FAQ

• **User mode** (for user programs)
  – Some resources cannot be used directly by a user program
  – Need *system calls for IO operations*

• **Supervisor (or Kernel) mode** (privileged mode for kernel)
  – Access to all resources
  – Input/output operations are done in kernel mode, hence require system calls.

• More details soon

**Programming assignments:**

• Requirement: submissions must compile/run on machines in the [CSB-120 Linux lab](#).

• Reason: the TAs will test them on department machines. There may be environment incompatibilities.
Today

• Input/output
  – Interrupts
  – DMA
• Multiprocessor, Multiprogramming, Multitasking
• Memory
• Storage
One Processor One program View

• External devices: disk, network, screen, keyboard etc.
• Device interface: Status and data registers
• User and Supervisor modes for processor
• I/O
  – Device drivers can use polling or interrupt
  – Interrupts need *context switch*
  – I/O done in supervisor mode
  – *System calls* invoke devise drivers
What a simple view don’t include

• Cache between CPU and main memory
  – Makes the main memory appear much faster
• Direct memory access (DMA) between Main Memory and Disk (or network etc)
  – Transfer by blocks at a time
• Neglecting the fact that memory access slower than register access
• Letting program run concurrently (Multiprogramming) or with many threads
• Multiple processors in the system (like in Multicore)
Information transfer in a system

• CPU Registers – (Caches) - Memory
  – CPU addresses memory locations
  – Bytes/words at a time
  – We will see some details

• Memory – (Controllers hw/sw) - external devices
  – Chunks of data
  – External devices have their own timing
    • DMA with interrupts
  – Disk is external!
I/O Hardware (Cont.)

- I/O Devices usually have registers where device driver places commands, addresses, and data
  - Data-in register, data-out register, status register, control register
  - Typically 1-4 bytes, or FIFO buffer
- Devices have addresses, used by
  - Direct I/O instructions
  - Memory-mapped I/O
    - Device data and command registers mapped to processor address space
I/O Transfer rates MB/sec

- System bus: 100000 MB/sec
- HyperTransport (32-pair): 100000 MB/sec
- PCI Express 2.0 (×32): 100000 MB/sec
- Infiniband (QDR 12X): 100000 MB/sec
- Serial ATA (SATA-300): 10000 MB/sec
- Gigabit Ethernet: 1000 MB/sec
- SCSI bus: 100 MB/sec
- FireWire: 100 MB/sec
- Hard disk: 10 MB/sec
- Modem: 1 MB/sec
- Mouse: 1 MB/sec
- Keyboard: 1 MB/sec

Colorado State University
Polling vs Interrupt

• Polling: IO initiated by software (P&P, ch 8)
  – CPU monitors readiness
  – Keeps checking a bit to see if it is time for an IO operation,
  – not efficient

• Interrupts: IO is initiated by hardware (P&P ch 10.2)
  – CPU is informed when the external device is ready for an IO
  – CPU does something else until interrupted
Interrupts

- Polling is slow
- Interrupts used in practice
- CPU **Interrupt-request line** triggered by I/O device
  - Checked by processor after each instruction
- Interrupt handler receives interrupts
  - Maskable to ignore or delay some interrupts
- **Interrupt vector** to dispatch interrupt to correct handler
  - Context switch at start and end
  - Based on priority
  - Some nonmaskable
  - Interrupt chaining if more than one device at same interrupt number
Interrupt-Driven I/O Cycle

1. CPU
   - device driver initiates I/O
   - CPU executing checks for interrupts between instructions
   - CPU receiving interrupt, transfers control to interrupt handler
   - interrupt handler processes data, returns from interrupt
   - CPU resumes processing of interrupted task

2. I/O controller
   - initiates I/O
   - input ready, output complete, or error generates interrupt signal
Interrupts (Cont.)

- Interrupt mechanism also used for **exceptions**, which include
  - Terminate process, crash system due to hardware error
  - Page fault executes when memory access error
  - OS causes switch to another process
  - System call executes via **trap** to trigger kernel to execute request
Direct Memory Access

• for movement of a block of data
  – To/from disk, network etc.
• Requires **DMA** controller
• Bypasses CPU to transfer data directly between I/O device and memory
• OS writes DMA command block into memory
  – Source and destination addresses
  – Read or write mode
  – Count of bytes
  – Writes location of command block to DMA controller
  – Bus mastering of DMA controller – grabs bus from CPU
    • Or **Cycle stealing** from CPU but still much more efficient
  – When done, interrupts to signal completion
Six Step Process to Perform DMA Transfer

1. Device driver is told to transfer disk data to buffer at address X
2. Device driver tells disk controller to transfer C bytes from disk to buffer at address X
3. Disk controller initiates DMA transfer
4. Disk controller sends each byte to DMA controller
5. DMA controller transfers bytes to buffer X, increasing memory address and decreasing C until C = 0
6. When C = 0, DMA interrupts CPU to signal transfer completion

Interrupt when done

Device driver: code
Device controller: hw
Direct Memory Access Structure

- high-speed I/O devices
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention
- Only one interrupt is generated per block
I/O Subsystem

• One purpose of OS is to hide peculiarities of hardware devices from the user

• I/O subsystem responsible for
  – Memory management of I/O including
    • buffering (storing data temporarily while it is being transferred),
    • caching (storing parts of data in faster storage for performance),
    • spooling (the overlapping of output of one job with input of other jobs) like printer queue
  – General device-driver interface
  – Drivers for specific hardware devices
Application I/O Interface

- I/O system calls encapsulate device behaviors in generic classes
- Device-driver layer hides differences among I/O controllers from kernel
- New devices talking already-implemented protocols need no extra work
- Each OS has its own I/O subsystem structures and device driver frameworks

- Devices vary in many dimensions
  - Character-stream or block
  - Sequential or random-access
  - Synchronous or asynchronous (or both)
  - Sharable or dedicated
  - Speed of operation
  - read-write, read only, or write only
# A Kernel I/O Structure

![Diagram of a kernel I/O structure]

## Software

<table>
<thead>
<tr>
<th>Software</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kernel I/O subsystem</td>
<td></td>
</tr>
<tr>
<td>SCSI device driver</td>
<td></td>
</tr>
<tr>
<td>keyboard device driver</td>
<td></td>
</tr>
<tr>
<td>mouse device driver</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>PCI bus device driver</td>
<td></td>
</tr>
<tr>
<td>floppy device driver</td>
<td></td>
</tr>
<tr>
<td>ATAPI device driver</td>
<td></td>
</tr>
</tbody>
</table>

## Hardware

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCSI device controller</td>
<td></td>
</tr>
<tr>
<td>keyboard device controller</td>
<td></td>
</tr>
<tr>
<td>mouse device controller</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>PCI bus controller</td>
<td></td>
</tr>
<tr>
<td>floppy device controller</td>
<td></td>
</tr>
<tr>
<td>ATAPI device controller</td>
<td></td>
</tr>
</tbody>
</table>

- SCSI devices
- Keyboard
- Mouse
- PCI bus
- Floppy-disk drives
- ATAPI devices (disks, tapes, drives)
Storage
Storage Structure

• Main memory – only large storage media that the CPU can access directly
  – Random access
  – Typically volatile (except for ROM)

• Secondary storage – extension of main memory that provides large nonvolatile storage capacity
  – Hard disks (HDD) – rigid platters covered with magnetic recording material
    • Disk surface divided into tracks, which are subdivided into sectors
    • The disk controller – transfers between the device and the processor
  – Solid-state disks (SSD) – faster than hard disks, lower power consumption
    • More expensive, but becoming more popular

• Tertiary/removable storage
  – External disk, thumb drives, cloud backup etc.
Storage Hierarchy

- Storage systems organized in hierarchy
  - Speed
  - Cost
  - Volatility

- **Caching** – copying information into faster storage system; main memory can be viewed as a cache for secondary storage

- **Device Driver** for each device controller to manage I/O
  - Provides uniform interface between controller and kernel
One or the other

Storage-Device Hierarchy

- registers
- cache
- main memory
- solid-state disk
- hard disk
- optical disk
- magnetic tapes
Performance of Various Levels of Storage

<table>
<thead>
<tr>
<th>Level</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>registers</td>
<td>cache</td>
<td>main memory</td>
<td>solid state disk</td>
<td>magnetic disk</td>
</tr>
<tr>
<td>Typical size</td>
<td>&lt; 1 KB</td>
<td>&lt; 16MB</td>
<td>&lt; 64GB</td>
<td>&lt; 1 TB</td>
<td>&lt; 10 TB</td>
</tr>
<tr>
<td>Implementation technology</td>
<td>custom memory with multiple ports CMOS</td>
<td>on-chip or off-chip CMOS SRAM</td>
<td>CMOS SRAM</td>
<td>flash memory</td>
<td>magnetic disk</td>
</tr>
<tr>
<td>Access time (ns)</td>
<td>0.25 - 0.5</td>
<td>0.5 - 25</td>
<td>80 - 250</td>
<td>25,000 - 50,000</td>
<td>5,000,000</td>
</tr>
<tr>
<td>Bandwidth (MB/sec)</td>
<td>20,000 - 100,000</td>
<td>5,000 - 10,000</td>
<td>1,000 - 5,000</td>
<td>500</td>
<td>20 - 150</td>
</tr>
<tr>
<td>Managed by</td>
<td>compiler</td>
<td>hardware</td>
<td>operating system</td>
<td>operating system</td>
<td>operating system</td>
</tr>
<tr>
<td>Backed by</td>
<td>cache</td>
<td>main memory</td>
<td>disk</td>
<td>disk</td>
<td>disk or tape</td>
</tr>
</tbody>
</table>

Movement between levels of storage hierarchy can be explicit or implicit

- Cache managed by hardware. Makes main memory appear much faster.
- Disks are several orders of magnitude slower.
Multilevel Caches

- **Cache**: between registers and main memory
  - Cache is faster and smaller than main memory
  - Makes main memory appear to be much faster, if the stuff is found in the cache much of the time
  - Hardware managed because of speed requirements

- **Multilevel caches**
  - L1: smallest and fastest of the three (about 4 cycles)
  - L2: bigger and slower than L1 (about 10 cycles)
  - L3: bigger and slower than L2 (about 50 cycles)
  - Main memory: bigger and slower than L3 (about 150 cycles)

- You can mathematically show that multi-level caches improve performance with usual high hit rates.
Concept: Caching

• Important principle, performed at many levels in a computer (in hardware, operating system, software)
• Information in use copied from slower to faster storage temporarily
• Faster storage (cache) checked first to determine if information is there
  – If it is, information used directly from the cache (fast)
  – If not, data copied to cache and used there
• Cache smaller than storage being cached
  – Cache management important design problem
  – Cache size and replacement policy
• Examples: “cache”, browser cache ..
Multiprocessors

• Past systems used a single general-purpose processor
  – Most systems have special-purpose processors as well
• **Multiprocessors** systems were once special, now are common
  – Advantages include:
    1. Increased throughput
    2. Economy of scale
    3. Increased reliability – graceful degradation or fault tolerance
  – Two types:
    1. **Asymmetric Multiprocessing** – each processor is assigned a specific task.
    2. **Symmetric Multiprocessing** – each processor performs all tasks
Symmetric Multiprocessing Architecture

CPU₀
- registers
- cache

CPU₁
- registers
- cache

CPU₂
- registers
- cache

memory
Multiprocessor

Multi-chip and multicore

• Multi-chip: Systems containing all chips
  – Chassis containing multiple separate systems

• Multi-core
Multiprogramming and multitasking

• **Multiprogramming** needed for efficiency
  – Single user cannot keep CPU and I/O devices busy at all times
  – Multiprogramming organizes jobs (code and data) so CPU always has one to execute
  – A subset of total jobs in system is kept in memory
  – One job selected and run via **job scheduling**
  – When it has to wait (for I/O for example), OS switches to another job

• **Timesharing (multitasking)** is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing
  – **Response time** should be < 1 second
  – Each user has at least one program executing in memory ⇒ **process**
  – If several jobs ready to run at the same time ⇒ **CPU scheduling**
  – If processes don’t fit in memory, **swapping** moves them in and out to run
  – **Virtual memory** allows execution of processes not completely in memory
Memory Layout for Multiprogrammed System

![Diagram of memory layout showing operating system, job 1, job 2, job 3, and job 4]
Operating-System Operations

- **Interrupt driven** (hardware and software)
  - Hardware interrupt by one of the devices
  - Software interrupt (**exception** or **trap**):
    - Software error (e.g., division by zero)
    - Request for operating system service
    - Other process problems include infinite loop, processes modifying each other or the operating system
Dual-mode operation allows OS to protect itself and other system components:

- **User mode** and **kernel mode**
- **Mode bit** provided by hardware
  - Provides ability to distinguish when system is running user code or kernel code
  - Some instructions designated as **privileged**, only executable in kernel mode
  - System call changes mode to kernel, return from call resets it to user

Increasingly CPUs support multi-mode operations:
- i.e. **virtual machine manager (VMM)** mode for guest **VMs**
Transition from User to Kernel Mode

- Timer to prevent infinite loop / process hogging resources
  - Timer is set to interrupt the computer after some time period
  - Keep a counter that is decremented by the physical clock.
  - Operating system set the counter (privileged instruction)
  - When counter zero generate an interrupt
  - Set up before scheduling process to regain control or terminate program that exceeds allotted time
A process is a program in execution. It is a unit of work within the system. Program is a *passive entity*, process is an *active entity*.

- Process needs resources to accomplish its task
  - CPU, memory, I/O, files
  - Initialization data
- Process termination requires reclaim of any reusable resources
- **Single-threaded process** has one *program counter* specifying location of next instruction to execute
  - Process executes instructions sequentially, one at a time, until completion
- **Multi-threaded process** has one program counter per thread
- Typically system has many processes, some user, some operating system running concurrently on one or more CPUs
  - Concurrency by multiplexing the CPUs among the processes / threads
Process Management Activities

The operating system is responsible for the following activities in connection with process management:

- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication
- Providing mechanisms for deadlock handling
Byte (B) = 8 bits (b)

- A kilobyte, or KB, is $1,024$ (or $2^{10}$) bytes
- A megabyte, or MB, is $1,024^2$ (or $2^{20}$) bytes
- A gigabyte, or GB, is $1,024^3$ bytes
- A terabyte, or TB, is $1,024^4$ bytes
- A petabyte, or PB, is $1,024^5$ bytes
Memory Management

- To execute a program all (or part) of the instructions must be in memory.
- All (or part) of the data that is needed by the program must be in memory.
- Memory management determines what is in memory and when
  - Optimizing CPU utilization and computer response to users
- Memory management activities
  - Keeping track of which parts of memory are currently being used and by whom
  - Deciding which processes (or parts thereof) and data to move into and out of memory
  - Allocating and deallocating memory space as needed
Storage Management

- OS provides uniform, logical view of information storage
  - Abstracts physical properties to logical storage unit - file
  - Each medium is controlled by device (i.e., disk drive, tape drive)
    - Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)

- File-System management
  - Files usually organized into directories
  - Access control on most systems to determine who can access what
  - OS activities include
    - Creating and deleting files and directories
    - Primitives to manipulate files and directories
    - Mapping files onto secondary storage
    - Backup files onto stable (non-volatile) storage media
Mass-Storage Management

• Usually disks used to store data that does not fit in main memory or data that must be kept for a “long” period of time
• Entire speed of computer operation hinges on disk subsystem and its algorithms
• OS activities
  – Free-space management
  – Storage allocation
  – Disk scheduling
• Some storage need not be fast
  – Tertiary storage includes optical storage, magnetic tape
  – Still must be managed – by OS or applications
  – Varies between WORM (write-once, read-many-times) and RW (read-write)
### Performance of Various Levels of Storage

<table>
<thead>
<tr>
<th>Level</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>registers</td>
<td>cache</td>
<td>main memory</td>
<td>solid state disk</td>
<td>magnetic disk</td>
</tr>
<tr>
<td>Typical size</td>
<td>&lt; 1 KB</td>
<td>&lt; 16MB</td>
<td>&lt; 64GB</td>
<td>&lt; 1 TB</td>
<td>&lt; 10 TB</td>
</tr>
<tr>
<td>Implementation technology</td>
<td>custom memory with multiple ports CMOS</td>
<td>on-chip or off-chip CMOS SRAM</td>
<td>CMOS SRAM</td>
<td>flash memory</td>
<td>magnetic disk</td>
</tr>
<tr>
<td>Access time (ns)</td>
<td>0.25 - 0.5</td>
<td>0.5 - 25</td>
<td>80 - 250</td>
<td>25,000 - 50,000</td>
<td>5,000,000</td>
</tr>
<tr>
<td>Bandwidth (MB/sec)</td>
<td>20,000 - 100,000</td>
<td>5,000 - 10,000</td>
<td>1,000 - 5,000</td>
<td>500</td>
<td>20 - 150</td>
</tr>
<tr>
<td>Managed by</td>
<td>compiler</td>
<td>hardware</td>
<td>operating system</td>
<td>operating system</td>
<td>operating system</td>
</tr>
<tr>
<td>Backed by</td>
<td>cache</td>
<td>main memory</td>
<td>disk</td>
<td>disk</td>
<td>disk or tape</td>
</tr>
</tbody>
</table>

Movement between levels of storage hierarchy can be explicit or implicit.
• Multitasking environments must be careful to use most recent value, no matter where it is stored in the storage hierarchy

![Diagram showing migration of data "A" from disk to register.]

• Multiprocessor environment must provide **cache coherency** in hardware such that all CPUs have the most recent value in their cache

• Distributed environment situation even more complex
  – Several copies of a datum can exist
  – Various solutions covered in Chapter 17 (*will not get to it*)