Design and Engineering of Computer Systems

Lecture 2:
Principles of Computer Systems Design

Mythili Vutukuru
IIT Bombay
Principles of Computer Systems Design

• Computer systems are complex, with strict expectations
  • Common design principles underlying the design of diverse systems
• This lecture: brief introduction to the common principles of computer systems design
  • Mostly common sense!
• Rest of the course: several examples of these design principles
• Understanding common principles and seeing common patterns across systems will help designing systems on your own
Modularity

• Systems are composed of multiple independently developed modules ("microservices") which interact with each other

• Example: when e-commerce server displays a webpage to user, information on the page is sourced from multiple modules (personalized recommendations, offers of the day, sponsored products, user’s shopping cart, ...)

browser → web server → shopping cart ← recommendation

→ user database
Abstraction

• Two modules interact via an API (application programmer interface)
  • Need not know the implementation details, only interface

• Example: users write software consisting of sequence of instructions, CPU hardware executes instructions
  • Interface: instruction set of CPU
  • Software need not know how CPU executes instructions

user software — Seq of instructions

CPU hardware instruction
Layering

• Modules organized into layers
  • Lower layers implement a functionality, and provide a service to higher layers
  • Higher layers use service from lower layers, build additional functionality
  • Peers in a layer interact without worrying about other layers

• Example: web browser requests webpage from web server, web server returns the response
  • Higher layer: browser and server are peers, exchange web request/response
  • Lower layer: routers and switches in the network forward data bits
Virtualization

- One physical object, appears as multiple virtual objects
  - Multiple users can use the object, without realizing they are sharing

- Example: multiple processes share a CPU on the computer, but each thinks it is running exclusively on the CPU
  - OS multiplexes multiple processes on same CPU

- Example: multiple virtual machines on same physical machine
  - Illusion of multiple machines on one machine
Hierarchy

- Smaller self-contained units at lower levels assembled into larger units at higher levels of hierarchy
  - Solve problem at a higher level first, then deal with lower level details

- Example: Internet addressing is hierarchical
  - Every machine connected to the Internet has an address (“IP address”) of the format: larger network identifier + smaller sub-network identifier + individual machine identifier
  - Message to a computer routed to bigger network first, then to smaller network
Indirection

• Insert an intermediate entity in a communication between two entities, to hide details of a direct connection
  • Assign a “name”, translate the name later to an address

• Example: Internet URL name resolution
  • Web servers have an address (IP address), which browser must contact
  • Browser requests a web page using a name (www.iitb.ac.in)
  • DNS is a service that maps the name to server IP address
  • Hides low level server address details from users
Parallelism

• Running multiple tasks in parallel for better performance
• Example: modern CPUs have multiple CPU cores, each running a program in parallel
• Example: a web server has multiple replicas to process requests in parallel
Concurrency

- Doing multiple tasks at the same time concurrently
- Related to, but different from, parallelism
- Example: multiple programs running concurrently on a CPU
  - Multiple CPU cores can run multiple programs in parallel
  - Or, multiple programs run concurrently on single CPU core (concurrency without parallelism)
Caching

• Store frequently/recently used objects close by, for quicker access in the future

• Example: memory hierarchy in a computer
  • Program instructions+data stored in memory, fetched by CPU during execution
  • Recently used instructions+data stored closer to CPU in CPU caches
Fixed sizing

• Use standard/fixed sizes of allocating resources, to avoid fragmentation of resource
• Example: RAM (main memory) is divided into fixed size chunks called "pages"
  • A program is assigned memory for code+data in granularity of pages
  • Even if lesser than page size required, full page allocated
• Example: data is stored in granularity of blocks on disk

![Fragmentation Diagram]
Indexing

• Collect metadata about data (information) in one place for easy lookup
• Example: index node (i-node) of a file
  • Data of a file is spread across multiple blocks on disk
  • Index node of a file stores information about all blocks of a file in one place
Separate state from computation

- Processing generates state. Store state separately from processor, so that state can be saved even if processor fails.
- Example: web server stores user data in a database. Even if server crashes or is replaced, user data can be accessed by another server.
Replication

• Save multiple copies of state of a system, so that at least one copy will survive failures
  • Of course, care to be taken to ensure multiple copies are consistently updated
• Example: databases replicate user data in multiple locations for fault tolerance

\[\text{DB} \rightarrow \text{data copy 1} \rightarrow \text{copy 2}\]
Logging

• Keep a record of all actions, so that any missed actions can be completed later

• Example: when making multiple changes to data in a database, keep a log of actions. If failure after doing some actions, can resume remaining actions by replaying from log.
Unlearning

• Sometimes, if there is good reason, ignore the standard principles
• Example: ignore the principle of abstraction, use information about CPU hardware (e.g., CPU cache structure) to optimize software programs

Software \(-\text{Seq of instrn}\)

\[ \overline{\text{CPU instructions}} \]

\[ \text{ISA} \]
Summary

• This lecture:
  • Common principles of designing computer systems ✓
  • Inter-related ideas, mostly common sense ✓
  • We will keep revisiting them throughout the course ✓

• Observe real life computer systems you interact with, and see these principles in action