

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

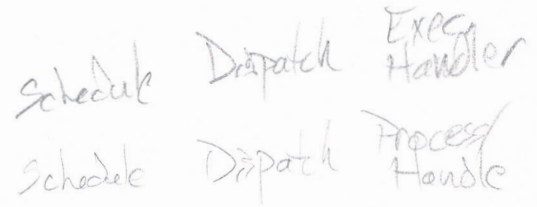
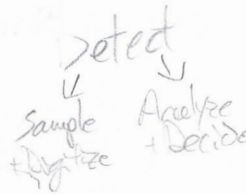
B. Today

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. Processing chain refinement:

1. Where does this fit into the big picture?
2. Slot machine selector for processing chain
 - a. Diagram: Sample & Digitize, Analyze & Decide, Schedule, Dispatch, Process event (e.g. Handler)
3. Various options available for each 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:
 - o + Easier for simple systems, easy to debug
 - o – Poor speed, timing stability and fairness without scheduler limit system scalability
 - ii. Hardware:
 - o + Extremely fast, stable timing, and power-efficient,
 - o – Specialized functionality (not universal like software), takes time to learn



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, but don't know what exactly.
 - iii. Known knowns: You know some peripherals can automatically detect an input signal's edge and
 - o Request (using the interrupt system) for its interrupt service routine to run (PORT)
 - o Request (using the direct memory access controller system) a data transfer among peripherals and memory (PORT, DMA)
 - o Signal to another peripheral that an input event has occurred (CMP -> ADC)
 - o Make a running timer peripheral capture the current counter value (Timer)

B. Processing chain has two groups of steps

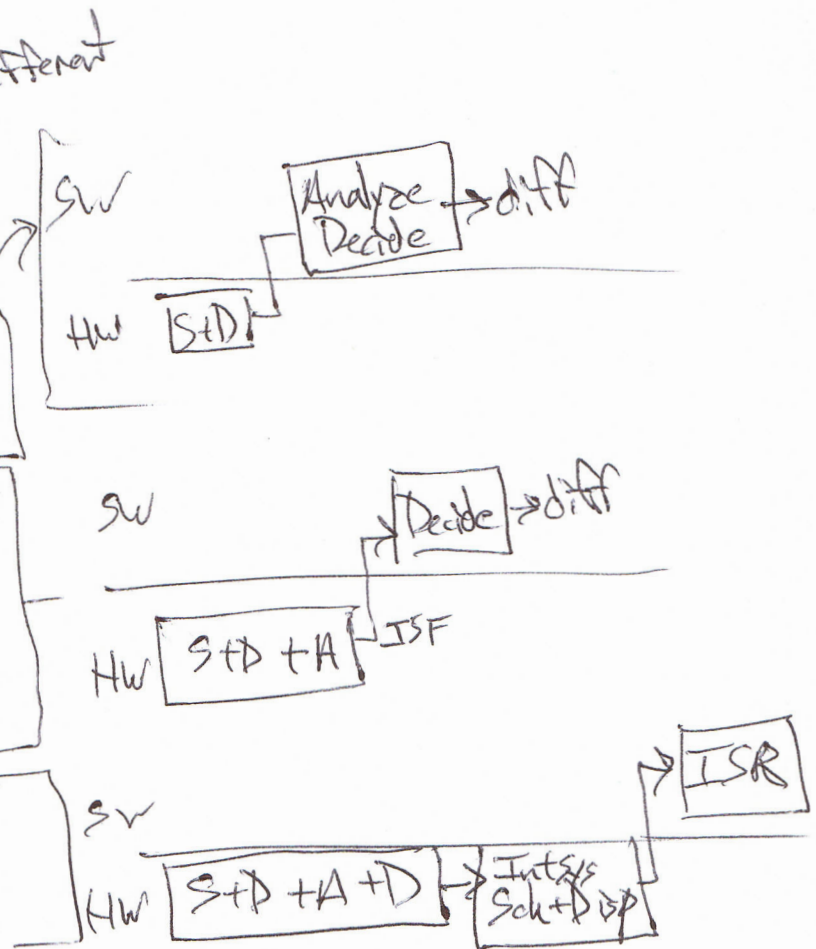
- 1. Synchronization: Should processing take place (eventually)?
 - a. Gives or takes away permission to scheduler to run this processing (e.g. handler code)
 - b. Steps:
 - i. Sample & Digitize input: Must be hardware for input signals
 - o Sample: capture input value at specific time
 - o Digitize sample into digital form (single or multiple bits)
 - Single bit
 - Digital input -> Port, GPIO,
 - Analog input to comparator
 - Multiple bits
 - Analog to Digital converter
 - ii. Analyze & Decide: Software and/or Hardware
 - o Analyze signal (past and present values, may depend on system state)
 - Software Rising Edge detection: if current S&D value

different from previous S&D,
then $\text{diff} = 1$ else $\text{diff} = 0$

- Hardware Rising Edge detection: PORT module compares current S&D value with previous S&D. If ~~same~~ set port interrupt status flag to 1 in PORT's pin control register (PCR), else 0

- Decide if event happened which affects handler code being able to run

- Software Rising Edge detection: if $\text{diff} == 1$ and current S&D value is 1, 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)



2. Scheduling and Dispatching: Decide what to do next, and start it

a. Dimensions...

- May be implicit (sequential program flow) or explicit scheduling operation (integrated into program or call to OS)
- One or multiple handlers to consider?

b. Decision tree

- 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

C. What code does sync/sched/dispatch and where is it? In thread and interrupt/exception handlers

- Start-up:** a CPU thread runs **Reset interrupt handler** to set up system, switch to thread mode and start running **main()** function in C program.

2. **After start-up:** two possible locations for sync/sched/dispatch, may be 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).
 - b. A **separate module** with scheduler/kernel (often part of OS or RTOS)
 - i. Code for scheduling operations code is in separate files
 - ii. OS code for operation runs when
 - o Invoked by application program with subroutine call or software interrupt. May be main thread, interrupt/exception handler, other threads (if provided
 - o Application program Interrupt/exception handler invokes OS operation
 - iii. Program source code can invoke OS operations through subroutine calls or software interrupts (SVC)
 - iv. OS code executed by main thread, ISRs/exception handlers

D. Basic Response Time Analysis

1. Blocking
2. Non-blocking...

E. Observations:

1. Sync/comm/sched/dispatch operations are often interdependent
2. If scheduler/OS can see all these operations, it can make better decisions and offer more features
 - a. Pause process execution when operation will block, switching in another process
 - b. Let software processes trigger each other with event flags, semaphores, etc.
Provide mutexes, etc.

III. Scheduler Wish List

- A. **Allow better responsiveness where needed**
 1. Dynamic task execution order
 2. Process Prioritization
 3. Process Preemption
- B. **Standardize code structure for modular code**
- C. **Features to simplify programming and reclaim idle time**