Topics for today:

More on virtual memory
– Segmentation

Begin interrupts

Segmentation

Segmentation is another form of virtual memory. In segmentation, logical memory is divided among relatively large blocks called *segments*.

Segmentation has several important differences from paging.

Segmentation

- Size of segments is variable.
- Segmentation is typically visible to the programmer (i.e., *not* transparent).
- There is more than one address space.
- Segmentation makes sense even if there is fully enough space in physical memory.

Example:

The Intel 8086 (ancestor of the Pentium and Celeron) addressed memory through four basic segments:

- Code Segment (CS)
- Data Segment (DS)
- Stack Segment (SS)
- Extra Segment (ES)

Segmentation provides protection when several processes are running concurrently. Each process has its own segment(s), so different processes cannot read or over-write each other's data.

Alternatively, processes could intentionally share some segments (e.g., shared data).

The biggest problem with segmentation is (external) *fragmentation*, when holes of varying size appear between segments in physical memory. Fitting segments into the remaining holes becomes complicated, and wasted space results.

Moving segments to compact memory and connect the holes is an option but is time-consuming.

Combining Paging & Segmentation

Virtual address space is divided into segments of variable length and those segments are divided into fixed-size pages. Main memory is divided into the same size frames.

Interrupts

An *interrupt* occurs when normal execution is halted temporarily in order for the CPU to attend to some more pressing matter.

Interrupts

There are (at least) two kinds of interrupts:

- Generated by the CPU itself

  (Overflow, dividing by zero, …)

- Externally generated (usually I/O)

Interrupt process

1) Device asserts an *interrupt request* on the interrupt request control line.
2) The CPU suspends what it is doing and pushes the contents of its registers (including the IR and PC) onto a stack.
3) Based on the request, the CPU looks up the address of a block of code (called the *interrupt handler* or *interrupt service routine*) in the *interrupt vector table*.

Interrupt process

4) The CPU executes the interrupt service routine in the normal way.
5) The previous settings for registers are popped back into the CPU, and the previous process picks up again at the point where it was suspended.