8/25/25

Minimal booting kernel - hello world OS

userspace
supervisor (kernel)
machine

Kernel directory

- linker script : build process (pull in stuff, and make sure in right order)
- entry.s : set up system stack set up so we can do things in C
- start.c : ?
- mkfs : make userspace?

making our own kernel - copy over from xv6 picking and choosing only necessary

- makefile :
  - add and remove the files we want to compile
    modify qemu build target to remove .img
  - qemu : don't need img because not making userspace
    drop smp to 1, single core OS
    delete drive, not using hard drive
    delete line below that too (Virtual io)

- kernel :
  - copy over all header files
    - doesn't have anything that should effect the OS, only has info on files we would plan to write
  - copy over entry.S
    - .section .text tells linker this is text segment
    - global _entry tells linker where the program should start
      - csrr : control register
  - copy over start.c
    - do everything in machine mode before going to main
    - MSTATUS keeps track of the previous mode
    - mret : machine return
      - our way of switching modes (return from machine mode and go to previous mode (status register))
      - because part of boot process, doesn't have previous mode we just set it to supervisor so that it will return to supervisor for our next steps
      - Exception program counter (EPC) : remember what step we were on before entering machine mode
      - disable paging (turn off virtual memory as it will happen later in supervisor mode)

- all interrupts and exception sent to supervisor mode
- give supervisor mode access to all physical memory
- delete clocks (no timer interrupts?)
- grab hardware thread id and store it while still in machine mode
- delete timer interrupts

- copy over main.c
  - volatile static int started = 0
    - volatile: can be changed from outside the programs control
    - static: keeps it in this file
  - CPU 0 initializes a bunch of stuff
    - while this happens, all the other CPUs spin monitoring "started" waiting for it not to be 0 so they can begin their processes
  - because we're running our kernel with only 1 CPU, we don't need to do spinning and started checks
  - consoleinit()
    - set up UART
  - kinit()
    - Physical page allocator
    - finds pages that weren't used by kernel and creates a linked list of them
  - kvminit()
    - create kernel page table
    - map virtual addresses to physical addresses
  - kvminithart()
    - turn on paging
    - for specific hart
  - trapinit()
    - trap vectors
    - capture address of code that will handle blank
  - plicinit()
    - set up interrupt controller
  - while(1): spin
- copy linker
  - delete trampoline

delete

8/27/25

- uart.c -

- Reg(reg)
  - macro ; acts like a function, but defined within source code
  - grabs address of register and typecast it (volatile uchar)
- RHR & THR at 0 bytes past UART starting address
- uartinit
  - writes values to registers
  - we are going to turn off interrupts and disable spinlock
- uartputc_sync
  - write a byte to uart ; blocking run
    - if 1: okay to write ; if 0: wait for 1
  - if panicked: make it inf loop

- console.c

- consputc - looks char trying to print if backspace: replace previous char with blank ; else: print char
- consoleinit
  - remove everything except uart

- printf.c - kernel messages

- spinlock - so only 1 cpu can call a printf at a time
  - we only have 1 cpu ; delete
  - remove panics
- printfinit
  - remove

1. entry.S
   - the linker starts execution at _entry label
   - stack is created
   - jumps to start() in start.c
2. start.c (machine mode)
   - performs initial hardware setup
   - saves CPU hart ID
   - sets mepc to the address of main function
   - sets mstatus to prep for transition to supervisor mode
   - executes mret which switches the CPU to supervisor mode and jumps to main
3. main.c (supervisor mode)
   - main function execution
   - calls a series of initialization functions