Homework

- Reading
  - Review previous material on “interrupts”
- Machine Projects
  - MP4 Due today
  - Starting on MP5 (Due at start of Class 28)
- Labs
  - Continue in labs with your assigned section

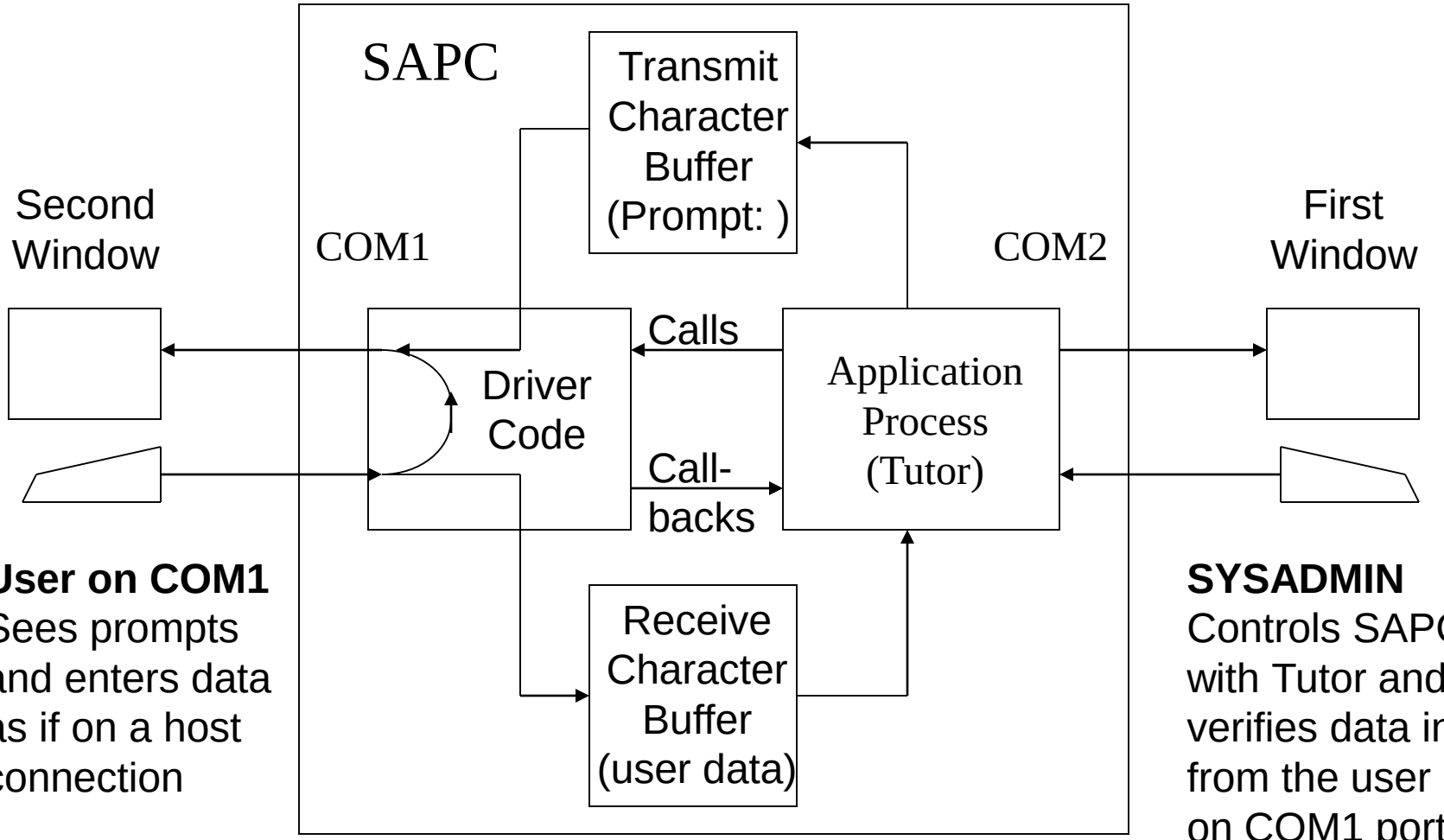
Discussion of MP4

- What did you learn?
- Did anyone do the optional software UART?
- Let’s look at the code for it as an exercise

Introduction to MP5

- Adding new code to provided tutor “cmds.c”
- Writing a COM1 port driver for Tutor to use
  - Started and stopped by the application (Tutor)
- Tutor cycles driver through this sequence:
  - Receives and buffers user entered data
    (with full duplex echo back to COM1 port)
  - Returns to callback function with receive data buffer
  - Transmits buffer of application data (prompt)
  - Returns to callback function when done

SAPC as Host to a User on COM1



User on COM1
Sees prompts and enters data as if on a host connection

SYSADMIN
Controls SAPC with Tutor and verifies data in from the user on COM1 port

What Code is Needed?

- In `cmds.c:`
  - The `spi` command function has been written for you
  - Write two call back functions
    - one for processing last interrupt in transmission and re-starting receiver interrupts
    - one for processing last interrupt in receiving and re-starting transmitter interrupts
- In `comintspack:`
  - Write init and shutdown for COM1 interrupts
  - Write an interrupt handler for IRQ4 (must handle either a transmit or a receive interrupt each call)

What's in cmds.c

- New PC-tutor command

  `spi <on|off>`

- Descriptions

  `spi on` calls `init_comints` to enable COM1 in transmit mode with transmit call back function (to print prompt first)

  `spi off` calls `shutdown_comints` to disable both transmit and receive interrupts

What's in cmds.c

- Receive callback function (`process_input`)
  - Process input completion (print buffer on COM2)
  - Disable input receiving via shutdown_comints()
  - Enable output transmission via init_comints()
- Transmit callback function (`process_output`)
  - Disable output transmission via shutdown_comints()
  - Enable input receiving via init_comints()
- These cause alternate COM1 transmit and receive

What's in comintspack.h?

- API symbolic constants

```
/* mode values */
#define TRANSMIT 0
#define RECEIVE 1
```

- API function prototypes

```
void init_comints (int mode,
  void (*callback)(char *),
  char *buffer,
  int size);
void shutdown_comints (void);
```

- You do NOT modify this file. Use it as-is!

What's in comintspack.c?

- Initialize COM1 port (`init_comints`)
  - Save callback function, buffer, and size in static memory
  - Clear out any characters already received
  - Set the interrupt gate
  - Enable the PIC for the IRQ4
  - For RX mode, enable RX interrupts in the UART's IER
  - For TX mode, enable TX interrupts in the UART's IER
- This function is called with interrupts disabled

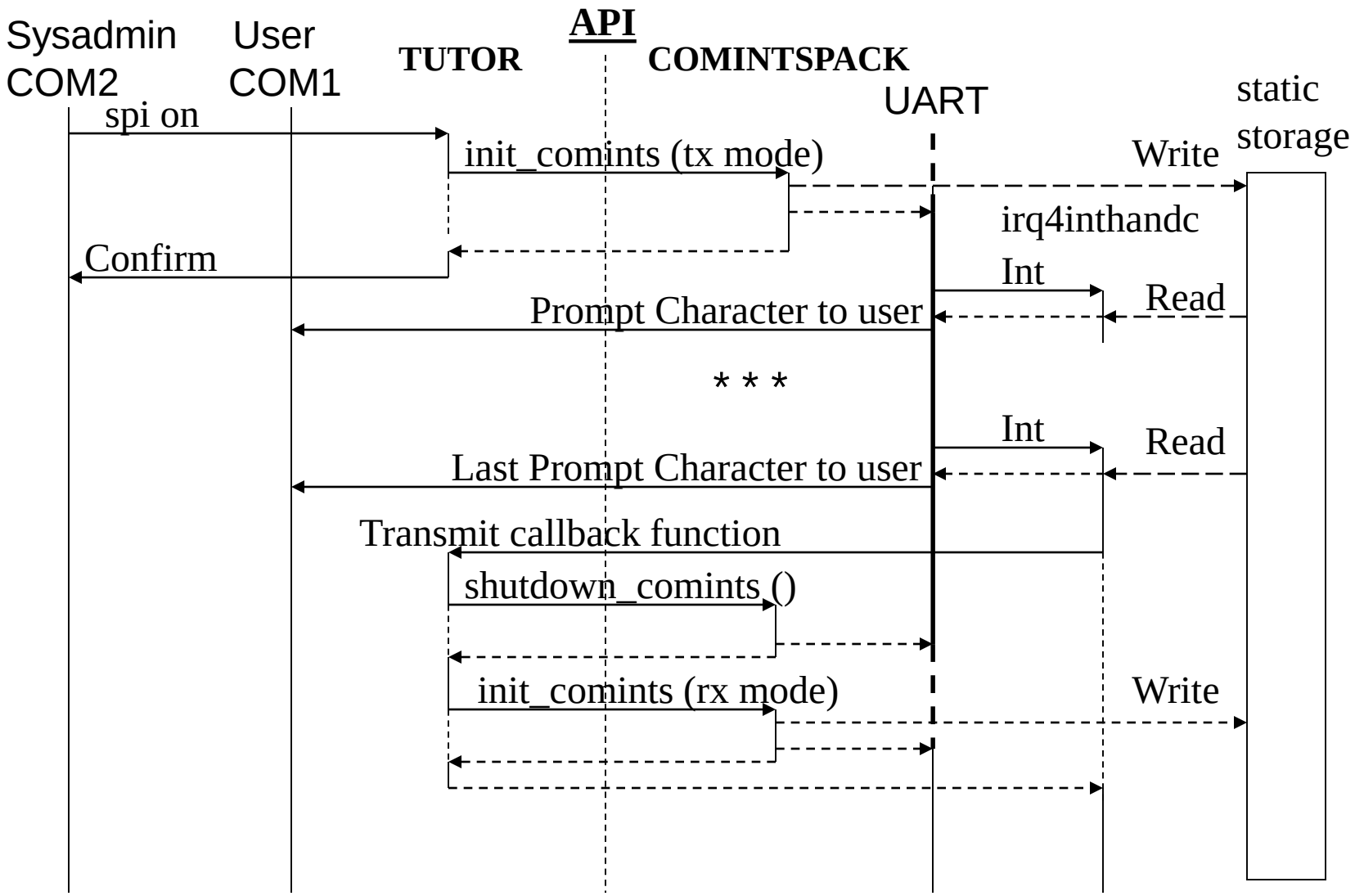
What's in comintspack.c?

- Shut down COM1 port (`shutdown_comints`)
  - Disable the PIC for the COM IRQ
  - Disable both interrupts in the UART's IER
- This function is called with interrupts disabled

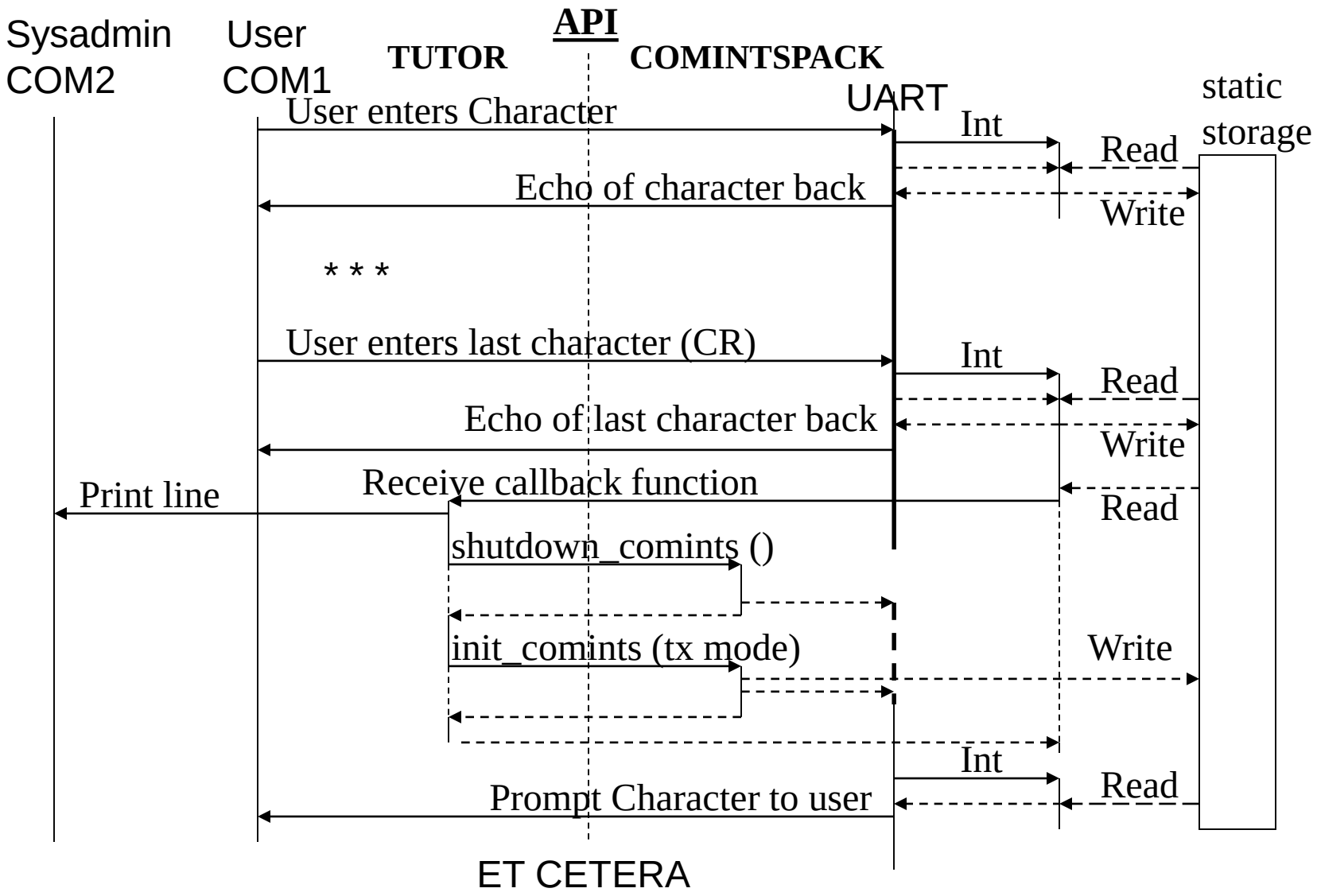
What's In comintspack.c?

- Interrupt Handler (`irq4inthandc`)
  - Acknowledge the PIC interrupt
  - For Receive
    - Input the character from COM1
    - Echo the character to COM1
    - Add to accumulated data in the application buffer
    - On end of line, call callback function passing buffer
  - For Transmit
    - Get the next outgoing character from application buffer
    - If not end of string ('\0'), output the character
    - Otherwise output CR and call callback function

Comintspack Ladder Diagram



Comintspack Ladder Diagram



UART Interrupts

- The UART is a real interrupt driven I/O device
- At system reset, all interrupt are disabled
- The UART has four conditions for interrupting
- We'll use two alternately - the receiver "data ready" and transmitter "THR empty" interrupts
- We program the UART to enable them via the COM1 Interrupt Enable Register (IER = 0x3f9)

UART Interrupts

- The UART interrupts each time it receives a char or the THR goes empty (depending on the interrupt enabled)
- COM1 is connected to pin IR4 on the PIC, its IRQ is 4.
- The nn code generated by the PIC for COM1 is 0x24, so its interrupt gate descriptor is IDT[0x24]
- ISR must send an EOI command to the PIC
- The ISR must read the received char or write the THR to cause the UART to remove its interrupt
- The UART hardware detects the inb or outb for the character and completes its interrupt-in-progress

UART Interrupts

- Two Parts of the Interrupt Handler
- `irq4inthand` – the outer assembly language interrupt handler
  - Save registers
  - Call C function irq4inthandc
  - Restore registers
  - iret
- `irq4inthandc` - the C interrupt handler
  - Does the work described earlier

Demonstration of Both Windows

COM1	COM2	
	PC-tutor> spi on	
Prompt:	comints for COM1 on	
see me type data	PC-tutor> see me type data^M^M	
Prompt:	timeon 5	I can still enter a PC-tutor cmd
	timer on	
	PC-tutor> (1)	Timer is operating independently
more data1	more data1^M^M	of the COM1 port with interrupts
Prompt:	(2)	
	(3)	
more data2	more data2^M^M	
Prompt:	timeoff	Another PC-tutor command
	timer off	
	PC-tutor> spi off	
	comints for COM1 off	
	PC-tutor> q	
	Exception 3 at EIP=00100110: Breakpoint	
	~q	
	Quit handler:	
~q	killing process 12521 Leaving board #7	
Quit handler:		
killing process 12932 Leaving board #-1		