Lecture 05 Notes – Synchronization and Scheduling and Dispatch, Scheduler Wish List

I. Overview

A. **Review**

1. Where does this fit into the big picture?
2. Sync and Do – How to "and"?
   a. Deferring work from ISR to thread with shared variables (notification, data)
   b. Generalizing sync behavior. Consider abnormal/edge cases (missing events, etc.)
3. Sync and Don't – Mutual Exclusion
   a. 2nd form of synchronization
   b. Don't overlap execution of critical sections for same object/resource

B. **Today**

1. Closer look at Sync/Comm/Sched
   a. Stages
   b. Options and implementations
   c. Basic response time analysis
2. Scheduler Wish List

II. Closer look at Sync/Comm/Sched

A. **Where does this fit into the big picture?**

B. **Processing chain options:**

1. Various options available for each step/slot. Can implement in software and/or hardware peripherals and/or interrupt system (advanced: and/or DMA and/or peripheral interconnect)
   a. Which to use?
      i. Software:
         - + Easier for simple systems, easy to debug
         - – Poor speed, timing stability and fairness without scheduler limit system scalability
      ii. Hardware:
         - + Extremely fast, stable timing, and power-efficient,
         - – Specialized functionality (not universal like software), takes time to learn hardware peripheral capabilities

b. One objective of class – give you an idea of what hardware can do, so you know you may have more options than just software
    i. Unknown unknowns: You don't know that hardware can do far more than basic functionality of synchronous programmed I/O (e.g. read input port now, analyze the reading)
    ii. Known unknowns: You know that the port peripheral can do a lot beyond basics (read input bits, write output bits), but don't know what exactly.
    iii. Known knowns: You know some peripherals can automatically detect an input signal's edge and
        - Request (using the interrupt system) for its interrupt service routine to run (PORT)
        - Request (using the direct memory access controller system) a data transfer among peripherals and memory (PORT, DMA)
        - Signal to another peripheral that an input event has occurred (CMP -> ADC)
        - Make a running timer peripheral capture the current counter value (Timer)

C. **Processing Chain Refinement**
1. Problem and Adjustment
    a. Previous steps in chain (Lecture 04): **Detect, Schedule, Dispatch, Do Process Work** (e.g. handle event)
        i. Part 1: Synchronize – Should processing take place (eventually)?
            - **Detect** gives or takes away permission to scheduler to run this processing (e.g. handler code)
        ii. Part 2: Do – Select processing to do, run it:
            - **Schedule** decides which processing has permission (is ready) to run and should be done next. Currently based on program order, later will enhance to consider other factors (priority, time, etc.)
            - **Dispatch, Do Process Work** (e.g. handler)
    b. Adjustment: Refine **Detect**
        i. Why? Some of Detect must be in hardware, while remainder may be hardware and/or software.
        ii. How? Split it up into two parts to match this division
            - **SQ**: **Sample** input and **Quantize**/digitize that sample. Must be done by hardware circuit.
            - **AD**: **Analyze** value and **Decide** if event occurred. **Scheduler** will use decision.
    c. New processing chain: **Sample & Quantize, Analyze & Decide, Schedule, Dispatch, Do Process Work**

2. **Understanding Synchronization with SQ and AD**. Should processing take place (eventually)?
    a. **Detect** is split into
        i. **Sample & Quantize** (Digitize) input (**SQ**): Must be hardware for input signals
            - **Sample**: capture input value at specific time (voltage may be analog or binary digital)
            - **Quantize/digitize** sample into digital form **cur_val** (single or multiple bits)
                - Single bit
                    - Digital input -> Port, GPIO,
                    - Analog input to comparator
                - Multiple bits
                    - Analog to Digital converter
        ii. **Analyze & Decide (AD)** if event happened: Software and/or Hardware
            - **Analyze** signal (past and present values, may depend on system state)
                - **Software** Rising Edge detection:
                    - if cur_val different from prev_val (previous cur_val) then diff = 1 else diff = 0 (simple logic version: diff = cur_val XOR prev_val)
                    - prev_val = cur_val
                - **Hardware** Rising Edge detection:
                    - PORT module compares current SQ value with previous one. Set port interrupt status flag (ISF) to 1 in PORT's pin control register (PCR), else leave ISF unchanged
            - **Decide** if event happened which affects handler code being able to run, save that decision for scheduling.
                - **Software** Rising Edge detection:
                    - if diff ==1 and cur_val == 1, then then tell the scheduler that the process is allowed to run
                - **Hardware + Software** Rising Edge detection: Still needs software.
                    - PORT interrupt is disabled, so code reads PORT's pin control register, tests the interrupt status flag (ISF). If 1 (event detected), then tell the scheduler that the process is allowed to run
                - **Hardware** Rising Edge detection:
                    - PORT interrupt is enabled, so sends interrupt request (IRQ) to interrupt controller (NVIC)

3. **Scheduling** and Dispatching: Decide what to do next, and start it
    a. Dimensions to consider…
        i. Where is code for scheduler? May be implicit (integrated into natural sequential flow of program) or explicit scheduling operation (integrated into program or call to OS)
        ii. One or multiple processes/handlers to consider?
    b. Decision tree
        i. Is this process allowed to run?
            - Yes: Dispatch and run it
            - No:
                - Blocking: Repeat software to analyze and decide (loop back to it)
                - Non-blocking: Advance to next process (via scheduler or program structure) and repeat decision tree analysis
                -

D. **Where is code to do sync/sched/dispatch of process work, and how do we get it to run?**
1. Can be in threads and/or interrupt/exception handlers
2. Two possible locations for sync/sched/dispatch. May have some in both.
    a. Sync/sched/dispatch code **integrated** (may be **implicit**: defined by parts of program's control flow)
        i. Code for scheduling operations is integrated into program source code, executes from main thread (and maybe interrupt/exception handlers).
        ii. This code runs when function gets to those integrated operations.
    b. A **separate module** with scheduler/kernel (often part of OS or RTOS)
        i. Code for scheduling (and related operations) is in different files/modules from application program
        ii. This code runs when **application program** or **OS function** invokes OS (macro, subroutine call or software interrupt (SVC)).
        iii. We will see OS often uses interrupt and exception handlers for key features (e.g. timer tick)

E. **Basic Response Time Analysis**
   1. For **which** process?
      a. This particular process
      b. All the other processes
   2. Basic Idea
      a. **Add up** how long everything else (other processes, Sync/Sched/Dispatch) might take in the best and worst cases to get bounds on best-case and worst-case response times
      b. **Blocking** vs. **Non-blocking** scheduling tests
         i. Blocking? If event never happens, will take infinite time to pass test.
         ii. Non-blocking? Finite (short!) maximum time to pass test.

F. **Observations:**
   1. Sync/comm/sched/dispatch operations are often **interdependent**, so
   2. If scheduler/OS can see all these operations, it can make better decisions and offer more features
      a. Example: If process event test will **block**, then pause process execution and automatically switch in another process. Reclaims Idle Time.

III. Scheduler Wish List

A. **Allow better responsiveness where needed**
   1. Dynamic task execution order
   2. Allow Process Prioritization to drive scheduling
   3. Allow Process Preemption by higher priority processes

B. **Generalize/standardize code structure for modular code**
   1. Move many/most scheduling decisions from user code to scheduler code

C. **Features to simplify programming and reclaim idle time**
   1. Provide mechanisms for synchronization: Let software processes trigger each other with event flags, semaphores, etc. Provide mutexes, etc.