Chapter 2: Operating System Structures

Contents

1 Operating System Services

1) **User Interface:**
   a) Most OSes have a **user interface** (UI), most commonly a **graphical user interface** (GUI).
   b) In GUIs, the interface is a window system with a pointing device to click and a keyboard to enter twxt.
   c) Mobile phones and tables have a **touch-screen interface**, enabling selection and typing with fingers.
   d) A **command line interface** (CLI) users text commands only.
   e) Some systems may provide more than one of these variations.
2) **Program execution:** The system must be able to load and run a program. The program must terminate, either normally or abnormally, raising an error.
3) **I/O Operations:** A running program may require I/O. For efficiency and protection, users cannot directly control I/O devices. OS provides a means to do I/O.
4) **File system manipulation:**
   a) Programs need to be able to create/delete files, read/write files and search by name.
   b) OSes include permissions on management and ownership of files.
   c) Many OSes provide a variety of file systems offering specific features and performance characteristics.
5) **Communications:**
   a) Processes may need to exchange information with each other. These processes may be executing on the same computer or networked systems.
   b) **Shared memory** involves processes reading and writing to a shared section of memory.
   c) **Message parsing** involves packets of information in predefined formats that are moved by processes.
6) **Error detection:**
   a) OS must be able to detect and correct errors.
   b) These may occur in CPU and memory hardware (memory error or power failure), I/O devices (connection error, parity error) or in the user program (arithmetic overflow, illegal instruction).
   c) The OS must take appropriate action to ensure correct and consistent computing. It may have to halt the system, or could return

an error code.

7) **Resource allocation:**
   a) Resources must be allocated to multiple processes running at the same time. OS may manage different types of resources.
   b) Some resources may have specific allocation code, while others may have a more general request code. There may be other routines for peripheral devices.
   c) CPU-scheduling routines take into account CPU speed, number of queued processes and so on while allocating resources.
8) **Logging:**
   a) Used to keep track of which processes use what resources.
   b) May be used for billing, accumulating statistics and configuring the system to enhance performance.
9) **Protection and security:**
   a) Owners of information may want to control its access in multiuser or networked system.
   b) Concurrently executing processes should not be able to interfere with each other.
   c) Protection involves ensuring that access to system resources is controlled.
   d) Security involves preventing outside users access to the system. This involves password authentication and recording I/O devices for instances of invalid access or break-ins.

2 User and Operating System Interface

2.1 Command Interpreters

1) A **command interpreter** allows users to directly enter commands to be performed by the OS.
2) Most OSes treat the command interpreter as a special program when a user logs on or a process starts.
3) OSes that provide mulitple command interpretes call them **shells**. For example, *Bourne Again Shell* (`bash`) or *Korn Shell*.
4) Main function of command interpreter is to get and execute the next user command. A common approach is to load a program and execute it with arguments.

2.2 Graphical User Interface

1) **Icons** represent programs, files, functions and directories.
2) Clicking with the mouse can open menus or directories, called **folders**.
3) Examples of GUIs for UNIX systems include KDE, GNOME, etc.

2.3 Touch Screen Interface

1) Command-line interface or mouse and keyboard system is impractical for mobile phones. Touchscreen interface used instead.
2) Interaction is made by using **gestures** on the touch screen.
3) Keyboards are simulated on the touch screen.

2.4 Choice of Interface

1) **System administrators** and **power users** who know systems use the CLI, as it is more efficient and faster access, and makes repetitive tasks easier, partly due to their own programmability (**shell scripts**).
2) Casual users use GUIs for tasks such as browsing the web or watching videos.
3) User interfaces may vary between systems and also between users within a system.

3 System Calls

1) **System calls** provide an interface to the services made available by an OS.
2) Generally low level functions implemented in C/C++/Assembly.

3.1 Application Programming Interface

1) Application developers design programs according to an **application programming interface** (API). The API specifies a set of functions that are available to an application programmer, along with parameters to pass and return values to expect.
2) The **libc** library written in C for UNIX OSes is an API to access OS code. These API functions typically implement actual system calls.
3) Benefits of transferring control to an API include portability and ease of use.
4) The **run time environment** (RTE) is a software suite needed to execute applications in a programming language. RTEs provide a **system call interface** that links system calls to the API functions.

5) Programmers must only obey the API, while most of the OS interface is hidden from them.
6) Three methods used to pash parameters to the OS:
   a) Using registers (at most five parameters).
   b) Storing parameters in a block or a table and storing the memory address of the block in the register.
   c) Using **stacks** to **push** or **pop** parameters.

3.2 Types of System Calls

3.2.1 Process Control:

1) If a system call terminates a program abnormally, the memory of the system at that point is dumped and an error message shown.
2) Memory dumps may be examined using a **debugger** to find errors, or **bugs** in a program.
3) Users interact with the command interpreter to issue appropriate actions in case of error.
4) Some systems may allow special recovery actions in case of error.
5) Error levels may be defined to distinguish severity of errors.
6) Processes should be able to create new child processes and access their attributes.
7) OS must manage concurrent access of a shared resource, usually via **locks**.

3.2.2 File Management:

1) System calls are required to create/delete files.
2) Files should be opened, read/written/stream repositioned and closed via system calls.
3) System calls should also provide for accessing attributes of the file.
4) System calls can be provided for copying and moving files.

3.2.3 Device Management:

1) Various resources controlled by OS can be thought of as devices. They can be either physical or virtual.
2) A system should request for exclusive access and be able to release a device.
3) When the device is allocated, system should be able to read, write and reposition the device, just like files.

3.2.4 Information Maintenance:

1) Includes system calls for current timestamp and system information.
2) System calls to dump memory and print stack-trace.
3) Microprocessors provide **single step** mode, where the CPU executes a trap caught by a debugger.
4) System calls to provide time profiles of a program; these indicate the amount of time a program executes at a set of locations.
5) System calls to get and set process attributes.

3.2.5 Communication:

1) **Message-passing model:**
   a) Communicating processes exchange messages with one another to transfer information.
   b) Communication may be direct or indirect through a mailbox.
   c) A connection must be opened to communicate.
   d) **Host names** or IP addresses of computers in a network or **process names** of various processes within a computer must be translated into known form for the OS to refer.
   e) Processes that receive connections are called **daemons**, which are system programs for this purpose.
   f) The source of communication is known as the **client** and the receiveing daemon is known as the **server**.
   g) System calls are provided for translation of names, managing connections and for sending messages.
2) **Shared-memory model:**
   a) Information exchanged by reading and writing in shared memory areas between processes.
   b) Processes must ensure that they are not writing simultaneously to the shared location.
   c) System calls are used to open and close shared memory regions.
3) Message passing is useful for smaller amounts of data, and easier to implement than shared memory.
4) Shared memory allows for maximum speed and convenience of communication. However, problems may exist in protection and synchronization of processes.

3.2.6 Protection:

1) Provides a mechanism to control access to resources.
2) System calls can set permissions of files and disks, or prevent certain users from accessing

resources.

4 System Services

1) **System services** or **System utilities** provide a convenient environment for program development and execution.
2) The various types of system services are:
   a) **File management:** Perform operations such as create, delete, list, access, rename, etc. on files and directories.
   b) **Status information:** Provide system time and resources; can be more complex, providing detailed performance, logging and debugging information. Some provide a **registry** to store and retrieve configuration information.
   c) **File modification:** Text editors may create and modify content of files, or search and perform transformations on the text.
   d) **Programming language support:** Compilers, interpreters, assemblers and debuggers provided to create programs in high-level programming languages.
   e) **Program loading and execution:** Loaders, linkers, and to execute and debug a program.
   f) **Communications:** These programs create connections among processes, users and systems. Allow for sending messages, file transfer or logging in remotely.
   g) **Background services:** System-program processes called daemons, **services** or **subsystems**. For example, daemons to manage remote connections, process schedulers, system monitoring error services, etc.
3) **Application programs** such as web browsers, e-mail clients, video/image viewers, word processors, etc. useful for users to solve everyday problems.

5 Linkers and Loaders

1) Source files are compiled into **relocatable object files**, that can be loaded into any physical location in memory.
2) A **linker** combines the relocatable object files into a single binary **executable** file. Other library object files may be included in this phase as well.
3) A **loader** loads binary executable files into memory locations where eligible to run on a CPU core.
4) Associated with linking and loading is **relocation**, which refers to assigning the final addresses of program parts and adjusting code and data to match those addresses so that library functions and variables can be accessed during execution.
5) When an executable is to be run, the shell forks a process for it, then invokes the loader to load the specified program into memory using the address space of the process.
6) Systems allow a program to dynamically link libraries when being loaded. These are known in Windows as dynamically linked libraries (**DLLs**).
7) Advantages of dynamic linking are that unnecessary libraries are not linked and loaded, and multiple processes can save memory use by sharing DLLs.
8) In UNIX systems, object files and executables are stored in **Executable and Linkable Format** (ELF). There are seaprate ELF formats for executable and relocatable files.
9) An ELF executable contains the program's entry point, which contains the first instruction that the program runs.
10) Windows uses the **Portable Executable** (PE) format and macOS uses the **Mach-O** format.

6 Why Applications are Operating System Specific

1) An application can be made cross-platform using these approaches:
   a) The application is written in an interpreted language such as Python or Ruby, which has an interpreter for multiple OSes. Performance is limited compared to the native OS, and the interpreter provides a subset of the features of the OS, limiting feature sets of the applications.
   b) The application can be written in an language that requires a virtual machine for execution, for example Java. It has an RTE, including a loader, byte-code verifier, etc. ported to various OSes to load and run the application in the virtual machine. Same disadvantages as for interpreters.
   c) Application is written in an OS specific API, hence it must be separately ported to other

OSes entirely. This is cumbersome and must be done with each update.

2) APIs are not provided across platforms, making it difficult to run applications on other OSes.
3) Each OS has a binary format for its applications. This includes header layouts, instructions and variables, and their locations in the file.
4) CPUs have varying instruction sets. Applications with the appropriate instruction sets will run properly.
5) System calls used by applications in one OS may not be present in another OS.
6) UNIX systems have adopted ELF formats to help this, but ELF is not tied to any CPU architecture.
7) **Application Binary Interface** (ABI) defines how different components of binary code can interface for a given OS and CPU architecture.
8) ABIs specify low-level details such as address width, methods of passing arguments to functions, size of data types, etc.
9) An ABI is specified for a given architecture and OS, meaning it does little for cross-platform compatability.

7 Operating System Design and Inplementation

7.1 Design Goals

1) These can be grouped as **user goals** and **system goals**.
2) Users want ease of use, ease of learning, reliability, speed and safety.
3) Developers want flexible and easy implementation which is error free and reliable.
4) There is no unique solution for defining the goals and they vary with the OS. This is a creative task.
5) The field of **software engineering** develops general design principles for creating an OS.

7.2 Mechanisms and Policies

1) **Mechanisms** determine how to do something.
2) **Policies** determine what will be done.
3)