OS System Calls and OS Structures

- System Calls
  - User programmes need to interact with the OS
  - A system call is the main way a user programme interacts with the OS
  - How is this done?
- OS structures:
  - Monolithic/Simple Structure
  - Layered Approach
  - Microkernels
  - Other

OS System Calls

- Protection Issues
  - CPU
    - Kernel can take CPU away from users (preemption) => users should not be able to do this
  - Memory
    - Prevent a user from accessing other user's data
    - Prevent users from modifying kernel code
  - I/O
    - Prevent users from doing illegal I/O operations
- => need to separate out functions
  - User code can be arbitrary but should not be able to modify kernel memory or code

OS System Calls

- Concept of privileged mode
  - User mode
    - Regular instructions
    - Access user memory
  - Privileged (kernel) mode
    - Regular instructions
    - Privileged instructions
    - Access user memory
    - Access kernel memory
    - E.g. memory address mapping, halt a processor

System Call Mechanism

- Design issues
  - User code makes a system call with parameters
  - The call mechanism switches code to kernel mode
  - Execute system call
  - Return with results
- OS API
  - Interface between an application and the OS kernel
  - Various ways of passing parameters
    - Registers, list, stack
  - How to return results, error messages?

System Calls

- How a system call works:
  - obtain access to system space
  - do parameter validation
  - system resource collection
  - ask device/system for requested item
  - suspend waiting for device
  - Interrupt makes this process (thread) ready to run
  - wrap-up
  - return to user
- example: read(fd, buffer, nbytes) - 11 steps

OS Structures

- Important to separate policies and mechanisms
  - Policy: what will be done?
  - Mechanism: how will it be done?
- Separate allows flexibility
  - E.g. use timer to protect CPU (mechanism), how long to set timer (policy)
- Simple Structure
- Layered Approach
- Microkernels
- Other

Monolithic/Simple Structure

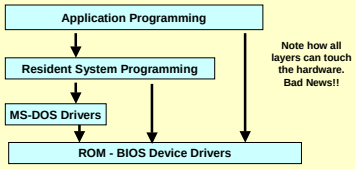
- No well defined structure
- All kernel routines are together, any can call any
- A system call interface (main programme, sys call, utility functions)
- Start off as small, simple, limited systems, and then grow
- Advantages:
  - Shared kernel space
  - Good performance
- Disadvantages
  - No information hiding
  - Inflexible
  - Chaotic
  - Difficult to understand

Monolithic/Simple Structure

- No well defined structure
- All kernel routines are together, any can call any
- A system call interface (main programme, sys call, utility functions)
- Start off as small, simple, limited systems, and then grow
- Advantages:
  - Shared kernel space
  - Good performance
- Disadvantages
  - No information hiding

Example: MS-DOS

MS-DOS
- Written to provide the most functionality in the least space
  - Not divided into modules
  - Its interface and levels of function are not well separated



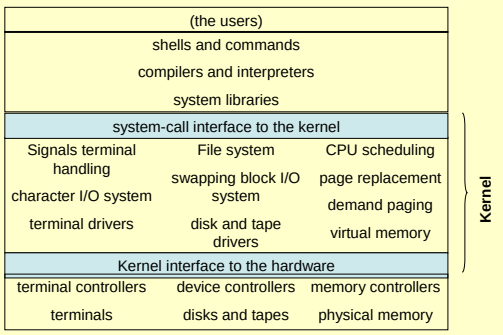
Layered Approach

- OS is divided into a number of layers (levels), each built on top of lower levels
  - The bottom layer (layer 0) is the h/w
  - The highest layer (layer N) is the user interface (UI)
- Layers are selected such that each uses functions (operations) and services of only lower-level layers
- GLUnix: Global Layered Unix

Layered Structure

- Unix
  - Original OS had limited structuring (due to limited h/w functionality)
  - 2 parts
    - System programmes
    - Kernel
      - consists of everything below the system-call interface and above the physical h/w
      - Provides the file system, CPU scheduling, memory management, and other OS functions

Unix



Microkernel

- Move all non-essential components from the kernel in to “user” space
- Main function: communication between client programmes and various services which are run in user space
  - Uses messages (no direct interaction)
- Advantages:
  - Easier to extend OS
  - Easier to port the OS to new architecture
  - More reliable (less code in kernel mode)
  - More secure
- Disadvantages:
  - Performance overhead of user space to kernel space communication

Modular Approach

- Most modern OSs implement kernel modules
  - Uses object-oriented approach
  - Each core component is separate
  - Each talks to the others over know interfaces
  - Each is loadable as needed within the kernel
- Similar to layers but more flexible
  - Naw module can call any other module
- Examples: Solaris

Solaris Modular Approach