Unit 5: System Design & Class Design

System Design

- During analysis the focus is on what need to be done
  - **During Design →** developers make decisions about how the problem will be solved.
- System design is the first design stage for devising the basic approach in solving the problem.
- During system design, developers decide the overall structure and style.
- The system architecture determines the **organization of system into sub systems**.

Decision made during System Design

THE FOLLOWING DECISIONS ARE MADE DURING SYSTEM DESIGN:

1. Estimate system Performance
2. Make a reuse plan
3. Organize the system into sub system.
4. Identify concurrency inherent in the problem
5. Allocate subsystem to hardware
6. Manage data stores
7. Handle global resources
8. Choose a software control strategy
9. Handle boundary conditions
10. Set trade-off priorities
11. Select an architectural style.

1. Estimating System Performance:

- Early in planning for new system, you should prepare rough performance estimate.
- Engineers call this as “back of the envelope” calculation.
- The purpose is not to achieve high accuracy, but merely to determine if system is feasible.
- The calculation should be fast, can estimate number of **transactions to be processed by the system**, **response time** needed **storage requirements** etc.

2. Making a Reuse plan:

- Reuse is often cited as an advantage of OO technology but reuse does not happen automatically.
- There are 2 different aspect of reuse
  - **Using existing things**
  - **Creating reusable new things**.

- Reusable things include **models**, **libraries**, **frameworks** and **patterns**.

1) **Reuse of models** is often most practical form of reuse. The logic in a model can apply to multiple problems.

2) **Libraries** – A library is a collection of classes that are useful in many contexts.

- The collection of classes must be carefully organized so that users can find them.

3) Frameworks – A framework is skeletal structure of a program that must be elaborated to build a complete application.

- Framework consists of more than just the classes involved and include a paradigm for flow of control and shared invariants.

4) Patterns – A patterns is a proven solution to a general problem.

- Various pattern target different phases of SDLC.
- There are pattern for analysis, architecture, design and implementation.
- You can achieve **reuse** by using existing pattern.

3. Organizing a System into Subsystems:

- First step in system design is to divide the system into pieces. Each piece of a system is called Subsystem.

- A subsystem is not an object or a function but a **group of classes, associations, operations, events and constraints** that are inter-related and have a well-defined and small interface with other subsystems.

- A system may be divided into smaller subsystems and each subsystem may further be divided into smaller subsystems of its own.

1. Relationships between subsystems:

- There are two types of relationships between subsystems

  a) Client-Server and

  b) Peer-to-peer.

- **Client-Server relationship:** here client calls on the server for performing certain task and server replies back with the result.

- For Ex: when someone checks their bank account from a computer, the computer acts as a client and forwards the request to an online bank (server). Then the bank's program (server) serves a response to the user in which the requested information is displayed.

- **Peer-to-peer relationship:** here, each subsystem may call on the others. The communication in this case can be much complex because individual subsystems may not be aware about each other.

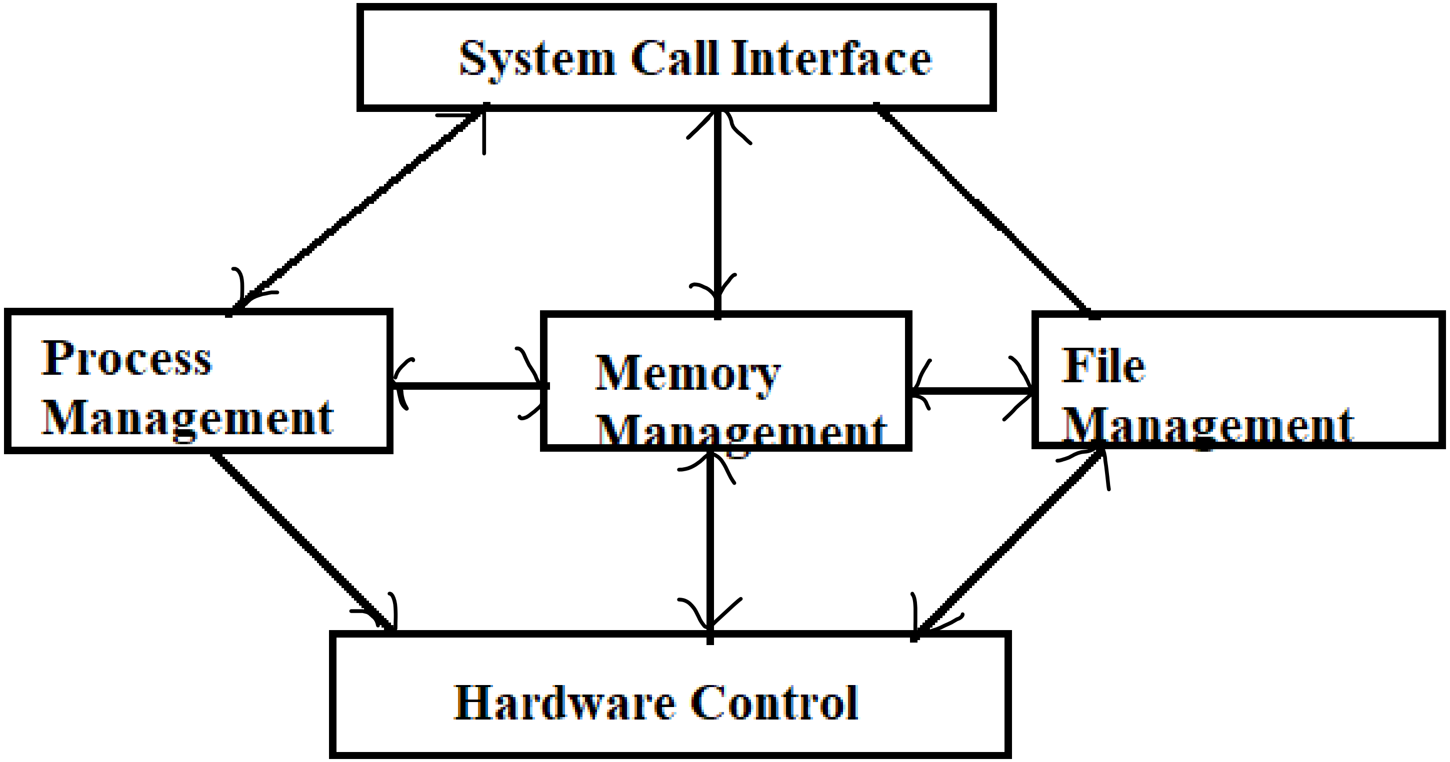
2. Layers and Partitions:

The decomposition (breaking) of system into subsystems may be organized as a sequence of horizontal layers or vertical partitions.

- **Layers:** A layered system is an ordered set of virtual worlds (set of tiers), **each built terms of the ones below it and providing the implementation basis for ones above it.**
- The object in each layer can be independent, although there is often some correspondence between objects in different layers.
- Layered architecture comes in two forms:
  - **Closed architecture** – each layer is built only in terms of immediate lower layer.
  - **Open architecture** – a layer can use features of any lower layer to any depth.

- **Partitions** – Partition **vertically divide** a system into several independent or weakly coupled subsystem, each providing one kind of service.
- Ex: A computer OS includes a file system, process control, Virtual memory management and device control.
- The subsystem may have some knowledge of each other, but this knowledge is not deep and avoids major design dependencies.

Example of Subsystem Partitions (OS Kernel Service)



- **Difference between layer and partitions is:**

1. Layer vary in their level of abstraction.

   Partition merely divide a system into pieces, all of which have a similar level of abstraction.

2. Layer ultimately depend on each other. Usually in client-server relationship.,

   Partitions are peers that are independent or mutually dependent (Peer-to-peer relationship).

Combining layers and partitions

- Layers can be partitioned and Partitions can be layered.
- Ex:

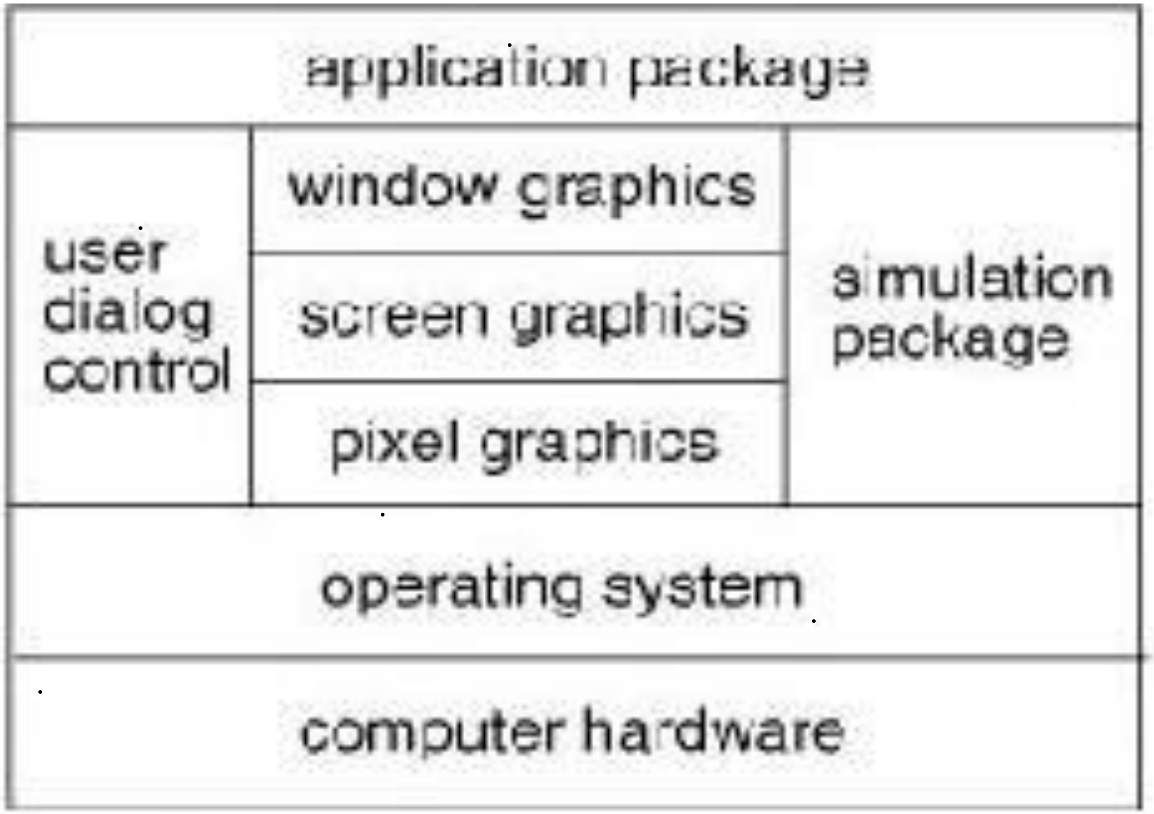


Figure 14.1 Block diagram of a typical application.
Most large systems mix layers and partitions.

4. Identifying Concurrency:

- concurrency can be very important for **improving the efficiency of a system.**
- One important goal of system design is to **identify the objects that must be active concurrently** & the objects that have **mutually exclusive activity**.
- **State model** is the guide to identifying concurrency. We can identify inherent concurrency and concurrent tasks.
- Identifying inherent concurrency - Two objects are inherently concurrent if they can receive events at the same time without interacting.
- Defining Concurrent Tasks – By examining the state diagrams of individual objects and the exchange of events among them, you can define concurrent tasks.

5. Allocation of Subsystems:

- Allocate each concurrent subsystem to a hardware unit, as follows:

1. Estimate performance needs and the resources needs to satisfy them.
2. Choose hardware or software for implementation of sub-systems
3. Allocate software subsystems to processors to satisfy performance needs and minimize inter processor communication (IPC).
4. Determine the connectivity of the physical units that implement the subsystems

6. Management of Data Storage:

- There are several alternatives for data storage, you can use **separately** or **in-combination: data structures, files and databases.**

- Different kinds of data stores provide trade-offs among cost, access time, capacity & reliability

- Ex: 1) A personal computer applications may use memory **data structures** and **files**.

  2) An accounting system may use a **database** to connect subsystem.

- Files are cheap, simple and permanent.
- File implementations vary for different computer systems, so portable applications must carefully isolate file-system dependencies.
- Databases, managed by DBMS systems are another kind of data store. Various types of DBMSs are available from vendors, including relational and OO.

- OO-DBMSs have not become popular in the mass market. So you should consider them only for specialty applications that have a wide variety of data types.

- ATM Eg; typical bank computer would use relational DBMS – they are fast, readily available, and cost-effective for kinds of financial applications.

7. Handling Global Resources:

- The system designer must identify global resources and determine mechanisms for controlling access to them.

There are several kinds of global resources

1. **Physical units:** Ex: include processors, tape drives & communication satellites.
2. **Space** Ex: include disk space, a workstation screen & the buttons on mouse.
3. **Logical names** Ex: include object IDs, filenames & class names.
4. **Access to shared data:** for example databases.

- Physical resource such as processors, tape drives etc. can control their own access by establishing a protocol for obtaining access.

- For logical resource like object ID or a database, there arises a need to provide access in a shared environment without any conflicts.
- One way to avoid conflict may be to employ a guardian object which controls access to all other resources. Any request to access a resource has to pass through a guardian object only.

8. Choosing a software control strategy:

- There are two kinds of control flows in software system:
  1. **External control**
  2. **Internal control**

1. External control - concerns the flow of externally visible events among the objects in the system.

- There are **three kinds of control** for external events
  - Procedure-driven sequential control.
  - Event-driven sequential control.
  - Concurrent system control.

- **Procedure-driven control:** the control lies within the program code. Procedure request external input and then wait for it, when input arrives control resumes within the procedure that made the call.

- **Event-driven control:** in sequential model, the control resides within a dispatcher or monitor that the language, subsystem or operating system provides.
  - In event driven, the developers attach application procedures to event and the dispatcher calls the procedures when the corresponding events occur.

- **Concurrent system control:** here control resides concurrently in several independent objects, each as a separate task.
  - A task can wait for input, but other tasks continue execution.

2. Internal control - refers to the flow of control within a process. It exist only in implementation and therefore is neither inherently concurrent nor sequential.

- There are 3 kinds of control flow :
  - Procedure calls,
  - Quasi-concurrent intertask calls eg: coroutines or lightweight processes.
  - Concurrent intertask calls.

9. Handling Boundary Conditions:

- Although most of system design concerns steady state behavior, consider boundary condition as well and address the following kinds of issues:

- **Initialization:** the system must initialize constant data, parameters, global variables, tasks, guardian objects and possibly the class hierarchy itself.

- **Termination:-** It is usually simpler than initialization because many internal objects can simply be abandoned

- The task must release any external resources that it had reserved.
- In concurrent system one task must notify other tasks about its termination

- **Failure :-** It is an unplanned termination of a system.

- Failure can arise from user errors, exhaustion of system resources, or an external breakdown.

- Failure can also arise from bugs in the system.

10. Setting Trade-off priorities:

- The system designer must set priorities that will be used to guide trade-off for the rest of design.
- These priorities reconcile desirable but incompatible goal.

Ex: a system can often be made faster by using extra memory, but that increases power consumption and costs more.

- Design trade-off involve not only the software itself but also the process of developing it.

- Design trade-off affect the entire character of system. The success or failure of the final product may depend on how well its goals are chosen.

11. Common Architectural Styles

- Several prototypical architectural styles are common in existing systems.
- Each of these is well suited to certain kind of system.

- Some kinds of system are listed below

1. **Batch transformation –** a data transformation executed once on an entire input set.

2. **Continuous transformation** – A data transformation performed continuously as inputs change.
3. **Interactive interface** – a system dominated by external interactions.
4. **Dynamic simulation-** a system that simulates evolving real-world objects.
5. **Real-time system** – a system dominated by strict timing constraints.
6. **Transaction manager** – a system concerned with storing and updating data, often including concurrent access from different physical locations.