

Chapter 2: Operating System Structures

CONTENTS

1	Operating System Services	1	
2	User and Operating System Interface	2	
2.1	Command Interpreters	2	
2.2	Graphical User Interface . .	2	
2.3	Touch Screen Interface . . .	2	
2.4	Choice of Interface	2	
3	System Calls	2	
3.1	Application Programming Interface	2	
3.2	Types of System Calls	3	
3.2.1	Process Control .	3	
3.2.2	File Management	3	
3.2.3	Device Management	3	
3.2.4	Information Maintenance . . .	3	
3.2.5	Communication .	3	
3.2.6	Protection	3	
4	System Services	3	

1 OPERATING SYSTEM SERVICES

1) User Interface:

- Most OSes have a **user interface** (UI), most commonly a **graphical user interface** (GUI).
- In GUIs, the interface is a window system with a pointing device to click and a keyboard to enter twxt.
- Mobile phones and tables have a **touch-screen interface**, enabling selection and typing with fingers.
- A **command line interface** (CLI) users text commands only.
- Some systems may provide more than one of these variations.

- Program execution:** The system must be able to load and run a program. The program must terminate, either normally or abnormally, raising an error.

- 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:

- Programs need to be able to create/delete files, read/write files and search by name.
- OSes include permissions on management and ownership of files.
- Many OSes provide a variety of file systems offering specific features and performance characteristics.

5) Communications:

- Processes may need to exchange information with each other. These processes may be executing on the same computer or networked systems.
- Shared memory** involves processes reading and writing to a shared section of memory.
- Message parsing** involves packets of information in predefined formats that are moved by processes.

6) Error detection:

- OS must be able to detect and correct errors.
- 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).
- 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:

- Resources must be allocated to multiple processes running at the same time. OS may manage different types of resources.
- Some resources may have specific allocation code, while others may have a more general request code. There may be other routines for peripheral devices.
- 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 multiple command interpreters 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 pass 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 receiving 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)