Memory management in real-time systems

Plan

1. Real time systems
2. Representation of different data types in memory
3. Memory management in real-time systems
4. Conclusion

Real-time systems

Real-Time Operating Systems are systems in which certain processes or operations have guaranteed minimum and/or maximum response times. That is to say, the system ensures that it will complete operation x after time t but before time t2, whatever t and t2 are, without fail, even at the expense of other lower priority operations

Real-time systems

Speed, in and of itself, is not critical; the primary goal is predictability. A response time less than t may be just as bad as, or worse than, one greater than t2.

Real-time systems

Real time operating systems are, as a general rule, only used in time-dependent embedded applications; general-purpose systems rarely if ever need to meet real-time constraints. When the do occur, real-time considerations may rule out the use of some common techniques, such as virtual memory, which may make the system's behavior less deterministic. Best-effort real-time functionality can however be useful in general purposes systems for supporting the needs of applications like digital audio workstations which demand both reliability (live recording) and low latency (real-time synthesis/processing), often at the same time.

Representation of data types in memory

Smallest unit of memory is bit: it holds 0 or 1, and nothing more. Every data in memory is just a collection of bits, 0s and 1s. To do that, computers convert out data to it’s own representation. Let’s look at different data types:

1. Boolean
2. Integers
3. Floating point numbers (real numbers)
4. Strings

Representation of data types in memory

Boolean types have one of two values: true or false.

If the value is true, it’s represented as 1 in memory, and if it’s false, it gets converted to 0.

So, just 1 bit of memory is needed to store a boolean value. But in practice, processors spare 1 bite (8 bits) to this type, because it makes easier to calculate the address of next values.

Representation of data types in memory

Integers are stored in memory in binary form. For example, decimal value 24 is 11000 in binary form. Representation of negative numbers depend on the size of the allocated memory to the value. If the number of bits for the value a is n, then -a should be represented such that a + (-a) = 0.

Integers. In real life, integer values are don’t have a limit. But memory has, so it puts it’s own limits to integers. Most programming languages have different types of integers for us to choose: int8, int16, int32, int64. The more memory it gives for the value, the larger numbers it can handle.

Representation of data types in memory

Floating point numbers are converted to scientific notation first.

123.4 in scientific notation becomes 1.234*10^2

Then, we have a number in the form m*10^e, where m and e are both integers, they are converted to binary as a regular integer.

In our example, m=1234 (10011010010) and e=2 (10), so in memory it’s represented as

10011010010 00000010

Representation of data types in memory

Strings or text values, are a collection of characters. Every character first coded to an integer value, and this value is stored in binary.

One of the most common coding schema is ASCII which contains all digits, punctuation marks, upper/lower case letters of English alphabet. This is enough for most of the programs, but it doesn’t have any other alphabet except English.

That’s why, nowadays most computers use UNICODE coding schema which is a superset of ASCII and can contain up to 2^32 characters

Memory management in real-time systems

Memory allocation is more critical in a real-time operating system than in other operating systems.

First, for stability there cannot be memory leaks (memory that is allocated but not freed after use). The device should work indefinitely, without ever needing a reboot. For this reason, dynamic memory allocation is frowned upon. Whenever possible, all required memory allocation is specified statically at compile time.

Memory management in real-time systems

Another reason to avoid dynamic memory allocation is memory fragmentation. With frequent allocation and releasing of small chunks of memory, a situation may occur where available memory is divided into several sections and the RTS is incapable of allocating a large enough continuous block of memory, although there is enough free memory. Secondly, speed of allocation is important. A standard memory allocation scheme scans a linked list of indeterminate length to find a suitable free memory block, which is unacceptable in an RTS since memory allocation has to occur within a certain amount of time.

Because mechanical disks have much longer and more unpredictable response times, swapping to disk files is not used for the same reasons as RAM allocation discussed above.

Conclusion

- In Real-Time Systems, time is a very important factor
- All data types are stored in binary form in the memory
- RTS programs should not dynamically allocate memory; all allocations should be static at compile time.