COSC 3360/6310
Monday, January 25

Announcements

- The first assignment is online
  - Folder ***First assignment*** of ***Assignments Channel*** on ***Teams***
  - Sub-folder ***Files*** of ***Resources*** folder on ***Prulu***

- ***Explanations*** will come on Wednesday

- Watch for ***sample inputs*** and ***outputs*** all by next Monday

The four missions (continued)

Functions of an OS

- ***Four*** basic functions
  - To provide a better user interface
  - To manage the system resources
  - To protect users’ programs and data
  - To let programs exchange information

Managing system resources

- ***Focus of the remainder of the course***
- ***Not an easy task***
  - Enormous gap between CPU speeds and disk access times

The memory hierarchy (I)

Level	Device	Access Time
1	Fastest registers (2 GHz)	**0.5 ns**
2	Main memory	**10-70 ns**
3	Secondary storage (flash)	**35-100 µs**
4	Secondary storage (disk)	**3-12 ms**
5	Mass storage (off line)	**a few s**

The memory hierarchy (II)

- To make sense of these numbers, let us consider an analogy

Writing a paper (I)

Level	Resource	Access Time
1	Open book on desk	**1 s**
2	Book on desk	
3	Book in UH library	
4	Book in another library	
5	Book very far away	

Writing a paper (II)

Level	Resource	Access Time
1	Open book on desk	**1s**
2	Book on desk	**20-140 s**
3	Book in UH library	
4	Book in another library	
5	Book very far away	

Writing a paper (II)

Level	Resource	Access Time
1	Open book on desk	**1s**
2	Book on desk	**20-140s**
3	Book in UH library	**20-55h**
4	Book in another library	
5	Book very far away	

Writing a paper (III)

Level	Resource	Access Time
1	Open book on desk	**1 s**
2	Book on desk	**20-140 s**
3	Book in UH library	**20-55 h**
4	Book in another library	**70-277 days**
5	Book very far away	

Writing a paper (V)

Level	Resource	Access Time
1	Open book on desk	**1 s**
2	Book on desk	**20s-140 s**
3	Book in UH library	**20-55 h**
4	Book in another library	**70-277 days**
5	Book very far away	**> 63 years**

Will the problem go away?

- New storage technologies
  - Cheaper than main memory
  - Faster than disk drives
- Flash drives
- Optane memory

Flash drives

- Offspring of EEPROM memories
- Fast reads
  - Block-level
- Slower writes
  - Whole page of data must be erased then rewritten
- Can only go through a finite number of program /erase cycles

Optane memory (I)

- Byte-addressable non-volatile memory (BNVM)
- Simpler design
  - Bits are stored as resistivity levels of a secret alloy
  - No transistors (≠ SRAM and DRAM)
- Faster than flash
  - 100-300 ns

Optane memory (II)

- Now
  - Non-volatile RAM
  - Disk cache
- In a few years
  - Could replace flash (phones, laptops, …)
  - Flash could replace disks (disk farms)
  - Disks could replace slower devices
  - Will require a ***redesign*** of file system

Optimizing disk accesses

- Two main techniques
  - Making disk accesses more efficient
  - Doing something else while waiting for an I/O operation

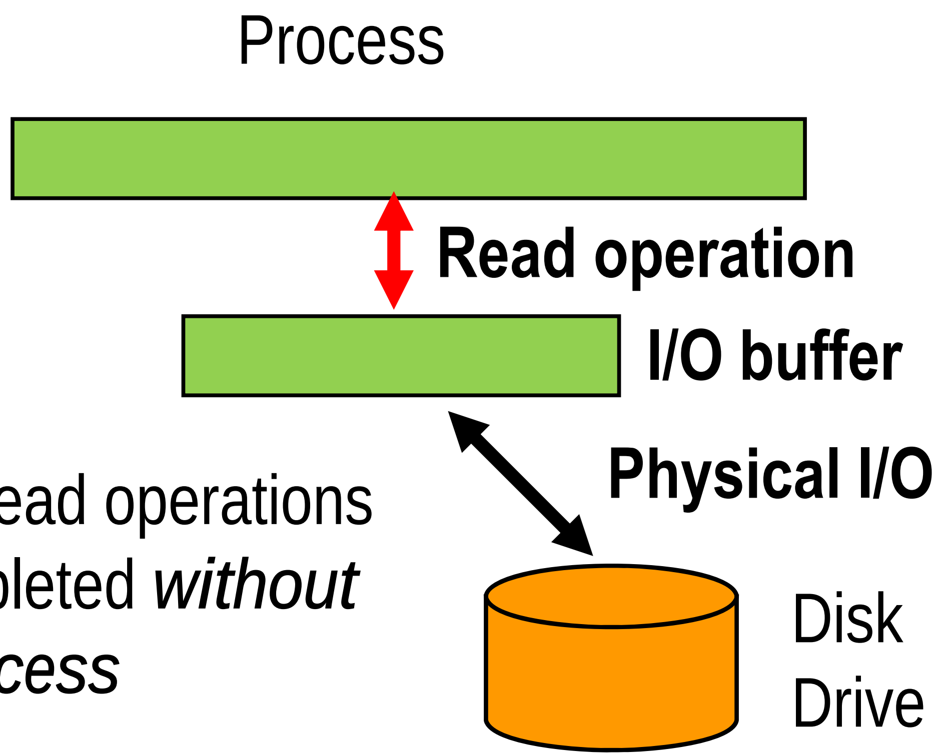
- *Not very different from what we are doing in our every day's lives*

Optimizing read accesses (I)

- *When we shop in a market that's far away from our home, we plan ahead and buy food for several days*

- The OS will read as many bytes as it can during each disk access
  - In practice, entire blocks (4KB or more)
  - Blocks are stored in the I/O buffer

Optimizing read accesses (II)



- Most small read operations can be completed *without any disk access*

Optimizing read accesses (III)

- Buffered reads work quite well
  - Most systems use it
- Major limitation
  - Cannot read ***too much ahead*** of the program
    - Could end bringing into main memory data that would ***never be used***

Optimizing read accesses (IV)

- Can ***also keep*** in a buffer recently accessed blocks hoping they will be accessed again
  - ***Caching***
- Works very well because we keep accessing again and again the data we are working with
- ***Caching is a fundamental technique of OS and database design***

Optimizing write accesses (I)

- *If we live far away from a library, we wait until we have several books to return before making the trip*

- The OS will ***delay writes*** for a few seconds then write an entire block
  - Since most writes are sequential, most small writes will not require any disk access

Optimizing write accesses (II)

- ***Delayed writes*** work quite well
  - Most systems use it
- ***Major drawback***
  - We will ***lose data*** if the system or the program crashes
    - After the program issued a write but
    - Before the data were saved to disk
  - Unless we use NVRAM

Doing something else

- *When we order something on the web, we do not remain idle until the goods are delivered*
- The OS can implement ***multiprogramming*** and let the CPU run another program while a program waits for an I/O

Advantages (I)

- Multiprogramming is very important in business applications
  - Many of these applications use the peripherals much more than the CPU
  - For a long time the CPU was the most expensive component of a computer
  - ***Multiprogramming*** was invented to keep the CPU busy

Advantages (II)

- Multiprogramming made ***time-sharing*** possible
- Multiprogramming lets your PC run several applications at the same time
  - MS Word and MS Outlook

Multiprogramming (I)

- Multiprogramming lets the CPU divide its time among different tasks:
  - One tenth of a second on a program, then another tenth of a second on another one and so forth
- Each core of your CPU will still be working on ***one single task*** at any given time

Multiprogramming (II)

- The CPU does not waste any time waiting for the completion of I/O operations
- From time to time, the OS will need to regain control of the CPU
  - Because a task has exhausted its fair share of the CPU time
  - Because something else needs to be done.
- This is done through ***interrupts*****.**

Interrupts (I)

- Request to interrupt the flow of execution the CPU
- Detected by the CPU hardware
  - ***After*** it has executed the current instruction
  - ***Before*** it starts the next instruction.

A very schematic view (I)

- A very basic CPU would execute the following loop:

```
forever {
    fetch_instruction();
    decode_instruction();
    execute_instruction();
}
```

- Pipelining makes things more complicated
  - And CPU much faster!

A very schematic view (II)

- We add an extra step:

```
forever {
    check_for_interrupts();
    fetch_instruction();
    decode_instruction();
    execute_instruction();
}
```

Interrupts (II)

- When an interrupt occurs:

  a. The ***current state of the CPU*** (program counter, program status word, contents of registers, and so forth) is saved, normally on the top of a stack

  b. A ***new CPU state*** is fetched

Interrupts (III)

- New state includes a new ***hardware-defined*** value for the program counter
  - Cannot “hijack” interrupts
- Process is totally transparent to the task being interrupted
  - A process ***never*** knows whether it has been interrupted or not

Types of interrupts (I)

- ***I/O completion interrupts***
  - Notify the OS that an I/O operation has completed,
- ***Timer interrupts***
  - Notify the OS that a task has exceeded its quantum of core time

Types of interrupts (II)

- ***Traps***
  - Notify the OS of a ***program error*** (division by zero, illegal op code, illegal operand address, ...) or a *hardware failure*
- ***System calls***
  - Notify OS that the running task wants to submit a request to the OS

A surprising discovery

- ***Programs do interrupt themselves!***

Context switches

- Each interrupt will result into ***two context switches***:
  - One when the running task is interrupted
  - Another when it regains the CPU
- Context switches are ***not cheap***
- The overhead of any simple system call is ***two context switches***

Remember that!

Prioritizing interrupts (I)

- Interrupt requests may occur while the system is processing another interrupt
- All interrupts are not equally urgent (as it is also in real life
  - Some are more urgent than other
  - *Also true in real life*

Prioritizing interrupts (II)

- The best solution is to *prioritize* interrupts and assign to each source of interrupts a ***priority level***
  - New interrupt requests will be allowed to interrupt lower-priority interrupts but will have to wait for the completion of all other interrupts
- Solution is known as ***vectorized interrupts***.

Example from real life

- Let us try to prioritize
  - Phone is ringing
  - Washer signals end of cycle
  - Dark smoke is coming out of the kitchen
  - …
- With vectorized interrupts, a phone call will never interrupt another phone call

The solution

Smoke in the kitchen
Phone is ringing
End of washer cycle
More low-priority stuff

Disabling Interrupts

- We can ***disable*** interrupts
- OS does it before performing short critical tasks that cannot be interrupted
  - Works only for single-threaded kernels
- User tasks ***must*** be prevented from doing it
  - Too dangerous

DMA

- Disk I/O poses a special problem
  - CPU will have to transfer large quantities of data between the disk controller's buffer and the main memory
- ***Direct memory access*** (***DMA***) allows the disk controller to read data from and write data to main memory without any CPU intervention
  - Controller “steals” memory cycles from CPU