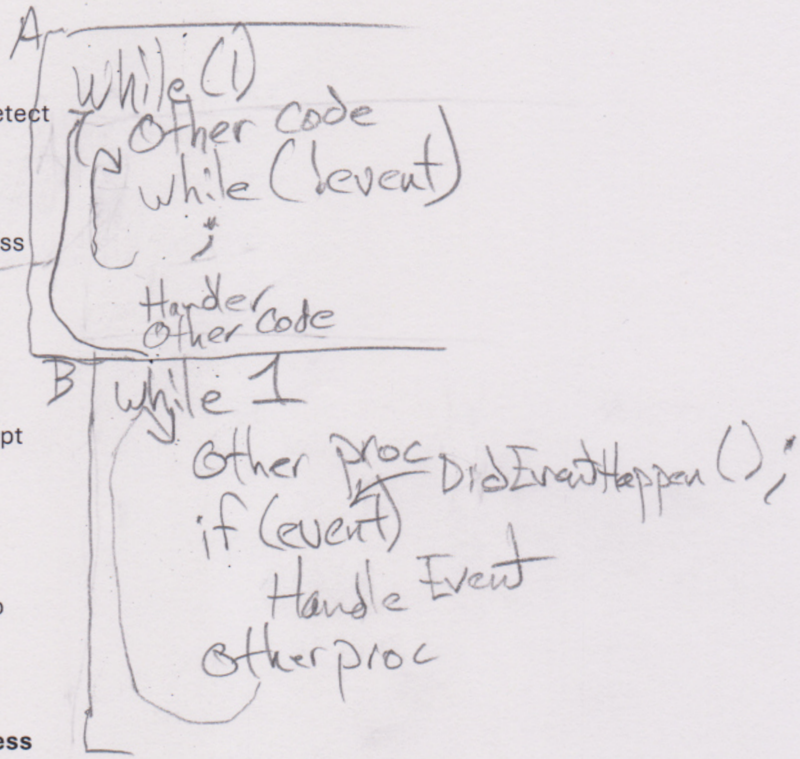
460 – 1A

Lecture 04 Notes –Synchronization, Communication, Mutual Exclusion

I. **Overview**

A. **Review**

1. Looked at Sync and Do for triggering processing from events, time
2. Sync for triggering handler process = Detect event + (schedule handler process + dispatch handler process) + execute handler process
3. Basic Sync methods for software process
   a. Blocking polling loop
   b. Non-blocking polling test within scheduler loop
   c. HW detects event (e.g. edge, serial event)
   d. HW detects event, requests interrupt service



B. **Today**

1. Refining Definitions
2. Sync and Do – How to "and"?
3. Sync and Don't – Mutual Exclusion Intro

II. **Refining Definitions**

A. **Meaning of "Process"**

B. **Our context (concurrent systems): Process does something in software or hardware**

1. HW Process: one state machine controlling behavior of data path.
2. SW Process: one stream of instructions executed sequentially.
   a. One **thread of execution**: single control flow, one program counter to specify next instruction. No splitting to go down multiple paths simultaneously.
3. SW Processes in C program w/o additional scheduler support
   a. One thread executes main function which never ends/returns, stopping only at power-off or sleep
   b. Interrupt/Exception handler threads (ISRs)
      i. Are threads which always end.
      ii. Rely on interrupt controller to
         - Preempt CPU execution of a lower priority thread (main, lower priority ISRs)
         - Later resume execution of preempted thread
   c. All SW processes able to access all of memory (instructions, data, peripheral

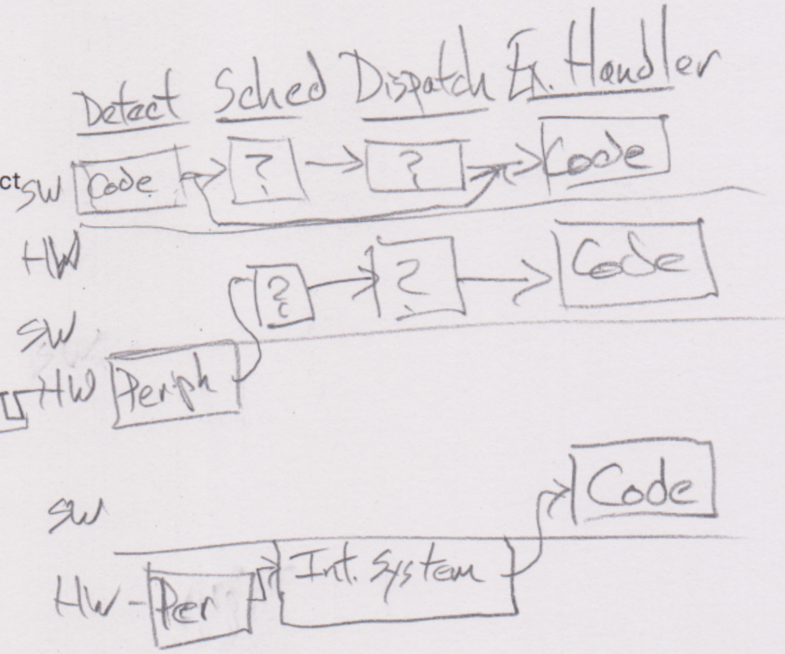
460-1B

Lecture 04 Notes –Synchronization, Communication, Mutual Exclusion

I. **Overview**

A. **Review**

1. Looked at Sync and Do for triggering processing from events, time
2. Sync for triggering handler process = Detect event + (schedule handler process + dispatch handler process) + execute handler process
3. Basic Sync methods for software process
    a. Blocking polling loop
    b. Non-blocking polling test within scheduler loop
    c. HW detects event (e.g. edge, serial event)
    d. HW detects event, requests interrupt service



B. **Today**

1. Refining Definitions
2. Sync and Do – How to "and"?
3. Sync and Don't – Mutual Exclusion Intro

II. **Refining Definitions**

A. **Meaning of "Process"**

B. **Our context (concurrent systems): Process does something in software or hardware**

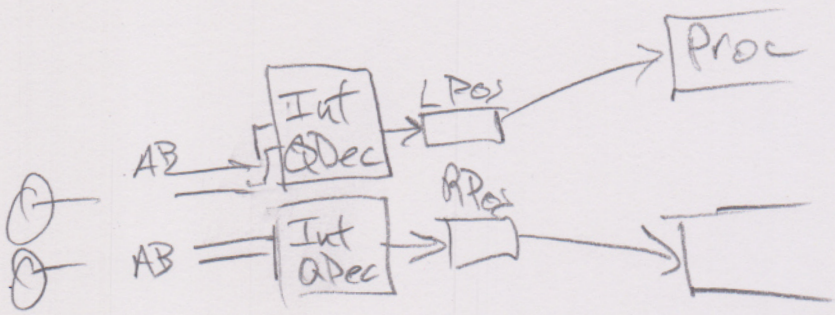
1. HW Process: one state machine controlling behavior of data path.
2. SW Process: one stream of instructions executed sequentially.
    a. One **thread of execution**: single control flow, one program counter to specify next instruction. No splitting to go down multiple paths simultaneously.
3. SW Processes in C program w/o additional scheduler support
    a. One thread executes main function which never ends/returns, stopping only at power-off or sleep
    b. Interrupt/Exception handler threads (ISRs)
        i. Are threads which always end.
        ii. Rely on interrupt controller to
            - Preempt CPU execution of a lower priority thread (main, lower priority ISRs)
            - Later resume execution of preempted thread
    c. All SW processes able to access all of memory (instructions, data, peripheral

control registers) unless restricted (memory protection, privilege level, memory remapping hardware)

i. Threads can share data through variables with fixed addresses (global, static allocation)

C. **Other major context (Operating Systems):**

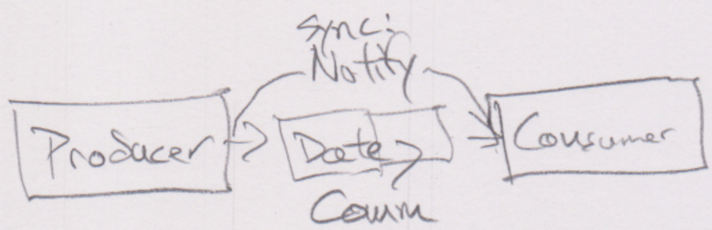
1. Not relevant in this class
2. SW Process:
    a. One or more threads of execution operating in one memory space.
    b. Threads able to access entire memory space of their process unless restricted
    c. Threads **not able** to access memory space of **other SW processes**.
        i. Often implemented by virtual memory system



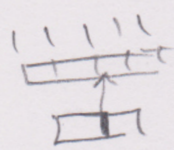
III. Sync and Do: How to "and"?

A. **What if we can't fit all the work into the ISR (or the process)?**

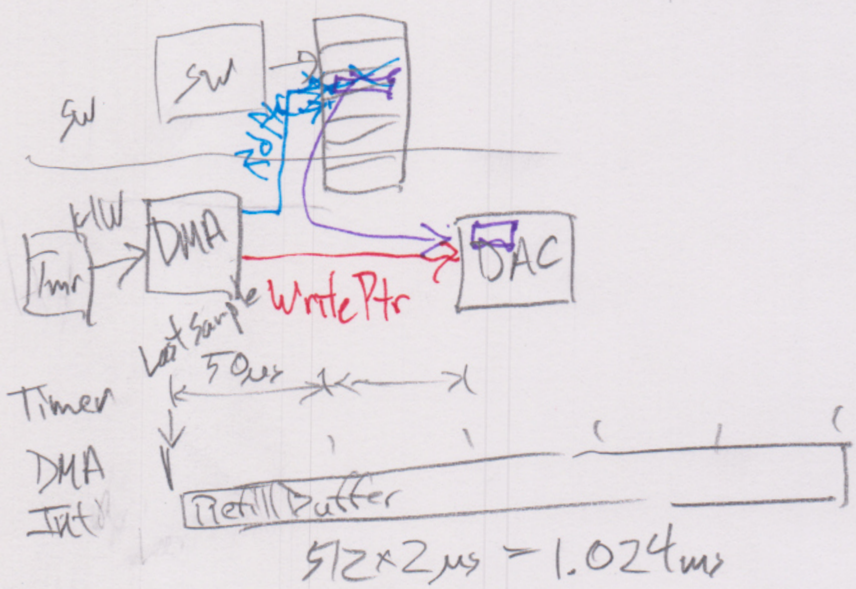
1. Why can't we?
    a. Structure Mismatch: Process need to sync or share data with each other
    b. Timing Impact: doing all the work in the ISR delays other processing too much
        i. Vulnerable timing: lower-priority ISRs, all threads



2. Need to split ISR/process into parts
    a. Producer
    b. Something to manage incomplete work
    c. Consumer
3. What do we need to share?
    a. Sync: something happened
    b. Communication: data describing what happened. Typically needs sync to let consumer know about new data. Exception: Get Latest Data
    Perio. Temperature update



4. Example: Waveform Generator
    a. Goal: generate analog waveform with precisely timed output updates (e.g. every 50 us)
    b. Implementation W7 – optimized to use timer, DMA, ISR to stabilize timing
        i. Every timer event (50 us apart) triggers DMA controller to transfer **next** sample from array (in memory array variable) to DAC data register
        ii. DMA requests interrupt as it does last transfer
        iii. DMA ISR has loop (calculate next sample, save to next location in buffer) to refill entire buffer
    c. Timing Challenge with W7
        i. ISR takes long time, delays other SW processing (lower priority ISRs, threads)

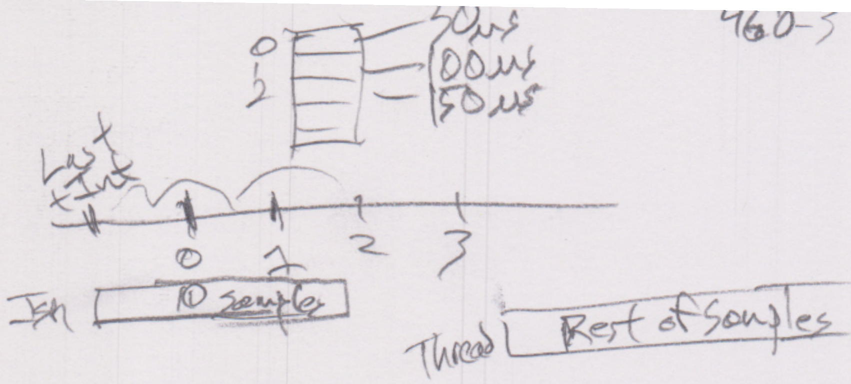


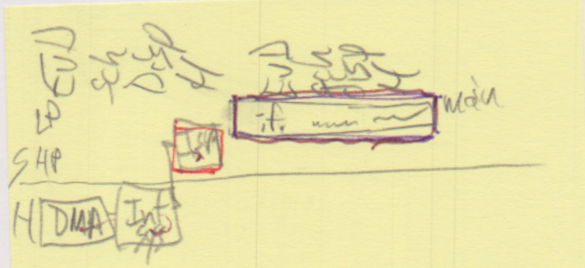
B. **Need to get information from ISR (producer) to main thread (consumer). How?**

C. **W7 Analysis**
   1. **Insight:** Data samples earlier in buffer are needed sooner (are more urgent) than later ones
   2. **Possible solution:** Do just the urgent buffer refill work in the ISR, defer the rest to a lower-priority thread.

D. **Implementation W9 will split work of ISR**
   1. DMA ISR is producer process. Just refills first U samples in buffer.
   2. Thread is consumer process. Refills remaining N-U samples in buffer.
      a. Somehow DMA ISR needs to tell main thread to finish refilling the buffer. Thread must **synchronize** with ISR.
   3. Is one example of **inter-process synchronization and communication.** Many others.

E. **How to make version W9**
   1. Basic Structure
      a. ISR: DMA completion
      b. Main thread loop: Does work for other parts of system: reading user interface controls, updating indicator LEDs, etc.
   2. ISR timing is **asynchronous** to main thread loop
      a. Don't know if ISR ran, so can't just refill buffer every time around the main loop
   3. Modify Structure
      a. Add a Shared Variable: Event Flag
         i. 1 = it happened,
         ii. 0 = nothing happened
      b. Processes
         i. ISR writes 1 to event flag
         ii. Main thread: While 1 loop
            - Tests each event flag **(non-blocking)**
            - If flag is 1, clear it to 0 and do processing
      c. Processing Chain Timeline
         i. Two processes
         ii. SW scheduler uses SW and HW (Ints)





P1 is Async to P2

P1
ISR or PreemptingTh

P2
Main thread (w/ Scheduler)

while 1

ISR
Write to SharVar

Shar Var

Read SharVar, test Handler

F. **Generalizations: Consider behavior for abnormal/edge cases**

1. OK for consumer to miss events (e.g. in event burst)? Get latest data?
   a. Yes: Count to 1, and no farther
   b. No: Use integer variable to count number of pending events (happened but not processed)
      i. Producer increments events_pending (ep)
      ii. Consumer decrements events_pending
2. OK to produce events if consumer hasn't consumed enough?
   a. Buffer size limits, etc.
   b. Producer needs to synchronize by checking events_pending before producing event
3. Implementation
   a. Decide how system should behave, add to requirements
      i. Hardware processes have behaviors defined for these cases
      ii. Very common for more embedded system requirements for **exception cases** than **normal operation**
   b. Implement the behavior
      i. Configure hardware (if available)
      ii. Bare metal (no SW support): algorithms in your code
      iii. Support from OS/RTOS or programming language. Semaphores (counting, binary)
4. Can also communicate data in shared variables
   a. Event Flag + Data Value = Synchronization + Communication
   b. Multiple pending events possible?
      i. Also need to save data for each event (queue, FIFO buffer)
      ii. How large to make buffer?
         - Depends on rates of data production and consumption, which depends on input events, time to execute processes, when/how many times processes get to execute
5. Deeper look at triggering sync behaviors possible.
   a. Can producer process generate another event if consumer process hasn't gotten it yet?
      i. No: Lock-step
      ii. Yes: How many events are possible

480-5

   b. Counting? Track number of pending, unserviced events...

IV. Sync and Don't: Mutual Exclusion

- A. **Intro**
  1. Another form of synchronization
     - a. Prevent Z from happening at a bad time
     - b. If A has happened but B hasn't, then don't let Z execute until B has happened
       - i. A begins "critical section"
       - ii. B ends "critical section"
- B. **Motivating Example 1: Two processes updating shared variable**
  1. Processes increment shared global variable operations:
     - a. P1 read/modify/write,
     - b. P2 read/modify/write
  2. Failure cases in slide
  3. OS Support Side Note
     - a. OS provides OS-managed objects (e.g. counting semaphore)
     - b. These are protected from corruption by requiring OS calls to access them.
     - c. The functions for the OS calls contain critical sections which are protected correctly.
- C. **Motivating Example 2: Motor Position/Speed controller with Zero Limit Switch**
  1. Processes:
     - a. P1 QD: increment or decrement position. Read/Modify/Write
     - b. P2 ZLS: Zero out position. Write.
  2. Failure Cases
     - a. Receive QD pulses while ZLS is closed? Add test to see if ZLS is closed(?)
     - b. ZLS interrupt during QuadDec Inc/Dec of position variable
       - i. **after read starts** (includes during modify) **AND before write starts**
- D. **Solutions**
  1. Prevent preemption
     - a. Interrupts
     - b. Thread scheduling
     - c. Mutual exclusion
  2. Support
     - a. Hardware
     - b. Instructions & Algorithms
     - c. OS/RTOS, Programming Language?