General OS Structure

Ways of Structuring

1) Monolithic - UNIX, Multics
2) Object - Amoeba
3) Hierarchical - THE
4) Small kernel w/OS activity in user processes - Amoeba, V

Issues:

Development and Testability:
- Hierarchical approach allows you to develop a particular portion of the OS and test it extensively.   Thus, addition of next portion of hierarchy need only be tested for a limited set of cases since correctness of the underlying has already been established. This constrains the number of test cases.
- Objects can only be used in certain ways, creating specific invariants that are true for the object. Testing to make sure the invariants are always true (and that's it) limits the amount of testing required.

Security:
- In the monolithic model, all privileges are given once you're in kernel model. This means that every module can alter data in other modules even though logically they should not be permitted to do so.
- The object model uses capabilities to insure that only specific operations may be performed by privileged users. This means that access to objects is restricted and prevents either accidental or malicious alteration of objects. In Amoeba, objects and associated capabilities are associated with specific server ports. Capabilities are only given to processes that have the right to perform operations on that object.
- In the hierarchical model, protection is provided between layers by establishing interfaces that must be used. Each lower layer has more privileges than the one above it. This means that higher layers cannot interfere with portions of the operating system that they do not have privileges to.
- In the small kernel model, protection is provided by only creating a small kernel with absolute privileges. By moving functionality into many user processes, each processes is limited regarding what damage it may do to processes providing other functionality.

Concurrency
Monolithic kernels perform coarse-grain locking. The other structuring models may provide a way to perform locking on a finer grain. Locking may be provided per object or per level in a hierarchy. If this is the case, more concurrency can be provided. Additionally, the small kernel clearly creates more concurrency because many user processes (previously in the kernel) can be executing at the same time.

General Trends in OS
Transparency
- V and Amoeba don't tell the user where the activity will be performed.
Hardware Independence
- Device independent I/O is established.
- Memory is made virtual and has its use automated.
Abstract Interfaces
- In Multics, all data is represented as segments.
- In UNIX, files are used as a general abstraction for dealing with pipes, filters, I/O device operations.
Movement of common functionality into OS

Specific OS

<u>MVS</u>
Issues:
Backwards compatibility
Movement of common code into OS
Allow reconfiguration of hardware with
- device independent I/O
- uniform access interfaces

Automation of previously manual tasks – scheduling, memory
Automated recovery in software

<u>Multics</u>
Tried lots of different ideas in one system like
- virtual memory – paging and segmentation
- sharing/protection
- segment interface

Written in higher level language
Ability to iteratively optimize modules while continually providing functionality
Users are not affected by processes of other users.
Device independent I/O
They state:
    Module division of responsibility
    Dynamic reconfiguration of HW
    Automatically managed multilevel memory
    Protection of programs and data
    System programming language