Porting uC-OS-III

This chapter describes how to adapt \( \mu \)C/OS-III to different processors. Adapting \( \mu \)C/OS-III to a microprocessor or a microcontroller is called porting. Most of \( \mu \)C/OS-III is written in C for portability. However, it is still necessary to write processor-specific code in C and assembly language. \( \mu \)C/OS-III manipulates processor registers, which can only be done using assembly language unless the C compiler supports inline assembly language extensions. Porting \( \mu \)C/OS-III to different processors is relatively easy as \( \mu \)C/OS-III was designed to be portable and, since \( \mu \)C/OS-III is similar to \( \mu \)C/OS-II, the user can start from a \( \mu \)C/OS-II port. In fact, this is the easiest way to do a \( \mu \)C/OS-III port.

If there is already a port for the processor to be used, it is not necessary to read this chapter unless, of course, there is an interest in knowing how \( \mu \)C/OS-III processor-specific code works.

\( \mu \)C/OS-III can run on a processor if it satisfies the following general requirements:

- The processor has an ANSI C compiler that generates reentrant code. In fact, the toolchain used must contain an assembler, C compiler and linker/locator. Finding such a toolchain is generally not an issue since there are a number of good toolchains available on the market.

- The processor supports interrupts and can provide an interrupt that occurs at regular intervals (typically between 10 and 1000 Hz). Most processors (especially MCUs) provide timer that can be used for this purpose. Some processors even have dedicated timers for use by an RTOS.

- Interrupts can be disabled and enabled. All current processors that we’ve worked with offer this. Ideally, the processor allows you to save the current state of the interrupt mask so that it can be restored.

- The processor supports a hardware stack that accommodates a fair amount of data (possibly many kilobytes).

- The processor has instructions to save and restore the stack pointer and other CPU registers, either on the stack or in memory.

- The processor has access to sufficient RAM for \( \mu \)C/OS-III’s variables and data structures as
well as internal task stacks.

- The compiler should support 32-bit data types. For some fast 32-bit processors, the compiler should also support 64-bit data types (typically “\texttt{long long}”).

The figure below shows the µC/OS-III architecture and its relationship with other software components and hardware. When using µC/OS-III in an application, the user is responsible for providing application software and the µC/OS-III configuration sections.
(1) The port developer is responsible for providing the µC/OS-III CPU Specific portion. A µC/OS-III port consists of writing or changing the contents of four kernel-specific files: `os_cpu.h`, `os_cpu_a.asm`, `os_cpu_a.inc` and `os_cpu_c.c`. 
A port also involves writing or changing the contents of two CPU specific files: cpu.h and cpu_a.asm.cpu_core.c is generally generic and should not require modifications.

A Board Support Package (BSP) is generally necessary to interface µC/OS-III to a timer (which is used for the clock tick) and an interrupt controller.

Some semiconductor manufacturers provide source and header files to access on-chip peripherals. These are contained in CPU/MCU specific files. You generally don’t need to modify any of these and thus, you can use them as-is.

Porting µC/OS-III is quite straightforward once the subtleties of the target processor and the C compiler/assembler are understood. Depending on the processor, a port consists of writing or changing between 100 and 400 lines of code, which takes a few hours to a few days to accomplish. The easiest thing to do, however, is to modify an existing port from a processor that is similar to the one intended for use.

A µC/OS-III port looks very much like a µC/OS-II port. Since µC/OS-II was ported to well over 45 different CPU architectures it is easy to start from a µC/OS-II port. Converting a µC/OS-II port to µC/OS-III takes approximately an hour. The process is described in Migrating from uC-OS-II to uC-OS-III.

A port involves three aspects: CPU, OS and board-specific code. The board-specific code is often called a Board Support Package (BSP) and from µC/OS-III’s point of view, requires very little.

In this section, we’ll present the steps needed to do a port from scratch. Actually, you’ll be starting from templates files that already contain placeholders for the code you’ll need to insert.

The following is the layout for this section:

- Conventions
- µC/CPU Port Files
- µC/OS-III Port Files
• BSP Files

• Testing a Port