the users to manage the hardware and operation of our computer within our program. To get around this, we hand over that control to the operating system (OS). The OS looks after many of the features of our hardware, allowing us to focus on writing our application.

An OS has some common features.

* It provides the software interface between our application and the computer hardware, giving us a framework to work within.
* It helps us develop applications conveniently and efficiently.
* It allows us to write code without having to learn the details of the hardware.

But what does an OS actually do? This is a collection of activities that the OS does.



As we delve deeper into the content of this course, we’ll explore each of these and their application in the embedded systems environment.

Operating systems such as Windows, Linux, Android, or iOS allow multiple programs to run at the same time, and they manage their hardware in a way that makes them easy and enjoyable to use. However, these operating systems aren’t suitable for the embedded environment.

Let’s explore the reasons for this:

* Firstly, the embedded environment may be resource constrained. We may be working with a low-cost, low-power microcontroller with limited processing power and memory.
* Secondly, in the embedded environment, we may not even have a user interface, which is important on a laptop or mobile device. We’re more concerned with the many other hardware devices surrounding our microcontroller.
* Finally, There may be demanding time pressures.

For an OS to be appropriate for the embedded environment, it must be:

* Designed to be used in embedded computers, possibly with limited processing power.
* Compact in size, giving it a low demand for memory.
* Responsive to the I/O needs of most embedded systems.
* Able to manage multitasking within real-time constraints.
* Able to respond in predictable time to external events - we refer to this as deterministic behavior.
* Power-conscious because a lot of our embedded microcontrollers will be battery-powered.

An embedded program will have a number of activities, no matter how simple it may be. The program can divide up those activities and develop program sections for each one. Within our OS, we call these activities tasks or threads. A task is a program strand or section which has a clear and distinct purpose and outcome. ​

Multi-tasking describes a situation where there are a number of tasks which need to be performed, ideally simultaneously or concurrently. ​This is fundamental to many of our embedded applications.

​

The tasks are usually not equally important. Therefore, different tasks have different priorities. A high-priority task has the right to execute before a low-priority task. ​

​

Some tasks are time-triggered, and others are event-triggered. A time-triggered task occurs after a certain period of time, and is usually periodic. Event-triggered tasks occur when a certain event takes place. ​

In the next few videos, you will see how these terms and concepts are applied to a special operating system known as the Real-Time Operating System.

**Visit arm.com for more information.**