# I/O Systems

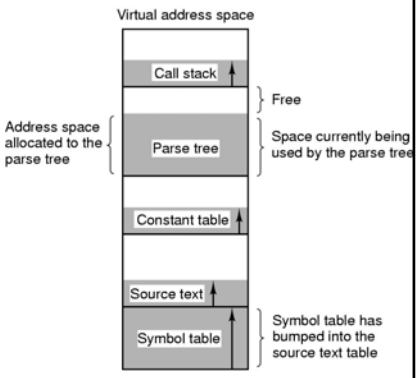
**CS 256/456**

**Dept. of Computer Science, University of Rochester**



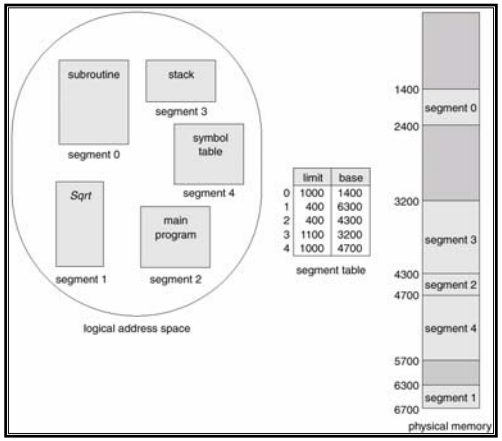
## Segmentation

- One-dimensional address space with growing pieces
- At compile time, one table may bump into another
- Segmentation:
  - generate segmented logical address at compile time
  - segmented logical address is translated into physical address at execution time





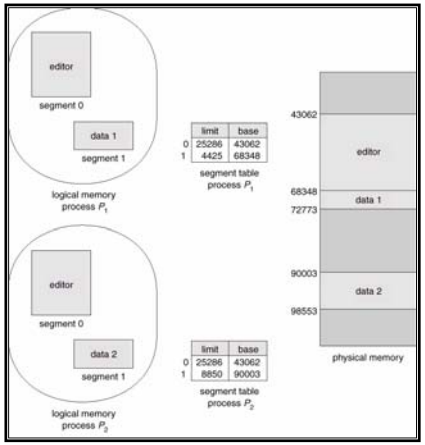

## Example of Segmentation



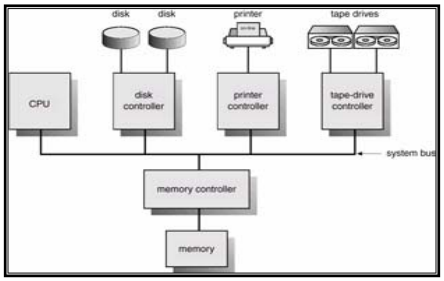



## Sharing of Segments

- Convenient sharing of libraries



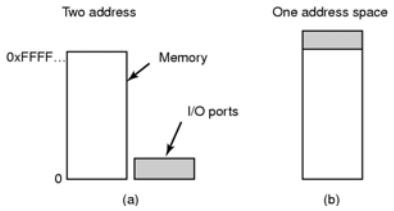
# I/O Device Controllers




- I/O devices have mechanical component & electronic component
- The electronic component is the device controller
  - It contains control logic, command registers, status registers, and on-board buffer space



# I/O Ports & Memory-Mapped I/O




I/O methods:

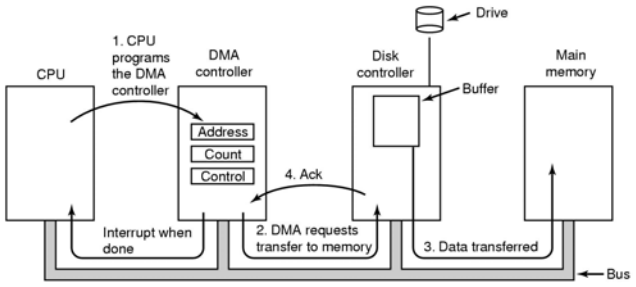
- Separate I/O and memory space; special I/O commands (IN/OUT)
- Memory-mapped I/O

Issues with them:

- Convenience/efficiency when use high-level language;
- Protection mechanisms;
- Special data access schemes: TEST
- Data caching



# Direct Memory Access (DMA)




- Are the addresses CPU sends to the DMA controller virtual or physical addresses?
- Can the disk controller directly reads data into the main memory (bypassing the controller buffer)?



# How is I/O accomplished?

- Polling-based
  - CPU spins and polls the I/O until it completes
- Periodical polling
  - Continuous polling consumes too much CPU
  - Instead, we poll periodically - saving CPU overhead, may not react immediately to hardware events
- Interrupt-driven
  - CPU initiates I/O and then does something else; gets notified when the I/O is done (interrupts)

## Interrupt Handlers

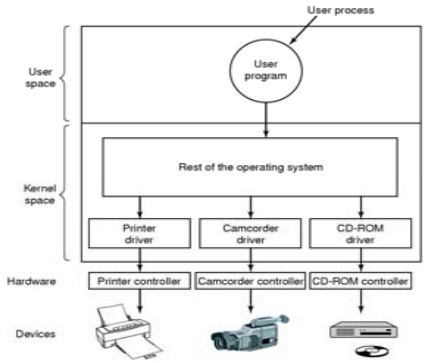
1. Save registers of the old process
2. Set up context for interrupt service procedure (switch from the user space to kernel space: MMU, stack, …)
3. Run service procedure; when safe, re-enable interrupts
4. Run scheduler to choose the new process to run next
5. Set up context (MMU, registers) for process to run next
6. Start running the new process

How much cost is it? Is it a big deal?

For Gigabit Ethernet, each packet arrives once every 12us.



## I/O Software Layers




- Device-dependent OS I/O software; directly interacts with controller hardware
- Interface to upper-layer OS code is standardized.



## Device Driver Reliability

- Device driver is the device-specific part of the kernel-space I/O software. It also includes interrupt handlers.
- Device drivers must run in kernel mode.
- ⇒ The crash of a device driver typically brings down the whole system.
- Device drivers are probably the buggiest part of the OS. Why?
- How to make the system more reliable by isolating the faults of device drivers?
  - Run most of the device driver code at user level.
  - Restrict and limit device driver operations in the kernel.



## Upper-level I/O Software

- Device independence
  - reuse software as much as possible across different types of devices
- Buffering
  - data coming off a device is stored in intermediate buffers
- Need for buffering
  - caching
  - speculative I/O

## Disclaimer

- Parts of the lecture slides contain original work of Abraham Silberschatz, Peter B. Galvin, Greg Gagne, Andrew S. Tanenbaum, and Gary Nutt. The slides are intended for the sole purpose of instruction of operating systems at the University of Rochester. All copyrighted materials belong to their original owner(s).