CG2271 Real Time Operating Systems

Tutorial 1

1. List and explain the differences between an RTOS and a standard operating system like Windows or MacOS. Why do these differences exist?

   RTOS: Small, highly customizable, emphasis on reliability and consistent timing. This is because most microcontrollers have limited processing power and memory capacity, so the ability to customize keeps footprint low. Reliability and consistent timing allow us to place guarantees on execution bounds so that strict time limits in hard real time systems are not violated.

2. In the RTOS Application Overview, we looked at an example application that was broken into 3 tasks to run on an RTOS. We could've just created a single application and run it WITHOUT an RTOS. What are the relative advantages and disadvantages of the two approaches?

   Without RTOS: Simpler, smaller foot-print, probably faster.
   With RTOS: Guaranteed timing bounds (note: does not mean that timing constraints are met; this only means that we always know the maximum amount of time it takes to do something). We can also extend the application to include driving displays, reading keyboards, dealing with other sensors and actuators, etc, more easily than trying to extend a single program.

3. What are the key differences between I/O ports on desktops and servers, against those found on microcontrollers? Explain why these differences exist.

   I/O ports on desktops and servers have to deal with a wider range of devices, e.g. display devices, communication devices, TV cards, signal processing cards, etc, and thus have a very general architecture.

   MCUs deal mostly with sensors and other MCUs, allowing us to have GPIO to deal with digial I/O, ADC and PWM to deal with analog input/output, TWI, USART, etc to deal with inter-MCU communication, etc. In addition all ports are on the MCU to simplify designing for the chip and lower costs for the user.

4. Given a printer that takes 10 ms to print a character, and given a CPU with a 100 MHz clock, how many clock cycles would be wasted in "programmed I/O" if we printed a 1,000 character document?

    1000 characters x 10 ms = 10 seconds spent polling the printer. This is quivalent to 1 billion clock cycles, or about a billion wasted instructions if each instruction takes one cycle.

    Now suppose we have the same printer and CPU, and it takes 25 ns to process an interrupt. How many clock cycles are used in total for interrupt processing if we printed the same document?

    Each character triggers an interrupt which needs 25 ns to process. 1000 characters will trigger 1000 interrupts, and total time spent is 25 microseconds. 1 cycle = 10ns, so this is equivalent to 2,500 clock cyles.

5. Comment on the efficiency of interrupt I/O vs. programmed I/O and why interrupt I/O is preferred. What are the disadvantages of interrupt I/O?

    Although it still takes 10 seconds to print the entire document, in interrupt driven I/O only 2,500 clock cycles out of 1,000,000, 000 are used to process interrupts related to the printing. The remaining cycles are used to execute code from other processes, etc. This is not the case with programmed I/O where the 1,000,000,000 cycles are spent "busy waiting".

    Unlike programmed I/O which just needs a status register that the program can read, interrupt driven I/O is inherently more complicated as the printer must be able to assert an IRQ on the CPU, and then monitor until the CPU acknowledges the printer's request, at which time the printer can de-assert the IRQ line. This is harder to design and takes more hardware.