Architecture of I/O Systems

- The device itself
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
- The device is a part of a computer system that uses data to communicate with the CPU
What happens if the device is slow compared to the CPU?
- Good choice if data must be handled promptly, like for a modem or keyboard.
- CPU observes the change to idle and reads the data if it was an input operation.
- If the operation succeeds, the controller changes the status to idle.
- Data-in is an input command.
- Controller reads the command register and performs the command, placing a value in
- CPU sets status to command-ready = controller sets status to busy
- CPU sets the command register and delays if it is an output operation.
- CPU busy-waits until the status is idle.

### Communication using Polling

### I/O Services Provided by OS

- Device drivers to implement device-specific behaviors.
  - Example:
  - Error handling and failure recovery associated with devices (command routines, for
  - I/O scheduling.
  - Buffering, caching, and spooling to allow efficient communication with devices.
  - Device allocation.
  - Operations appropriate to the files and devices.
  - Access control.
  - Naming of files and devices. (On Unix, devices appear as files in the /dev directory)
Transfer I/O

CPU somewhat, but still providing better performance than if the CPU had to do the

- The DMA controller and the CPU compete for the memory bus, slowing down the

  transfer is complete, instead of when each byte is ready.

- The DMA controller operates the bus and interrupts the CPU when the entire

  DMA is complete.

- The DMA reads the DMA's locations of the source and destination of the transfer.

- DMA is efficient, it has an address register.

- Use a sophisticated DMA controller that can write directly to memory. Instead of

  DMA

Solution: Direct memory access (DMA)

For devices that transfer large volumes of data at a time, (like a disk)

Direct Memory Access

Interrupts

- Start the next operation for that device.

- If the last command was an input operation, retrieve the data from the device register.

- Determine which device caused the interrupt.

  On an I/O Interrupt:

  - Complete an I/O operation.

  - Rather than being busy waiting, the device can interrupt the CPU when it

Communication using Interrupts
The DMA controller interrupts the CPU when the transfer is done.

- Physical memory
- It is transferred over the bus by the DMA controller into a buffer in
- A disk buffer stores a block when it is read from the disk.

I/O devices typically contain a small on-board memory where they can store

I/O Buffering

Device Characteristics:
- Examples: Keyboard (sequential, character), disk (block, random or sequential)
- Operations: Input, output, or both
- Synchronous or asynchronous
- Transfer unit: character or block
- Most devices are asynchronous, while I/O system calls are synchronous
- Timing: Synchronous or asynchronous
- Access methods: sequential or random access
- OI/impl: non-blocking or blocking

New devices can be supported by providing a new device driver.
Device dependencies are encapsulated in device drivers.
Standard interfaces are provided for related devices.

The OS provides a high-level interface to devices, greatly simplifying the

Application Programmer's View of I/O Devices
lower memory and disk (sometimes later). Faster.
write-back policy (write only to the fastest memory containing the block, write to disk). Higher reliability.
write-through policy (write to all levels of memory containing the block, including to disk).

What should happen when we write to a cache?

- Example: Write (disk, address)
  - If (block in memory) update block in memory
  - Else read cache (disk, address)
  - If (block in memory) return value from memory
  - Else read from disk (disk, address)

- Example: Read (disk, address)
  - If (block in memory) return value from memory
  - Else read cache (disk, address)
  - Else read from disk (disk, address)

- Idea: Keep recently used disk blocks in main memory after the I/O call that brought them into memory completes.

- Improve disk performance by reducing the number of disk accesses.

**Caching**

Why buffer on the OS side?

- Write from the kernel buffer to the disk is done later.
  - Write() copies data to a kernel buffer and returns control to the user program. The
    - To minimize the time a user process is blocked on a write:
      -屏障 one block at a time.
        - Example: Read the file from the network one packet at a time. Stores to disk
          - To cope with devices that have different data transfer sizes.
            - Example: Compute the contents of a display in a buffer (slow) and then zap the
              -屏障 to the screen (fast)

- Example: Compute the contents of a display in a buffer (slow) and then zap the
  - To cope with speed mismatches between device and CPU.
Utilization:
- Increase physical memory to reduce amount of time paging and thereby improve CPU utilization.
- Improve CPU utilization.
- Increase the number of devices to reduce contention for a single device and thereby:
  - Offload computation from the main CPU by using DMA controllers.
  - Reduce interrupt frequency by using large data transfers.
  - Reduce data copying by caching in memory.

Approaches to improving performance:
- I/O is typically supported via system calls and interrupt handling, which are slow.
- Conversion from multiple processes.
- Slow devices and slow communication links.

I/O is expensive for several reasons:

**Summary**

**Call:**

1. User process requests a read from a device.
2. OS checks if data is in a buffer. If not,
   - OS transfers the data to the user process and places the process in the ready queue.
3. OS transfers the data to the user process and places the process in the ready queue.
4. When the process gets the CPU, it begins execution following the system call.

**Dma:**

- DMA controller interrupts the CPU when the transfer is complete.
- From the device.
- DMA controller transfers the data to the kernel buffer when it has all been retrieved.
- Device driver tells the DMA controller what to do and blocks input.

Putting the Pieces Together - a Typical Read Call:
What needs to happen on a context switch to support memory management:

- Memory protection
- Ability to move processes
- Ability to share memory with other processes
- Ability to grow processes
- Coping with fragmentation
- Hardware support required
- Address translation

For each strategy, understand these concepts:

Memory Management (cont.)

Exam Review
What does the OS do when a page fault occurs?

What is a page fault? How does the OS know it needs to take one, and what is demand paging?

What is a TLB? How is one used?

What does the OS store in the page table?

What is paging? a page, a page frame?

Topics you should understand:

Paging

Corresponding physical address:

Given a virtual address and the necessary tables, determine the

using contiguous allocation:

Given a request for memory, determine how the request can be satisfied

Things you should be able to do:

Memory Management (cont.)
Page faults. Determine how the different replacement algorithms would handle the

I. Given a page reference string and a fixed number of page frames, what should you do?

Things you should be able to do:

- What considerations influence the page size that should be used?
- What is thrashing and what are strategies to eliminate it?

Page (cont.)

What is a working set?

What impact on the performance of paging?

What is temporal locality? What is spatial locality? What effect do these have on the performance of paging?

How do global and per-process (aka a local) allocation differ?

- Enhanced second chance
- Second chance
- LRU
- FIFO
- MIN

Page replacement algorithms. For each understand how they work.

Page (cont.)
Topics you should understand:

**I/O Systems**

- Caching and buffering
- Polling and interrupts
- Direct Memory Access

**File Systems**

1. What is a file, a file type?
2. What types of access are typical for files?
3. What does the OS do on a file open, file close?
4. What is a directory?
5. What is a link?
6. What happens if the directory structure is a graph?
7. How does an OS support multiple users of shared files?
8. Strategies for laying files out on disk: Advantages and disadvantages.

(c) Indexed
(b) Linked
(e) Continuous allocation
General Skills

- You should have a good sense of how the pieces fit together and how changes in one part of the OS might impact another.
- You will not be asked to read or write C++ code.
- You will not be asked detailed questions about any specific operating system, such as Unix, Nachos, Windows NT, ...