Registry → centralized database to store configuration setting and options.

Tashfeen Abbasi

Chapter no 2 (2.1, 2.2, 2.3, 2.4, 2.6, 2.8, 2.9)

2.1 Operating system Services.

- User Interface (UI) → • graphical user interface (GUI) on windows and mice. (regular users)
  - touch interfaces on mobile phones
  - Command line Interfaces (CLI) for text-based inputs
- Interprocess Communication (IPC) → • allows processes to communicate with each other
  - Shared Memory (Processes communicate by reading & writing to shared region of memory)
  - Message Passing (Processes send and receive messages to exchange data). (done by pipes or sockets).
- Command-line-interface (CLI) → • allows users to input textual commands and executed by OS.
  - CLI relies on command interpreter (also used in Shell in UNIX or LINUX)

prefered by users with deep system knowledge.

Examples: • Bourne-Again Shell (Bash) • C shell (csh) • Korn Shell (ksh)

- Methods to implement commands in (CLI)
  - Interpreter-Inclusive (Built-in-commands) → part of shell. • Command interpreter (Shell) contains code for each command. • Command like creating, deleting or copying files directly trigger code inside interpreter
  - System programs (Commands are handled by separate program) → shell finds & run these commands. • interpreter only do external programs to perform commands
    - UNIX → • rm command to delete file but executes external program named rm.

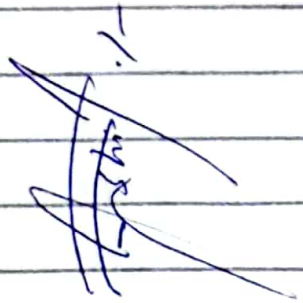
- GUI offers a user-friendly alternative to CLI.
- Xerox Alto (1973) → first computer to feature a GUI.
- GUI gained popularity with Apple Macintosh in 1980s.
  ↓ evolved and adopt aqua interface in macOS.
- UNIX system used CLI but now feature GUI like KDE & GNOME.
- Touch-Screen interface → • modern phones uses.
- macOS → • Histovically GUI-focused (Aqua GUI)
  • Now, provides GUI & CLI due to Unix-based kernal.

2.3 System Calls (way of a program to request services from OS.)
↓
Bridge b/w Software & operating system. (function in APIs. typically invoke system calls)
- getting files names → opening files → reading & writing → closing files.

i) • Application Programming interface (APIs) → set of function avalible to programmer,
(Set of rules and tools that allows software application to communicate with each other).
  • Window API
  • POSIX API
  • Java API.
- Programmer access APIs via libraries provided by OS. e.g., UNIX & Linux in C library called as C library or glibc.
- Advantages → • Portability (if program uses api, move program to different ports or computers, as long as those system also supports same API)
  (API & RTE hide implementation details from programmer)
  • Abstraction (easy to use complex system function)
  ↓ API interact with OS easily rather than direct system calls.
- Runtime environment (RTE) → • provides System call interface which links code with system's system calls.
- System call interface → maintains table of system calls, associated number

Boot loader → Small program that initilizes system hardware and loads OS into memory when a computer starts up.
(Grub)

- Parameters passing → • Registers (parameters are passed directly in CPU register)
• Block / memory table (Parameters are stored in memory blocks, address of this block is passed in a register)
• Stack (Parameter pushed onto a stack by program and popped off by OS.



ii) System calls Categories →
• Process Control (system call manages execution of the process)
• File Management
• Device Management (manage hardware devices)
• Information Maintenance
• Communication
• Protection.

• Message Passing Model
• Shared-memory Model

Single Step mode / Trap mode
• CPU mode for debugging
• After every instruction is executed, CPU generates a trap or interrupt.

2.4 System Services / System Utilities.

1) File Management
2) Status information
3) File Modificatio
4) Programming language support
5) Program loading & execution
6) Communications
7) Background Services.

Application Binary interface (ABI) → How binary code components interact at low level e.g., addressing, parameter passing & stack organization.
Compiled code can run

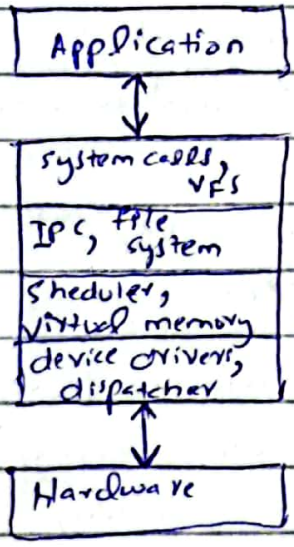
2.6 Why Applications are OS specific?

- System calls & APIs → • OS provides unique set of system calls & API that application use to interact with OS.

- Application to be made available on multiple OS →
  i) Interpreted languages (python, ruby)
  (• Runs on any OS with interpreter.
  • Slower performance)
  ii) Language with VM (Java)
  • Compiles code into bytecode
  iii) Standard languages or APIs (POSIX or UNIX)
  • Compiles code for specific OS.

2.8 OS Structure

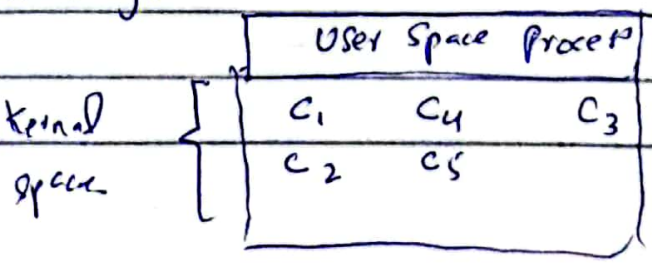
- Monolithic Structure →
  • Core functions run in a single address space.
  • Combines OS components into one large executable file.
  • Orignal UNIX used monolithic.
  • Linux also uses monolithic kernal approach. It runs in a single address space. Uses a modular design → components can be loaded & unloaded dynamically.



1) larger
2) faster
3) If one component crash, whole system crashes.
4) difficult to add new functionality
5) difficult debugging.

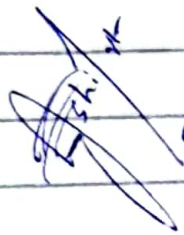
- Advantages → • offer high performance
  • running all kernal operations in single address space, avoid performance costs.
- Challenges → • Complex due to large size
  • Changes might effect other parts of kernal.

For understanding:



Each layer hides complexity of its lower layers from higher layers.

- Layered Approach → (no. of layers (levels) each built on top of lower layer.)



used in Computer networks and web application

Bottom layer: Hardware
Higest layer: user interface.

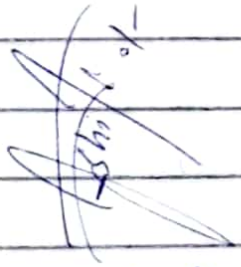
- Each layer only interacts with its adjacent layer (higher layers use function from lower layer)
- Each layer is implemented using operations and services provided by lower layers.
- This design allows changes in one layer to impact not the entire system.

- Advantages →
  - Easier debugging, as layers are tested and developed independently.
  - layers only need to know what operations do, not how they are implement.
- Challenges →
  - defining clear layer can be difficult.
  - reduce system performance due to multiple layers

- Microkernal → (minimizes functionality within kernal itself)
  - Minimal core:-
    - handles essential function like process and memory management
  - Message passing:- Communication occurs through message passing, ensuring user programs do not directly interact with kernal.
  - Adding new services is easier.
  - Examples → Darwin (macOS, iOS) → mach microkernal
    QNX (embedded system) → neutrino microkernal
- Challenges → suffer from performance issues due to overhead of message passing and context switching between processes.

Loadable Kernal modules (LKMs) → methodology designed for operating-system design
(Code to extend running kernal)

- **Modules →**
  - LKM approach allows the kernal to have core set of components, with additional services dynamically linked either at boot time or during runtime.
  - e.g., Linux uses LKMs to support device drivers and filesystems which can be inserted into kernal when needed.



- **Hybrid System →**
  - most OS use a combination of different structures
    - Linux → Monolithic Kernal
    - Windows → largely monolithic.

↓

different components or features from different structures.
e.g., Combine features from monolithic kernal and microkernal into one cohesive kernal design.