C S4680-101_EMBEDDED SYSTEMS (SPRING 2021)

Project 8: Introduction to ACX - A Cooperative eXecutive

Background

A a general purpose operating system (OS) is essentially a program that provides **services** to other programs (or "processes" when in their running state) through resource management. By centralizing commonly required services an operating system provides an abstraction of the underlying hardware resources, provides for software re-use, and improves reliability and maintainability. Resources managed usually include:

- Processor (scheduling)
- Memory (stack, heap management)
- I/O devices (including storage)
- System utilities (e.g. inter-process and network communication)

There are many kinds of operating systems. Operating systems for general-purpose computer systems are often designed to support:

- multiple users
- multiple processes per user
- multiple threads ("light-weight" processes)

Sometimes these operating systems may support multiple processors as well. General purpose operating systems provide a rich set of services to user and kernel processes.

Operating systems designed for special-purpose computer systems (including embedded systems) may have different goals. For example, these might include:

- support for real-time (deterministic) processes/threads
- support for safety-critical systems (e.g. redundancy, fault-tolerance/recovery)
- support for very low-power systems
- support for very low-cost systems (small memories, inexpensive processors, etc.)

Once the goals of an operating system are established, its capabilities are reflected in its underlying data structures. For example, a general purpose OS may use data structures that provide maximum dynamic flexibility in process management. Lists, queues, and trees provide flexibility and performance at the cost of memory use. Virtual memory provides a very useful abstraction for programs while incurring a significant hardware and system software cost. Embedded systems, on the other hand, may use statically defined quantities and simple data structures to reduce memory cost and/or to provide timing determinism. General purpose operating systems are assumed to be "preemptive" in their operation, meaning that processes and/or threads may be preempted at any point in their execution by the OS in order to place a higher priority process into execution. This is called rescheduling. Sometimes the term "operating system" is assumed to imply "preemptive scheduling". It is possible for a (simple) operating system to be non-preemptive--meaning that processes must cooperate in sharing of the CPU by invoking some OS function during their execution and avoiding busy-wait loops. This kind of "operating system" is sometimes called an "executive". We will be learning to use a simple multi-threading executive for our next project. (Graduate students will be implementing

ACX - A Cooperative eXecutive

ACX is a very simple multi-threading cooperative executive. It supports the following:

- Up to 8 threads (at least 1)
- Independent thread stacks, allocated statically
- Stack overflow detection
- Three bits of thread control per thread, residing in three 8-bit variables (**disable**, **suspend**, **delay**)
- 16-bit delay counter per thread
- Global system tick counter (32 bit)

- System timer with 1 msec resolution

Kernel State

In order for ACX to support threads and thread scheduling it needs some basic data structures. These will include variables for:

- thread stacks
- thread stack control (saved SP and base SP values)
- thread delay counters
- thread state variables (disable, suspend, delay)
- current thread ID
- global system tick counter

There will also be a couple of useful typedefs and #defines.

The **ACX API** provides brief descriptions of ACX functions and data.

ACX.zip contains header files and libraries for using ACX and ACXSerial functions. (ACXserial provides a set of serial port functions that provide character FIFOs and that use ACX rescheduling calls when waiting to read/write data. This allows other threads to run while the calling thread is waiting for a serial I/O operation to complete. The FIFO buffering allows writing in such a way that blocking is less likely. Serial I/O is interrupt driven.

Notes

"Canary values" are placed in thread stacks. The canary values may be placed at the very last byte of each thread's stack (lowest addressable byte of the stack). It is fairly straightforward to check these using the stack base pointers in the stack control structures. If stack overflow is detected, ACX enters an error state that blinks the onboard LED to indicate which thread stack was overwritten.

Before returning, the **x_init** function copies the call stack (return address, etc.) into the Thread 0 stack area, and sets SP to point to this stack area so that when it returns the calling function will become Thread 0.

Problem

Implement the problem of Project 7 ("Button Flash") using ACX, without any interrupts (except the timer interrupt built into ACX). You may use up to 8 concurrent threads (or any number if using substitution with x_new) and any of the ACX functions. **Your goal is to develop a solution that is as simple and understandable as possible.** In general using more threads can make a solution simpler, up to a point. You can decide what makes the solution easiest and most maintainable. Remember that you can also replace threads using the **x_new** function. **Consider ways to make your solution more general by making it easy to update the flash and/or rotate intervals and patterns.**

Submission

Demonstrate your project to your instructor.
Upload a zip file called ACX_ButtonFlash.zip containing your entire solution directory.

Submission status

Submission status	Submitted for grading
Grading status	Not graded
Due date	Tuesday, April 6, 2021, 11:55 PM
Time remaining	Assignment was submitted 13 days 22 hours late
Last modified	Tuesday, April 20, 2021, 10:02 PM

File submissions	Project08.zip	April 20 2021, 10:02 PM
Submission comments	Comments (0)	

Edit submission | Remove submission

You can still make changes to your submission.

◀ Project 7: Digital I/O with Buttons and LEDs

Jump to...

Final Project: Temperature Control System ▶