Operating Systems

1

Significance of this lecture

- This lecture focuses on a node and their central control mechanisms
- How is software linked to physical hardware?
- What controls exist for our physical resources?
- This lesson is concerned with important operating system concepts that impact on distributed system design and behaviour.

2

Topics

- Topics we will cover
  - Operating System Architectures (Monolithic Operating Systems, Microkernels)
  - Kernels
  - Protection
  - Processes and Threads
    - Statefull vs stateless server design
    - Single threaded vs multithreaded server design
  - Architectures for multithreaded servers
  - Thread synchronisation and thread scheduling

3

The DIS model

- Resource sharing is an important aspect of a DIS
- What is involved in sharing?
  - Clients invoke operations on resources on remote nodes or other local processes
    - This process is simplified through the use of middleware
  - In a DIS client applications/services send request to the middleware
  - The middleware then sends its requests to the host operating system (OS)
  - The OS then utilizes computer and network hardware to send requests
- So "What is the role of the OS?"
  - The OS provide an abstraction of the hardware (network and computer)
    - i.e. processors, memory, communications, storage, etc.
  - Middleware talks to the OS rather than the hardware directly

4

Layers of a computer

Application, services

Middleware

OS

Computer / network hardware

5

Layers of a computer

Application, services

Middleware

OS

platform

Computer / network hardware

6

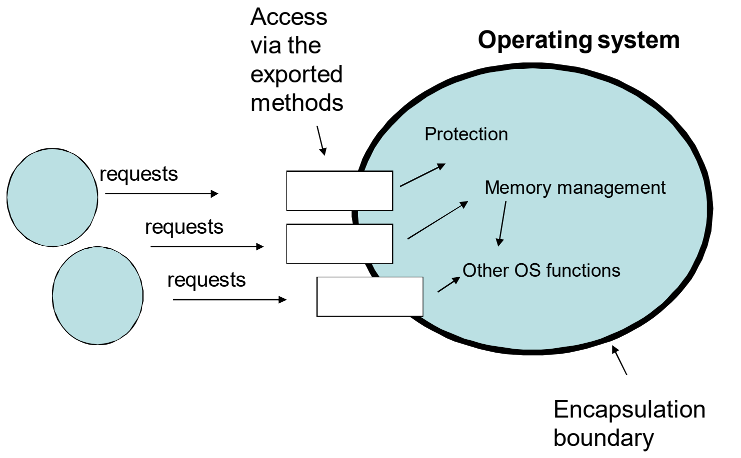
OS layers

- Required functionality of an OS
  - Encapsulation:
    - With a **useful** set of operations to clients
    - Unnecessary details should be transparent (memory management)
  - Protection:
    - protection to resources from incorrect and illegitimate access
  - Concurrent processing:
    - transparent concurrent process management

- With respect to remote invocations an OS also needs the following functionality:
  - Communication:
    - to remote processes or at least other processes within the computer
  - Scheduling
    - Invoked operations need to be schedules to be processed.

7

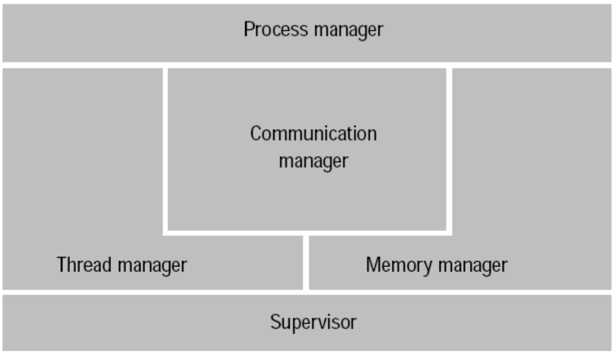
Encapsulation



8

The core OS functions

Micro-kernel architecture



9

OS layers

- Process manager:
  - Manages the creation and operations of processes
- Thread manager:
  - Creates, synchronizes and schedules threads
- Communication manager:
  - Communication between threads on the same host
- Memory manager:
  - Physical and virtual memory
- Supervisor:
  - The base layer - the hardware abstraction layer
  - Interrupts, system traps, register manipulation, control of the memory management, processor control

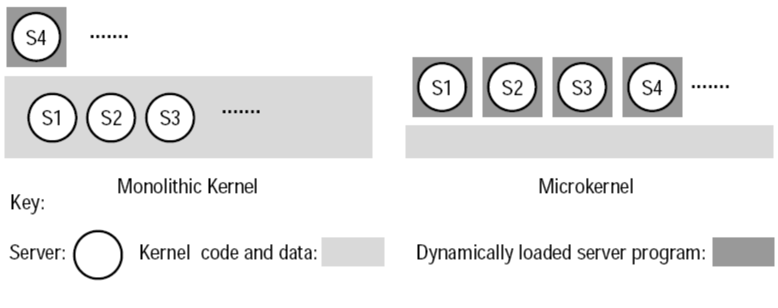
10

Micro/Monolithic kernels

- Micro kernel
  - Implement as much as possible outside the kernel
  - Execute less code in Kernel Mode
  - Leads to more context switches
  - Requires inter process messaging
- Monolithic kernel
  - Nearly all functions exist within the kernel
  - Executes more code executed in kernel mode
  - Kernels can become very large
  - Faster execution
  - Has high function and code dependence

11

Micro/Monolithic kernels



12

Safeguards and protection

- Resources require protection from illegitimate access
  - illegitimate access is access without privileges
    - An operation is performed that the client does not have access rights to perform
  - illegitimate access is access in unplanned way
    - An operation is performed that is not permitted at all
- Protection is mainly the responsibility of the OS kernel

13

Safeguards and protection

- Features of the OS Kernel:
  - It is always running,
  - Its code has complete access to the physical resources of the computer
    - Processors, memory, etc.
- The kernel allocates resources to processes
  - For example, memory is allocated to user processes and they can only access memory inside its allocated region of memory
- Kernel protection
  - **User** processes run their own code in their allocated resource space
  - However **user** processes can also run kernel code
  - They do this through calling kernel supplied handlers
    - These handlers control the access safeguarding against illicit activities
  - Using interrupts or machine-level traps

14

Processes and threads

- A process typically contains an execution environment
  - Execution environment
    - Address space
    - Thread synchronization and communications
    - High level resources (open files etc)
  - Execution environments are expensive to set-up
  - Within its environment a process could decide to run many activities concurrently
  - These concurrent activities are called threads

15

Processes and threads

- Creating a process
  - Creating the process and its execution environment
  - More complex in a DIS
    - Chose a target host
    - Maybe determined by policy
      - i.e. Always at originator
      - Could be based on load balancing (see pp217-218)
  - Create the execution environment on the selected host
    - Two approaches:
      - Share the address space of the existing execution environment
      - Create a new address space

16

Processes and threads

- Threads (Threads of execution)
  - Exist and run within a process execution environment
  - Cannot access threads in other processes
  - Can be dynamically created and destroyed by the process
  - Allow process to perform concurrent operations

17

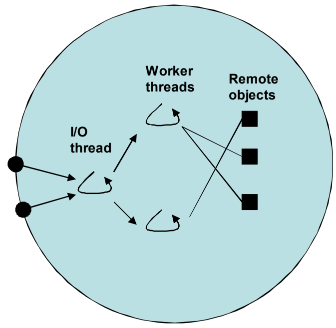
Processes and threads

- Threads
  - Threads improve performance through concurrency
  - Consider:
    - A single threaded server processes in which it takes 2ms to process a client request
      - Now, assume we have 10 requests = 20ms to process
    - A multi-threaded server processes with 2 threads that can run concurrently
      - We have 10 requests going to 2 different threads
      - 5 requests for each threads so that's 10m/s to process
    - With 5 threads we might reduced 4m/s
  - Of course there might be other bottlenecks
    - serial hardware such as disk access
    - caching could help for disk access

18

Server threading

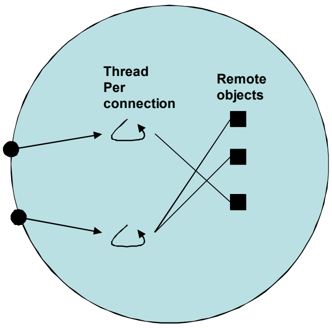
- Thread per request
  - A new thread is generated for each request
  - They are destroyed when the request completes
  - Increases throughput but introduces creation and destruction overheads
  - This is managed via an I/O thread



19

Server threading

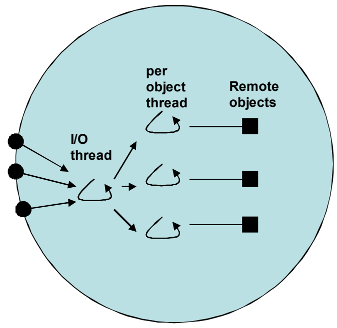
- Thread per connection
  - Creates a new thread for each connection that is made to the server
  - Connection could make many requests through its single thread



20

Server threading

- Thread per object
  - Allocates a thread to each remote object
  - This is managed via an I/O thread



21

Client threading

- Client threads
  - Client threading is also beneficial
  - Consider an RMI call that blocks
    - A thread could be generated for the request that would block until its responses
    - Another thread could then continue working whilst the request is serviced

22

Pros and cons of threads

- Advantages of multi-threads
  - Threads are more economical to create as they share a execution environment
  - Switching between threads is cheaper than switching between processes
  - Process threads can share data and resources easily
- Disadvantages of multi-threads
  - Sharing data and resources circumvents the OS protection
  - A thread might alter data being used by another thread!

23

Thread synchronization

- Concurrency with threads can lead to errors
  - as lost updates for example
- Solution
  - Use a monitor to specific portions of code, entire methods or objects
  - A **Monitor** is an abstract data type, i.e. a combination of data structures and operations, where at most one thread may be executing at one time.
    - The role of the monitor is to ensure that **only one thread** has access to the specified portion of code at any one time
    - Thus threads might be told to block whilst they wait for access (a notification) that the object or code portion is free

24

Thread scheduling

- Threads need to be told when to **run** and when to **stop**
- There are two main alternatives:
  - Preemptive scheduling
    - Greatest control
    - Can have problems with synchronization
  - Non-preemptive scheduling
    - Avoids some synchronization problems
    - Offers less control
    - Threads can only be halted when they make a system call
    - Could lead to long executions without an opportunity to halt the process

25

Stateless and Stateful servers

- Information that a server maintains about the status of ongoing interactions with clients is called *state* information
- Stateless servers
  - Do not keep state on clients and can change their own state without informing the client
  - Web servers are the classic example of stateless servers
  - Simplifies recovery after crashes
  - Can lead to improved scalability
- Stateful servers
  - Maintain state on each of their clients
  - For example file servers are often (but not always) stateful in terms of maintaining information on clients (caching status,etc)
  - Makes crash recovery more complex
  - Can lead to performance optimisation

26

Conclusions

- Operating System Architectures
  - Monolithic kernels
  - Microkernels
- Kernels
- Protection
- Processes and Threads
  - Stateful Vs stateless server design
  - Single threaded Vs multithreaded server design
- Architectures for multithreaded servers
- Thread synchronisation and thread scheduling

27

END

28