OS notes

History of computers

TL;DR

Computer Architecture

The Von Neumann Architecture

Levels of programming languages

1. Executable File ("Machine code")
2. Object File linked with other Object Files ("Libraries") into 1.
3. ASM Source which is assembled into 2.
4. C/C++ Source code which is compiled into 3.
5. ML/Java Byte-code which is interpreted

This is analogous to operation of the computer.

Layered Virtual Machines

Think of a virtual machine in each layer built on the lower VM; machine in one level understands language of that level

0. Digital Logic Level
1. **Conventional Machine Level**
2. **Operating System Level**
3. Assembly Language Level
4. Compiled Language Level
5. Meta-Language Level

Registers

- Very fast on-chip memory
- Typically 32 or 64 bits
- 8 to 128 registers is usual
- Data is loaded onto registers before being operated on
- Registers may not be visible to the programmer
- Most processors have *data* and *control* (special meaning to CPU) registers

Memory Hierarchy

1. CPU
2. Cache
    i. fast, expensive
    ii. several levels of cache
3. Main Memory
4. DISK I/O
    i. I/O devices usually connected via a bus
    ii. very slow and cheap

Fetch-Execute Cycle

PC initialised to fixed value on CPU reset. Then repeat until halt:

1. instruction *fetched* from memory address in PC into instruction buffer
2. Control Unit *decodes* the instruction
3. Execution Unit *executes* the instruction
4. PC is updated either explicitly by a jump or implicitly

Buses

A bus is a group of 'wires' shared by several devices. Buses are cheap and versatile but can become a bottleneck on performance. A bus typically has:

- address lines
- data lines
- control lines

A bus is operated in a master-slave protocol: e.g. to read data from memory, CPU puts address on a bus and asserts 'read'; memory retrieves data, puts data on bus; PC reads from bus. In some cases an initialisation protocol is needed to decide which device is the bus master.

Bus Hierarchy

Interrupts

There are devices much slower than the CPU. We can't have CPU wait for these devices. Also, external events may occur.

Interrupts provide a suitable mechanism. Interrupt is a signal line into CPU. When asserted, CPU jumps to a particular location (e.g. on x86 on interrupt the CPU jumps to address stored in relevant entry of table pointed to by IDTR control register). The jump saves state; when the interrupt handler finishes, it uses a special return instruction to restore control to original program.

Direct Memory Access

DMA means allowing devices to write directly (via a bus) into main memory, e.g. CPU tells device 'write next block of data into address x' and gets an interrupt when done.

PCs have a basic DMA; IBM main-frames have I/O channels which are a sophisticated extension to DMA.

What is an Operating System for?

- handles relations between CPU, memory and devices
- handles allocation of memory
- handles sharing of memory and CPU between different logical tasks
- handles file management
- (in Windows) handles the UI graphics

Kernel - single (logical) program that is loaded at boot time and has primary control of the computer

Early batch systems

In the beginning, OS simply transferred programs from punch cards into memory. Operator had to set up entire job, programmatically.

Monitor is a simple resident OS that reads jobs, transfers control to programs, receives control. Monitor is permanently resident, programs must be loaded into a different area of the memory.

Batches of jobs can be put onto one tape and read in turn by the monitor - reduces human intervention.

Protecting the monitor from the users, which should not be able to:

- **memory protection**: write to monitor memory
- **timer control**: run forever
- **privileged instructions**: directly access I/O or certain other machine functions
- **interrupts**: delay the monitor's response to external events

Multiprogramming

Jobs would waste 75% CPU cycles waiting on I/O. *Multiprogramming* was introduced to tackle this. Monitor loaded several user programs when one is waiting for I/O, run another.

Multiprogramming, means the monitor must:

- manage memory among the various tasks
- schedule execution of the tasks

Time-sharing

Allow interactive access to computer with many users sharing. Early system gave each user 0.2s of CPU time; then save state and load state of next scheduled user.

Virtual Memory

Multitasking and time sharing are much easier if all tasks are resident rather than being swapped in and out of memory.

Virtual memory - decouples memory as seen by the user task from physical memory. Task sees virtual memory which may be anywhere in real memory or paged out to a disk.