* **Abstraction** is good, but don’t forget about reality.
* “Hello world”
* #include <stdio.h>
* int main () {

printf(“hello, world! \n”);

* }
* you create the files:
* hello.c hello.i hello.s hello.o
* -------------------🡪 preprocessor ----🡪 compiler ---🡪 assembler -🡪 linker --
* source program assembly program (binary code)

relocatable object

* printf.o
* -🡪 hello executable obj
* consider the lifetime of the code (editor, system, closed)
* hardware organization of a typical system
* logically: main memory
* linear array of bytes
* each with its own unique address
* physically:
* collection of DRAM

registers

* program
* counter

Cache

Level 1

main memory

* bus interface system bus i/o bridge memory bus
* expansion slots
* USB controller graphic adapter network adapter disk controller
* Mouse keyboard display network disk
* expansions slots
* motherboard is a printed circuit that has expansions slots that you an connect things to, like the main things of your system
* controllers are on the system’s motherboard
* adapters is card that plugs in
* DMA – direct memory access
* The executable is taken from the disk, through the controller to the i/o bridge to the main memory
* From main memory to i/o bridge through the system bus to the registers, and finally to the ALU

Then to the graphic adapter

System takes a lot of time moving from one place to another (“not real work”)

**Caches**

Disk -> main memory -> processor

* -> display device
* definition – a temp. staging area for info the processor may need
  + locality
  + temporality
* things are stored in places such as disk and main memory
* if you access one memory in data, most likely to access it again

registers store words

Word

Cash lines

Disk blocks

small, fast, expensive/ per byte

registers

cache L1 (SRAM)

cache L2 (SRAM)

cache L3 (SRAM)

Main Memory (DRAM)

distributed file system, web servers

local secondary storage - disk

cheaper slower bigger

* bit – binary digit (two possible values)
  + from digit to binary digit, it’s easier to go to a “0/1 system”
* byte – 8 bits
* word – traditionally 32-bits.. each row in main memory would be a word
  + now even 64-bits
* **Operating Systems** manages hardware

application programs

virtual machine

process

virtual memory

files – collection of bytes

OS

i/o devices

main memory

processor

* **processes** – OS abstraction for a running program
  + appear to have complete control
  + reality multiple processes running concurrently
    - context switch – allows you to run then another (so small they appear to be running in parallel)

} context switch

} context switch

kernel code executing

disk interrupt ->

disk interrupt ->

read ->>

Process B

Process A

* **virtual memory** – an abstraction that creates an illusion that a process can access anywhere in memory

|  |
| --- |
| kernel virtual memory |
| user stack (created at run time) |
| memory mapped region for shared libraries |
|  |
| run-time heap (created at run time by malloc) |
| read/write data |
| read-only code and data |
|  |

* invisible to user code