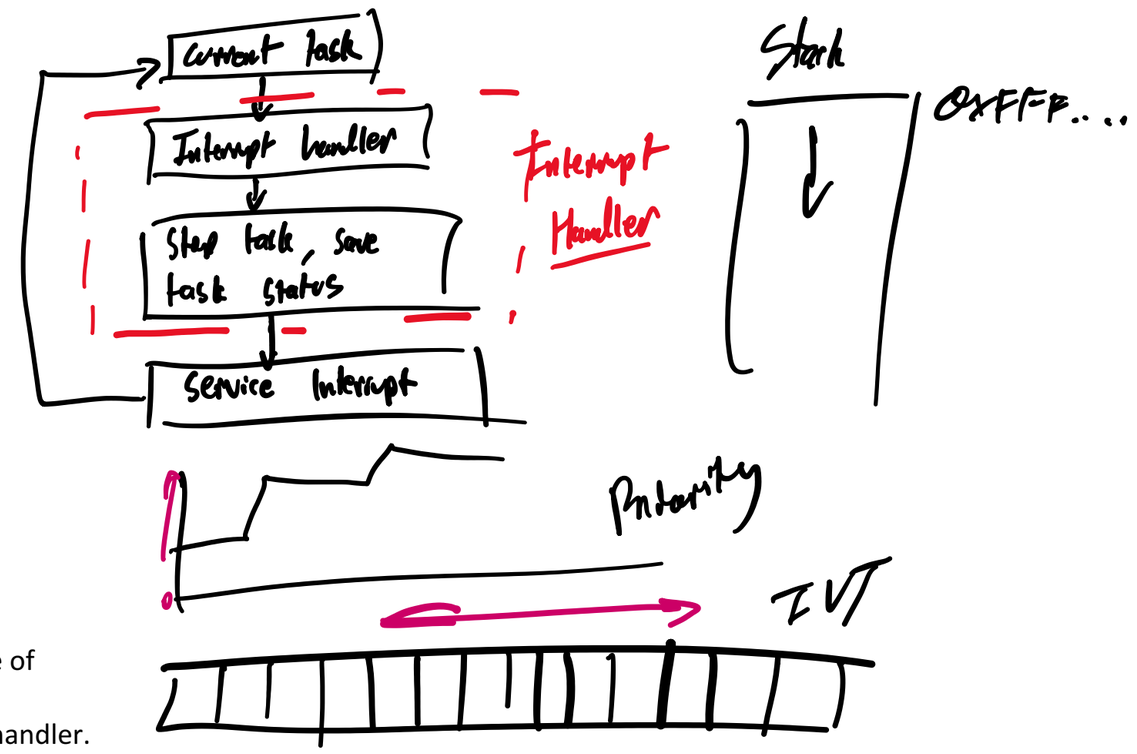
Software Interrupts.

Thursday, March 2, 2023 1:06 PM

Nikunj Parasar
Gaucho Racing UCSB

- Polling Vs. Interrupts
  - Polling: continual checks, like standing in front of the phone all day
  - Interrupt: like only picking up phone when rings, other processes not affected

- An interrupt is a signal sent from a device or from software to the OS
- Causes the OS to temporarily stop what its doing and service the interrupt using the interrupt handler
- If required, less important functions are overridden or delayed
- A computer can technically only run one program at a time
- The ability for applications to interrupt each other gives the illusion of multitasking
  - In reality, thy are time sharing, using scheduled interruptions
- Interrupt Handler
  - Handles the interrupt signals as they are received
  - Prioritizes the interruptions
  - Places them into a queue to be handled
- **interrupt vector table (IVT)**
  - data structure that associates a list of interrupt handlers with a list of interrupt requests in a table of interrupt vectors.
  - Each entry of the interrupt vector table, called an interrupt vector, is the address of an interrupt handler.



- What happens when an interrupt/exception occurs
  - CPU saves current context
    - Program counter, registers, etc.
  - CPU grabs address of interrupt service handler from a Vector Table
    - Each interrupt has an index into the table
  - CPU jumps to interrupt service handler
  - Interrupt service handler does operations
    - Clears flags, … etc
  - Interrupt Service handler exits
  - CPU restores context and returns to main flow of program

- Interrupts in Teensy 4.1 Arduino for the VCU
  - Every interrupt has a flag bit, this is set by hardware when the interrupt trigger condition occurs
    - Flag remembers the interrupt condition has occurred until it has been handled by software
  - Flag bit set even if interrupts not used
  - Every interrupt also has mask bit
    - Enables or disables the individual interrupt
  - To configure an interrupt:
    - Configure the peripheral
    - Reset the interrupt flag
    - Set interrupt mask
    - Enable global interrupt with sei()
- Syntax
  - AVR Interrupt header
    - #include <avr/io.h>
    - #include <avr/interrupt.h>
  - These headers will define the ISR() macros for each possible interrupt routine/vector
  - Example:
    - Handling Timer 0 Overflow
      - ISR(TIMER0_OVF_vect){//code here}
- Design strategy
  - Keep interrupt service routines short and simple
  - Nested interrupts
    - Some interrupt service routines enable the global interrupt with sei()
    - Usually done when an interrupt may take long to execute
    - Use caution to not allow already in service interrupt to trigger again
      - This would cause infinite calling until all memory is overwritten
      - Generally safest to not call sei() in any interrupt service
- Shared variables
  - Must use volatile keyword
    - Instructs compiler to always access the variable
  - When accessing shared variables, make sure that you do not get incorrect results by another interrupt being called in the middle of the service
    - Use cli() to disable and then sei() after to re-enable after the operation
    - Making local copies of volatile variables is not a bad idea either to make sure that they save their state just incase
- Note: we can also disable individual mask bits if we don't want to affect other unrelated interrupt services

Interrupt Vector, Mask & Flag Names

ISR() Name: The name used with ISR() to define the interrupt service routine.

Mask: (byte,bit#) Bit that enables this interrupt. See accessing a single bit for C syntax to write byte,bit# pairs.

Flag: (byte,bit#) Flag indicates if the interrupt is pending. Many flags are reset by writing 1 (yes, that seems horribly backwards, but that's the way the hardware works). Most flags are automatically reset when the interrupt service routine is called.

ISR() Name	Mask	Flag	Function
INT0_vect	EIMSK,IINT0	EIFR,INTF0	Interrupt Request, External Signal
INT1_vect	EIMSK,IINT1	EIFR,INTF1	
INT2_vect	EIMSK,IINT2	EIFR,INTF2	
INT3_vect	EIMSK,IINT3	EIFR,INTF3	
INT4_vect	EIMSK,IINT4	EIFR,INTF4	
INT5_vect	EIMSK,IINT5	EIFR,INTF5	
INT6_vect	EIMSK,IINT6	EIFR,INTF6	
INT7_vect	EIMSK,IINT7	EIFR,INTF7	
PCINT0_vect	PCICR,PCIE0	PCIFR,PCIF0	Pin Change
PCINT1_vect	PCICR,PCIE1	PCIFR,PCIF1	
TIMER0_COMPA_vect	TIMSK0,OCIE0A	TIFR0,OCF0A	Timer 0 Compare A Match
TIMER0_COMPB_vect	TIMSK0,OCIE0B	TIFR0,OCF0B	Timer 0 Compare B Match
TIMER0_OVF_vect	TIMSK0,TOIE0	TIFR0,TOV0	Timer 0 Overflow
TIMER1_CAPT_vect	TIMSK1,ICIE1	TIFR1,ICF1	Timer 1 Input Capture
TIMER1_COMPA_vect	TIMSK1,OCIE1A	TIFR1,OCF1A	Timer 1 Compare A Match
TIMER1_COMPB_vect	TIMSK1,OCIE1B	TIFR1,OCF1B	Timer 1 Compare B Match
TIMER1_COMPC_vect	TIMSK1,OCIE1C	TIFR1,OCF1C	Timer 1 Compare C Match
TIMER1_OVF_vect	TIMSK1,TOIE1	TIFR1,TOV1	Timer 1 Overflow
WDT_vect	WDTCSR,WDIE	WDTCSR,WDIF	Watchdog Timer
USART1_RX_vect	UCSR1B,RXCIE1	UCSR1A,RXC1	USART Receive
USART1_TX_vect	UCSR1B,TXCIE1	UCSR1A,TXC1	USART Transmit Complete
USART1_UDRE_vect	UCSR1B,UDRIE1	UCSR1A,UDRE1	USART Transmit Ready
SPI_STC_vect	SPCR,SPIE	SPSR,SPIF	SPI Transfer Complete
ANALOG_COMP_vect	ACSR,ACIE	ACSR,ACI	Analog Comparison Change
EE_READY_vect	EECR,EERIE	(none)	EEPROM Operation Complete
USB_GEN_vect	(complex)	(complex)	USB Device Event
USB_COM_vect	(very complex)	(very complex)	USB Communication