REAL TIME OPERATING SYSTEMS

Lesson-14:
Memory Optimization

1. Memory Optimization

Memory Optimization

- Compact code without effect on performance
- Code means code compiled and assembled executable in the given system
- Compact code also reduces the total number of CPU cycles, and thus, the total energy requirements

2. Memory Saving Methods

Using Compressed data structures

- Use compressed data structure─ provided de-compression algorithm plus compressed data structures combined together take less memory in the system compared to the case when only un compressed data structures are used .

Compact codes

- For example, Thumb® 16-bit Instructions in 32-bit ARM processor

Appropriate Data type Declarations

- Use unsigned bytes especially within the for-loops and while loops for a short and a short for an integer, if possible, to optimise memory use system.
- Avoid if possible the use of 'long' integers and ‘double’ precision floating point values especially within the for-loops and while loops

Static (global) variables

- use static (global) variables if shared data problems are tackled and use static variables in case it needs saving frequently on the stack

Limited Library functions

- Avoid use of library functions if a simpler coding is possible

Configure the RTOS functions

- For example, if queues not needed the RTOS queue functions are not ported in the ROM image
- Use a configurable, scalable, hierarchical RTOS using which in the ROM image only the needed functions and objects for the functions are saved in place of all the kernel functions

Less Stack data Memory and Optimize stack and buffer allocations

- Reduce use of frequent function calls and nested calls
- Allocated stack and buffer are first filled with the specific bytes. Then find that in worst cases of running of the embedded system, how many filled bytes do not change.
- Reduce the allocated stack and buffer sizes for finalized code

Assembly code

- use the assembly codes for simple functions like configuring the device control register, port addresses and bit manipulations if the instruction set
- Use inline modifiers for all frequently used small sets of codes in the function or the operator overloading functions if the ROM is available in the system

Optimize Commands

- Combine two functions, if possible, when having more or less similar code
- Use, if feasible, alternatives to the *switch* statements a table of pointers to the functions
- *C++*— use the classes without multiple inheritance, without template, with runtime identification and with throwable exceptions

Optimize Classes

- For example, use J2ME with device configurations when programming in Java

Summary

We learnt

- Memory optimization saves memory without effecting system performance
- Use various standard ways for optimizing the memory needs in the system
- Number of methods exist for optimizing memory needs

End of Lesson 14 of Chapter 8