OPERATING SYSTEMS

Chapter 1
Introduction

What Is An Operating System

A modern computer consists of:

- One or more processors
- Main memory
- Disks
- Printers
- Various input/output devices

Managing all these components requires a layer of software – the **operating system**

What Is An Operating System

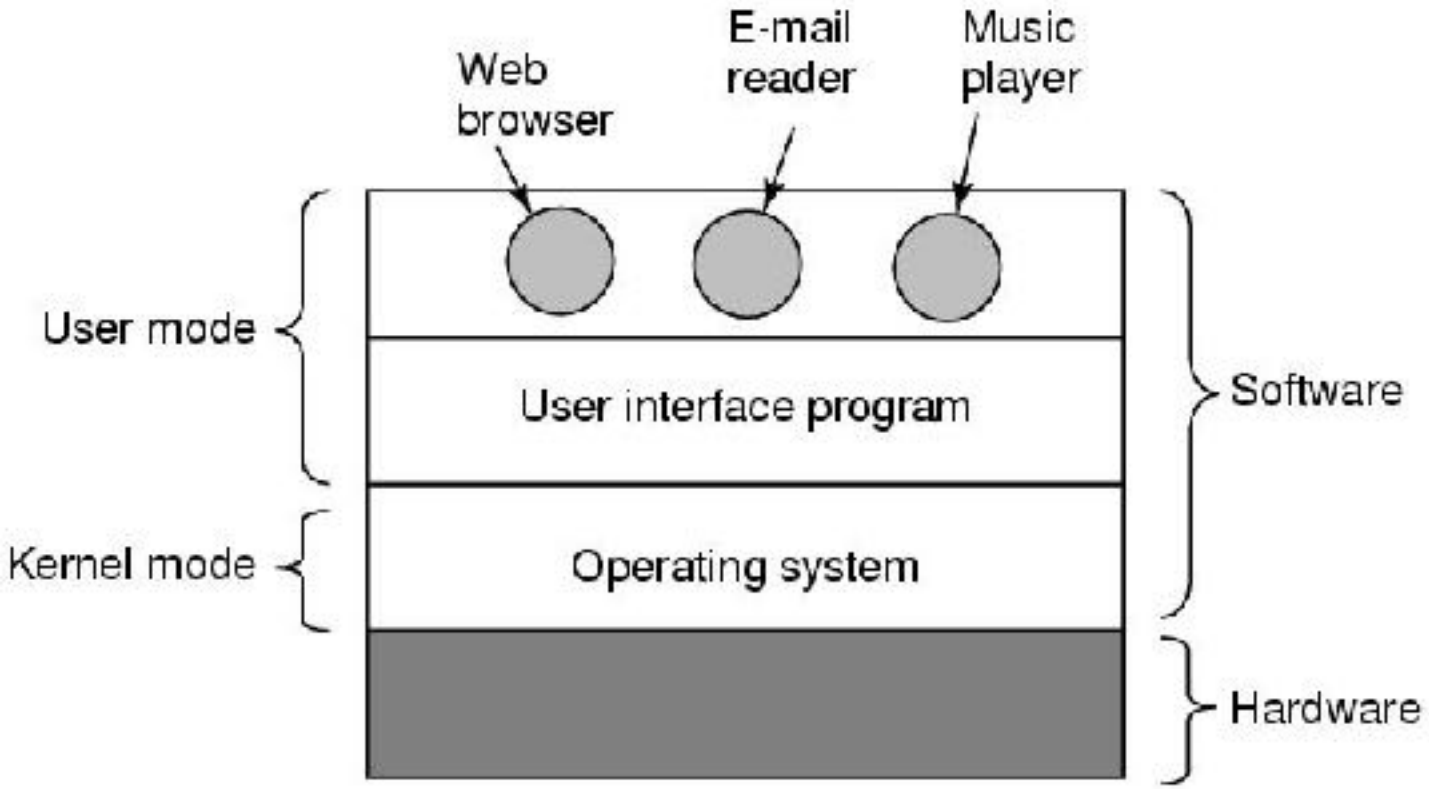


Figure 1. Where the operating system fits in.

What is an Operating System?

- On top of hardware is OS
- Most computers have two modes of operation:
  - Kernel mode and user mode
  - OS runs in **kernel mode,** which has complete access to all hardware and can execute any instruction
  - Rest of software runs in user mode, which has limited capability
  - Shell or GUI is the lowest level of user mode software

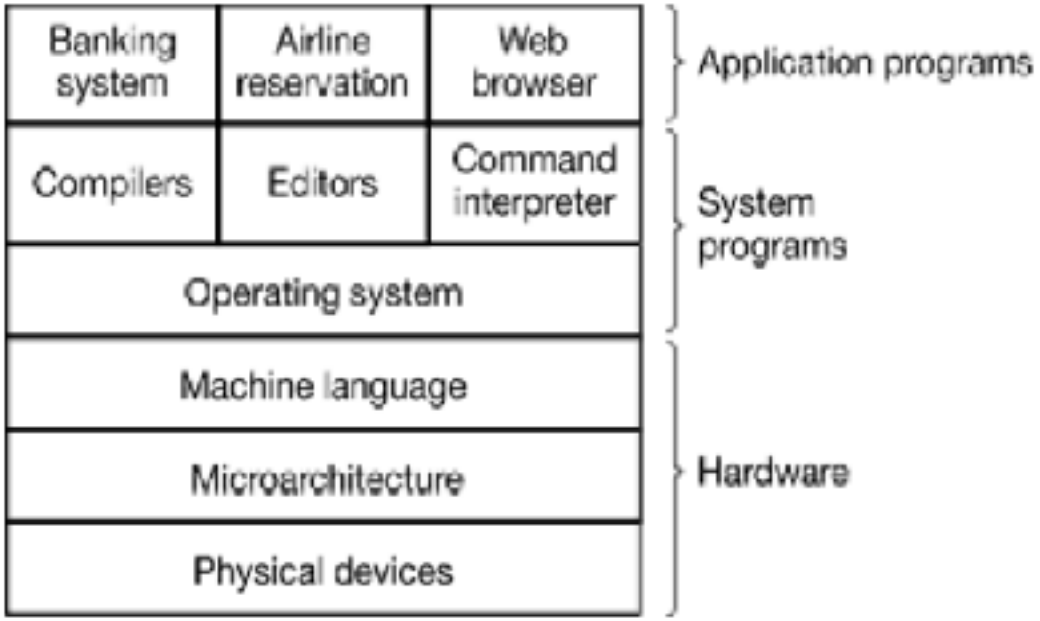


Figure 1-1. A computer system consists of hardware, system programs, and application programs.

What is kernel

Kernel is the core of the operating system.

It is the **first** program of operating system that is **loaded into the main memory** to start the working of the system.

Kernel basically translates the commands entered by the user in a way to make the computer understand that what has user requested.

Kernel acts as a **bridge** between application software and hardware of the system.

Kernel directly communicates with the hardware and let it know what the application software has requested. An operating system is unable to run without the kernel as it is the important program for the working of the system.

Kernel takes care of the **memory management**, **process management**, **task management** and **disk management**. Kernel checks out the memory space for the proper execution of the application program. It creates and destructs memory which helps in execution of the software.

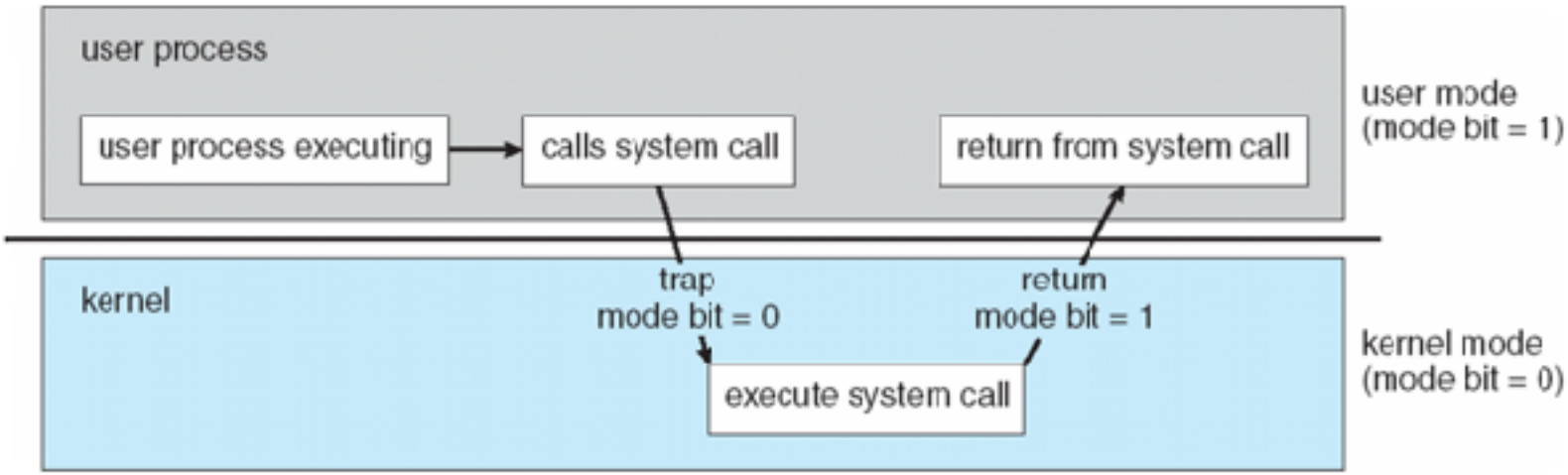
BASIS FOR COMPARISON	KERNEL	OPERATING SYSTEM
Basic	Kernel is an important part of the operating system.	Operating System is a system program.
Interface	Kernel is an interface between software and hardware of the computer.	Operating System is an interface between user and hardware of the computer.
Type	Monolithic kernels and Microkernels.	Single and Multiprogramming batch system, Distributed operating system, Realtime operating system.
Purpose	memory management, process management, task management, disk management.	In addition to the responsibilities of Kernel, Operating System is responsible for protection and security of the computer.

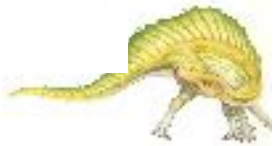
Operating-System Operations

- **Dual-mode** operation allows OS to protect itself and other system components
  - **User mode** and **kernel mode**
  - **Mode bit** provided by hardware
    - Provides ability to distinguish when system is running user code or kernel code
    - Some instructions designated as **privileged**, only executable in kernel mode
    - System call changes mode to kernel, return from call resets it to user

Transition from User to Kernel Mode

- Timer to prevent infinite loop / process hogging resources
  - Timer is set to interrupt the computer after some time period
  - Keep a counter that is decremented by the physical clock.
  - Operating system set the counter (privileged instruction)
  - When counter zero generate an interrupt
  - Set up before scheduling process to regain control or terminate program that exceeds allotted time





What is an operating system?

- Two functions:
  - From top to down: provide application programmers a clean abstract set of resources instead of hardware ones
  - From down to top: Manage these hardware resources

The Operating System as an Extended Machine

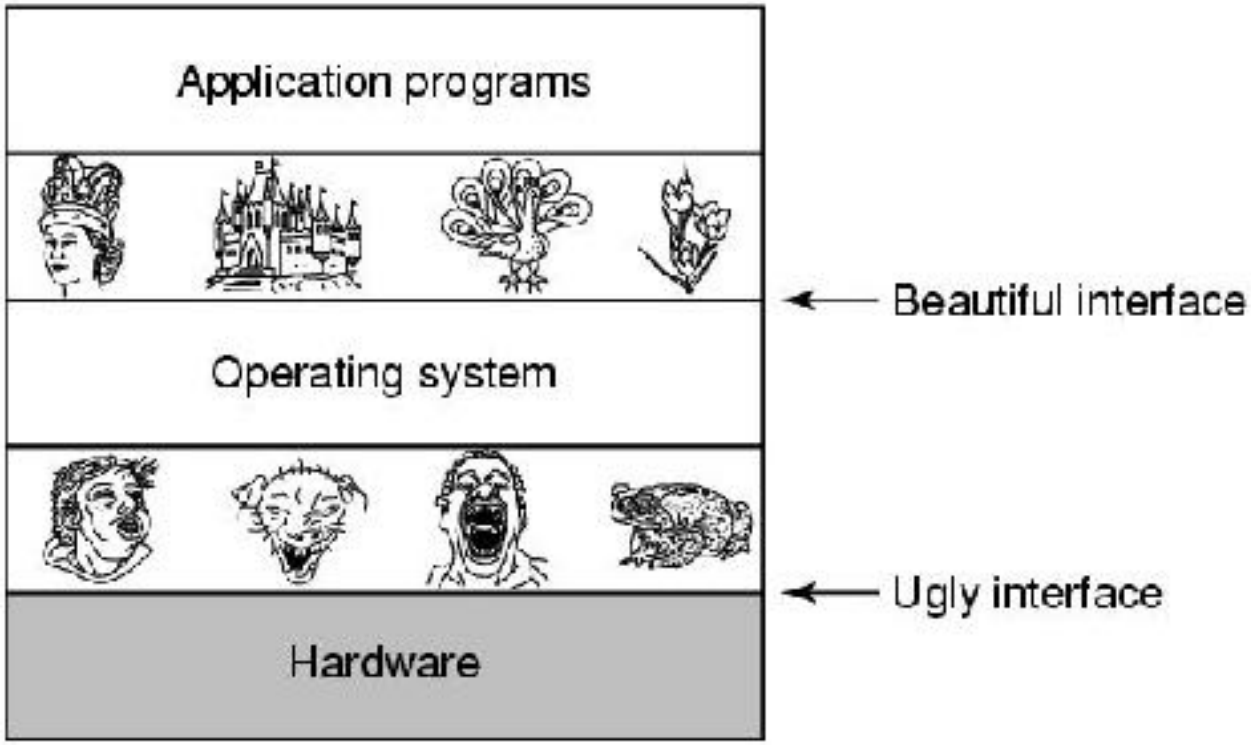


Figure 1-2. Operating systems turn ugly hardware into beautiful abstractions.

The Operating System as a Resource Manager

- Allow multiple programs to run at the same time
- For multiple users: Manage and protect memory, I/O devices, and other resources
- Includes multiplexing (sharing) resources in two different ways:
  - In time
  - In space

The Operating System as a Resource Manager

- When a resource is **time multiplexed**, different programs or users take turns using it. First one of them gets to use the resource, then another, and so on.
- For example, with only one CPU and multiple programs that want to run on it, the operating system first allocates the CPU to one program, then after it has run long enough, another one gets to use the CPU, then another, and then eventually the first one again.

Time mux…cont….

- Determining how the resource is time multiplexed who goes next and for how long is the task of--→ the operating system.

- Example of time multiplexing is sharing the printer. When multiple print jobs are queued up for printing on a single printer, a decision has to be made about which one is to be printed next.

Space Multiplexing

- The other kind of multiplexing is **space multiplexing**, instead of the customers taking turns, each one gets part of the resource.
- For example, main memory is normally divided up among several running programs, so each one can be **resident at the same time.**
- This raises issues of fairness, protection, and so on, and it is up to the operating system to solve them.

Evolution of Operating System
The First Generation (1945-55) Vacuum Tub

- The first ones used mechanical relays but were very slow, with cycle times measured in seconds. Relays were later replaced by vacuum tubes.
- These machines were filling up entire rooms with tens of thousands of vacuum tubes, but they were still millions of times slower than even the cheapest personal computers available today.

The Second Generation (1955-65) Transistors and Batch Systems

- The introduction of the transistor in the mid-1950s changed the picture radically.

- These machines, now called **mainframes,** were locked away in specially air conditioned computer rooms

- Given the high cost of the equipment, people quickly looked for ways to reduce the wasted time. The solution was the **batch system.**

- The idea behind it was to collect a tray full of jobs in the input room and then read them onto a magnetic tape using a small (relatively) inexpensive computer

- such as the IBM 1401, which was very good at reading cards, copying tapes, and printing output, but not at all good at numerical calculations. Other, **much more expensive machines, such as the IBM 7094, were used for the real computing.**

The Third Generation (1965-1980) ICs and Multiprogramming

- The 360 was the first **major computer line to use (small-scale) Integrated Circuits (ICs), thus providing a major price/performance advantage over the second-generation machines**

- **were built up from individual transistors.**

- They popularized several key techniques absent in second-generation operating systems. Probably the most important of these was **multiprogramming.**

- Having multiple jobs safely in memory at once requires special hardware to protect each job against mischief by the other ones, but the 360 and other third-generation systems were equipped with this hardware.

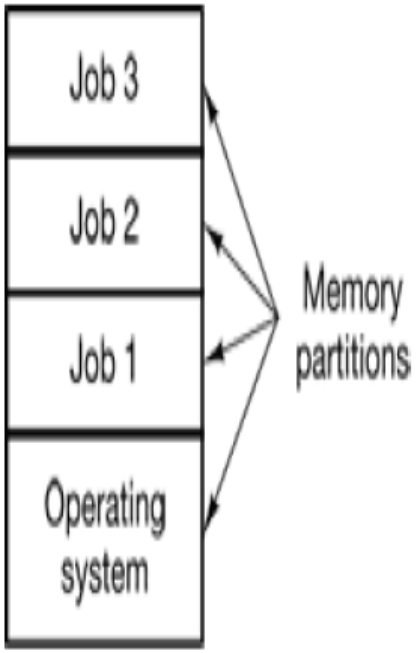


Figure 1-4. A multiprogramming system with three jobs in memory.

The Fourth Generation (1980-Present) Personal Computers

- With the development of **LSI (Large Scale Integration) circuits,** chips containing thousands of transistors on a square centimetre of silicon.

- Where the minicomputer made it possible for a department in a company or university to have its own computer, the microprocessor chip made it possible for a single individual to have his or her own personal computer.
- Engelbart invented the **GUI (Graphical User Interface),** complete with windows, icons, menus, and mouse.

What is Multiprogramming, Multiprocessing and Multitasking.

Multiprogramming:

•In a multiprogramming system there are one or more programs loaded in main memory which are ready to execute.

•Only one program at a time is able to get the CPU for executing its instructions (i.e., there is at most one process running on the system) while all the others are waiting their turn.

- Multiprocessing sometimes refers to executing multiple processes (programs) at the same time.

- In fact, **multiprocessing refers to the *hardware* (i.e., the CPU units) rather than the *software* (i.e., running processes)**. If the underlying hardware provides more than one processor then that is multiprocessing.

Multitasking

- Multitasking refers to having multiple (programs, processes, tasks, threads) running at the same time.

- The illusion of parallelism is achieved when the CPU is reassigned to another task (i.e. *process* or *thread context switching*).

THE OPERATING SYSTEM ZOO:-

→Mainframe Operating Systems
→Server Operating Systems
→Multiprocessor Operating Systems
→Personal Computer Operating Systems
→Real-Time Operating Systems
→Embedded Operating Systems
→Smart Card Operating Systems

Mainframe Operating Systems

- These computers distinguish themselves from personal computers in terms of their I/O capacity.
- Mainframes are also making something of a comeback as high-end Web servers, servers for large-scale electronic commerce sites, and servers for business-to-business transactions

- Three types of services:
  - Batch processing
  - Transaction processing
  - Time sharing

Server Operating Systems

- They run on servers, which are either very large personal computers, workstations, or even mainframes.
- They serve multiple users at once over a network and allow the users to share hardware and software resources.
- Servers can provide print service, file service, or web service. Internet providers run many server machines to support their customers and Web sites use servers to store the Web pages and handle the incoming requests.
- Typical server operating systems are UNIX and Windows 2000. Linux is also gaining ground for servers.

Multiprocessor Operating Systems

- An increasingly common way to get major-league computing power is to connect multiple CPUs into a single system.

- These systems are called parallel computers, multicomputer, or multiprocessors.

Personal Computer Operating Systems

- Their job is to provide a good interface to a single user.
- They are widely used for word processing, spreadsheets, and Internet access.
- Common examples are Windows 98, Windows 2000, the Macintosh operating system, and Linux.

Real-Time Operating Systems

- These systems are characterized by having time as a key parameter.
- For example, in industrial process control systems, real-time computers have to collect data about the production process and use it to control machines in the factory.

- Often there are hard deadlines that must be met. For example, if a car is moving down an assembly line, certain actions must take place at certain instants of time, if a welding robot welds too early or too late, the car will be ruined. If the action absolutely *must* occur at a certain moment (or within a certain range), we have a **hard real-time system.**

- Another kind of real-time system is a **soft real-time** system, in which missing an occasional deadline is acceptable. Digital audio or multimedia systems fall in this category.

Embedded Operating Systems

- A palmtop computer or **PDA (Personal Digital Assistant)** is a small computer that fits in a shirt pocket and performs a small number of functions such as an electronic address book and memo pad.
- Embedded systems run on the computers that control devices that are not generally thought of as computers, such as TV sets, microwave ovens, and mobile telephones.

- Examples of such operating systems are PalmOS and Windows CE (Consumer Electronics).

Smart Card Operating Systems

- The smallest operating systems run on smart cards, which are credit card-sized devices containing a CPU chip.

- They have very severe processing power and memory constraints. Some of them can handle only a single function, such as electronic payments, but others can handle multiple functions on the same smart card.

Computer Hardware Review

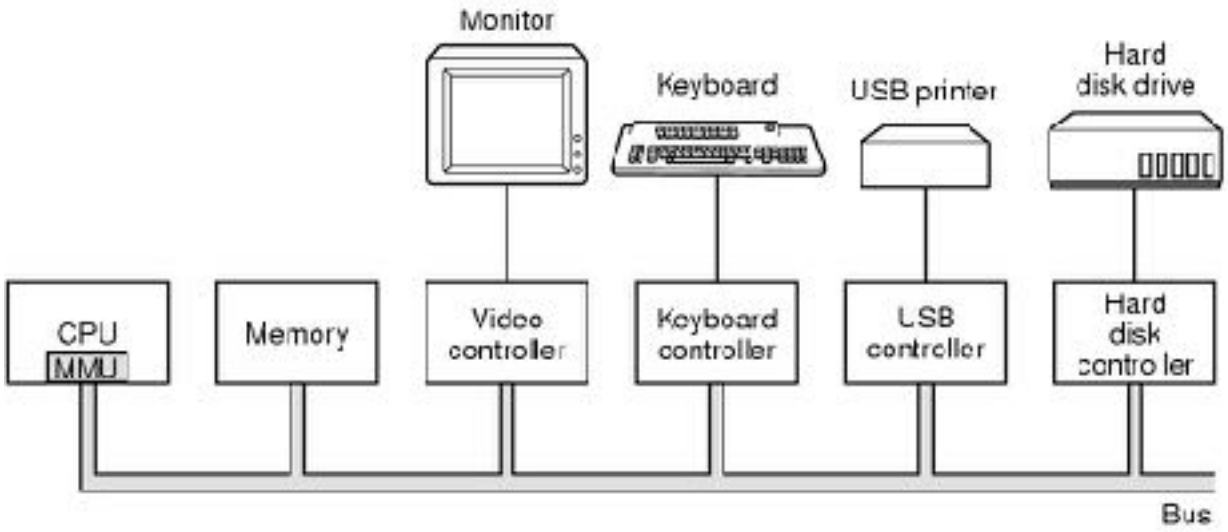


Figure 1-6. Some of the components of a simple personal computer.

Memory

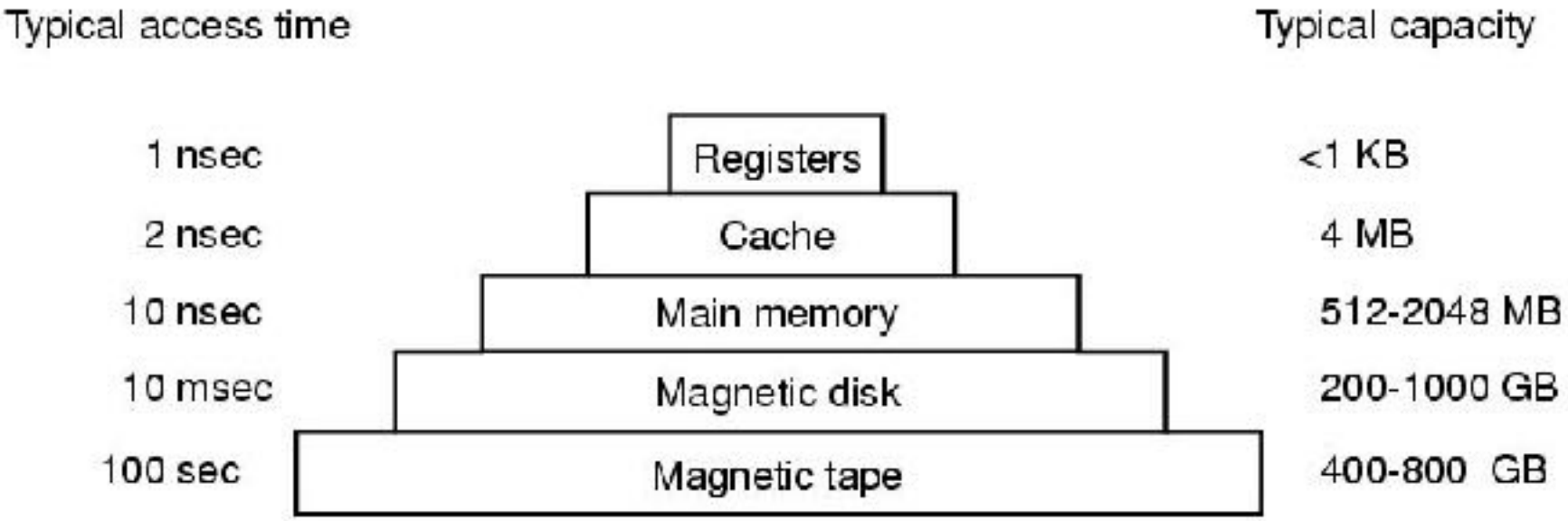


Figure 1-9. A typical memory hierarchy.

I/O Devices

- Controller runs a device-accepts commands from the OS and executes them
- Complicated business
  - Eg. Disk controller gets command to read sector x on disk y. Must convert to (cylinder, sector, head) address, move arm to correct cylinder, wait for sector to rotate under the head, read and store bits coming off the drive, compute checksum, store bits as words in memory
  - Controller contains a small embedded computer to run its device

Device Driver

- OS software that talks to controller-gives commands, accepts responses
- Each controller manufacturer supplies a driver for each OS
- Driver runs in kernel mode
- Controller has registers which are used to communicate with the driver
- Three modes of communication
  - Polling
  - Interrupts
  - DMA

Operating System Concepts

- Processes
- Address spaces
- Files
- Input/Output
- Protection
- The shell
- others

System calls

- The goal of interface between user programs and the operating system is primarily about dealing with the abstractions.

- System calls is the interface users contact with OS and hardware

- **Example of read system call**

It has three parameters: the first one specifying the file, the second one pointing to the buffer, and the third one giving the number of bytes to read. Call from C program will look like

count = read(fd, buffer, nbytes);

The system call (and the library procedure) return the number of bytes actually read in count

- If the system call cannot be carried out owing to an invalid parameter or a disk error, count is set to −1

Steps

- Step 1-3: The first and third parameters are called by value, but the second parameter is passed by reference, meaning that the address of the buffer (indicated by &) is passed, not the contents of the buffer.
- Step 4 :Then comes the actual call to the library procedure. This instruction is the normal procedure-call instruction used to call all procedures

- Step 5 : library procedure puts the system-call number in a register
- Step 6: Then it executes a TRAP instruction to switch from user mode to kernel mode and start execution at a fixed address within the kernel
- Step 7: The kernel code that starts following the TRAP examines the system-call number and then dispatches to the correct system-call handler, usually via a table of pointers to system-call handlers indexed on system-call number

- Step 8: the system-call handler runs
- Step 9: Once it has completed its work, control may be returned to the user-space library procedure at the instruction following the TRAP instruction
- Step 10: This procedure returns to the user program in the usual way procedure calls return
- Step 11: To finish the job, the user program has to clean up the stack, as it does after any procedure call

System Calls

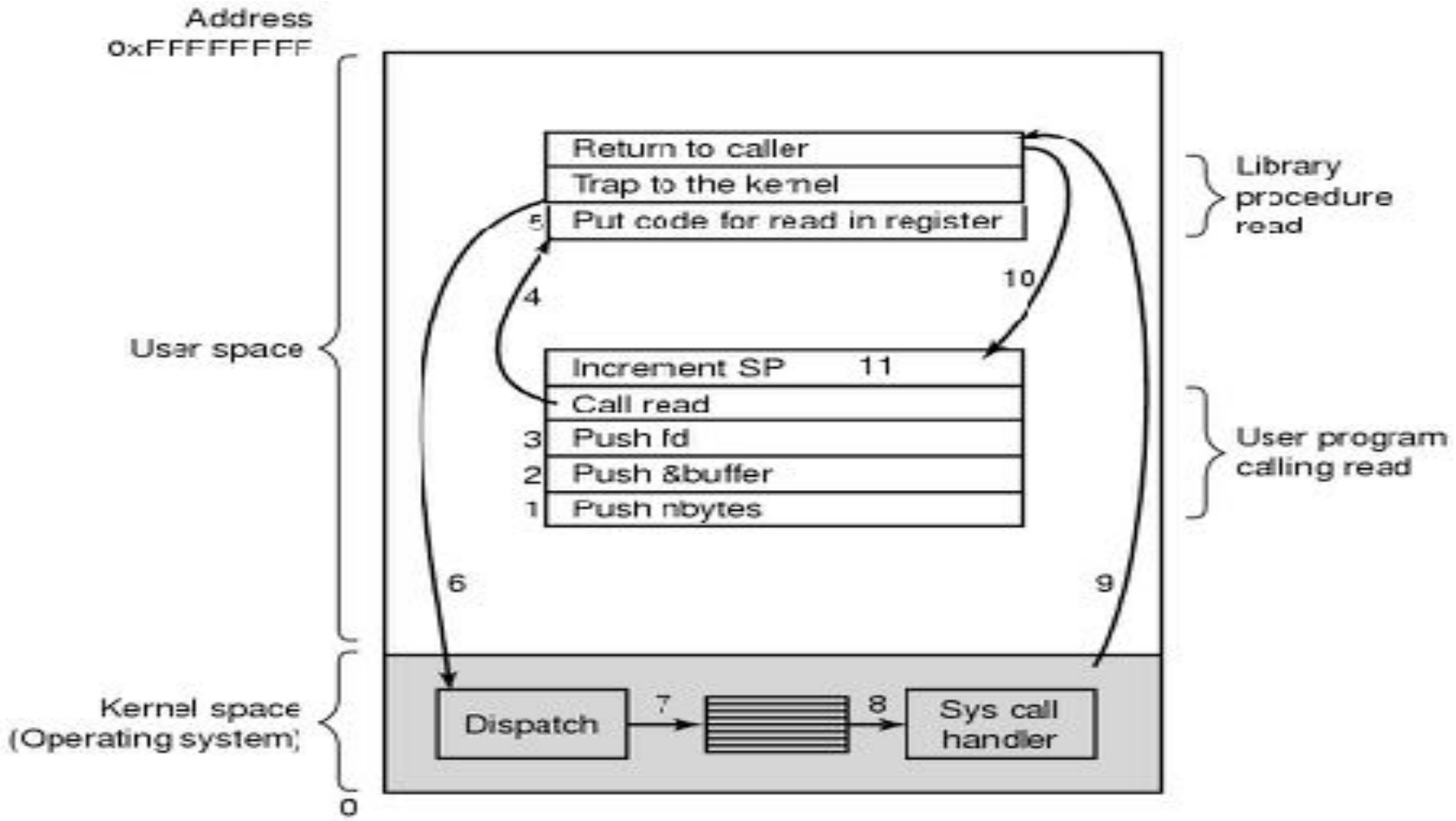


Figure 1-17. The 11 steps in making the system call read(fd, buffer, nbytes).

System Calls for Process Management

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

Figure 1-18. Some of the major POSIX system calls.

Fork System call

- Fork is used to create a new process. It creates an exact duplicate of the original process
- The fork call returns a value, which is zero in the child and equal to the child's PID (Process IDentifier) in the parent.
- After creation of process both parent and child starts execution from next instruction.

```
Main()
{
  fork();
  fork();
printf("hi");
}
```

System Calls for File Management (1)

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing, or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Figure 1-18. Some of the major POSIX(The Portable Operating System Interface) system calls.

System Calls for File Management (2)

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

Figure 1-18. Some of the major POSIX system calls.

Miscellaneous System Calls

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

Figure 1-18. Some of the major POSIX system calls.

Operating Systems Structure

Operating system from inside has following types:

- Monolithic systems
- Layered systems
- Microkernels

Operating Systems Structure

Monolithic systems – basic structure:

- The operating system is written as a collection of procedures, linked together into a single large executable binary program.
- Each procedure in the system is free to call any other one. A main program that invokes the requested service procedure.

- In terms of information hiding, there is essentially none—every procedure is visible to every other procedure.

- Service is provided using system call.

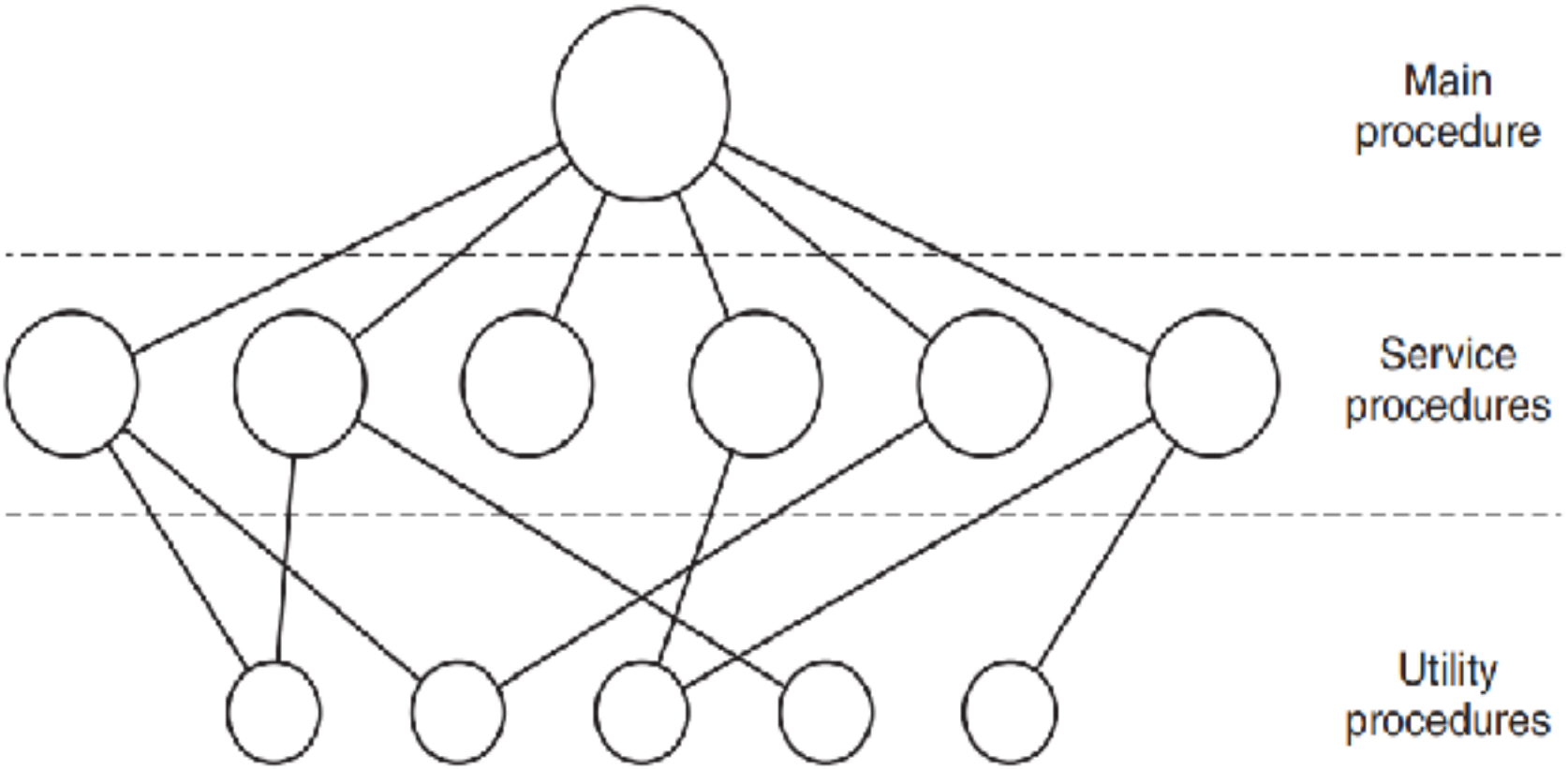
- A crash in any of the procedures will take down the entire operating system

- Example : Unix , DOS

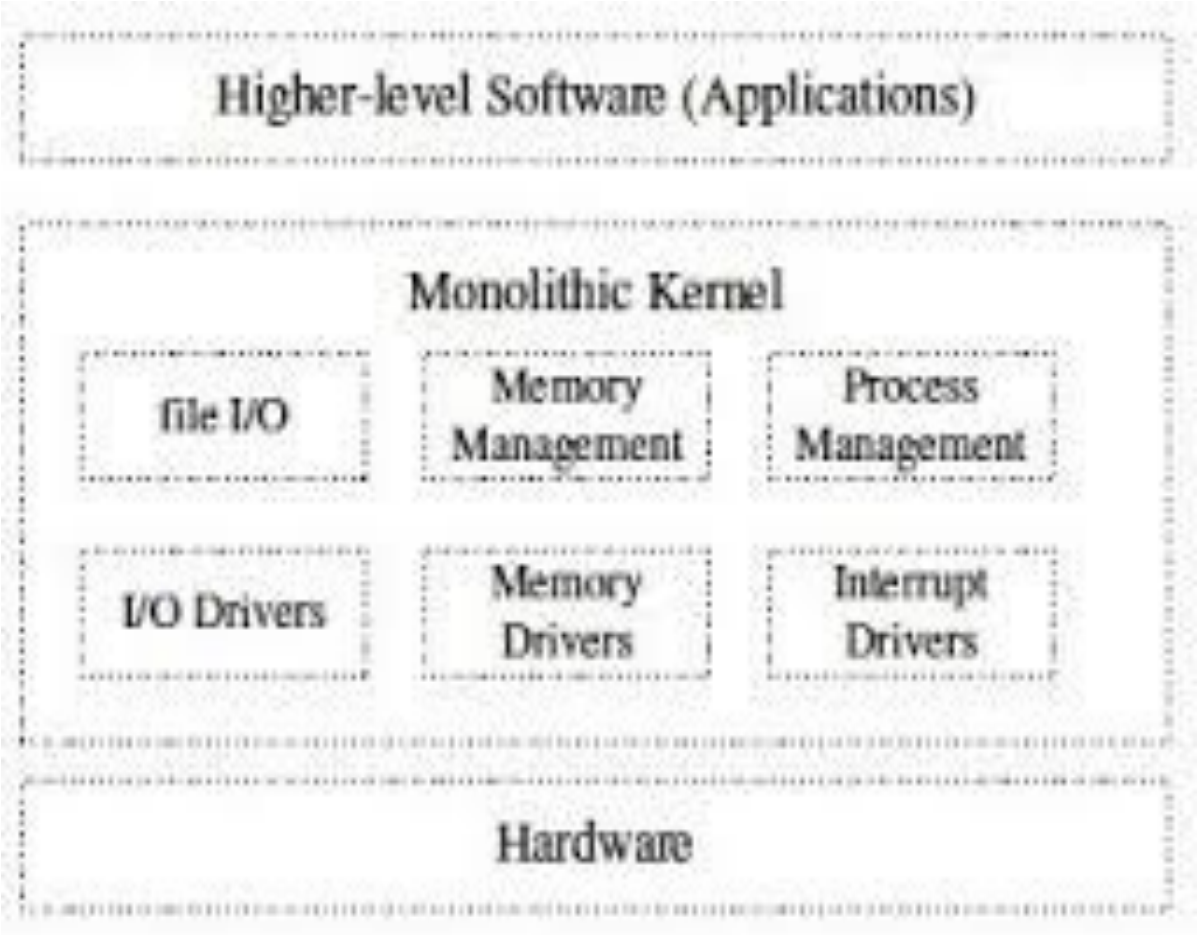
.

Basic structure for the operating system:

- 1. A main program that invokes the requested service procedure.
- 2. A set of service procedures that carry out the system calls.
- 3. A set of utility procedures that help the service procedures



A simple structuring model for a monolithic system.


Higher-level Software (Applications)
Monolithic Kernel
file I/O
Memory Management
Process Management
I/O Drivers
Memory Drivers
Interrupt Drivers
Hardware

Layered Systems

- Organize the operating system as a hierarchy of layers, each one constructed upon the one below.
- Layer 0 deals with allocation of the processor, switching between processes when interrupts occurred.
- Above layer 0, the system consisted of sequential processes, each of which could be programmed without having to worry about the fact that multiple processes were running on a single processor

Layered Systems

- Layer 1 handles memory management.
- Layer 2 handled communication between each process and the operator console (that is, the user).
- Layer 3 took care of managing the I/O devices and buffering the information streams to and from them.

Layered Systems

Layer 4 was where the user programs were found.
Example: THE system

Layered Systems

Layer	Function
5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

Figure 1-25. Structure of the THE operating system.

Microkernel

- The basic idea behind the microkernel design is to achieve high reliability by splitting the operating system up into small, well-defined modules, only one of which—the microkernel—runs in kernel mode and the rest run as relatively powerless ordinary user processes.
- Example : MINIX3

- In Minix 3 outside the kernel, the system is structured as three layers of processes all running in user mode. The lowest layer contains the device drivers.

- Since they run in user mode, they do not have physical access to the I/O port space and cannot issue I/O commands directly.

- Above the drivers is another user-mode layer containing the servers. One or more file servers manage the file system(s), the process manager creates, destroys, and manages processes

- User programs obtain operating system services by sending short messages to the servers asking for the POSIX system calls.

- One interesting server is the **reincarnation server**, whose job is to check if the other servers and drivers are functioning correctly.

- The system has many restrictions limiting the power of each process.
- drivers can touch only authorized I/O ports

Microkernels

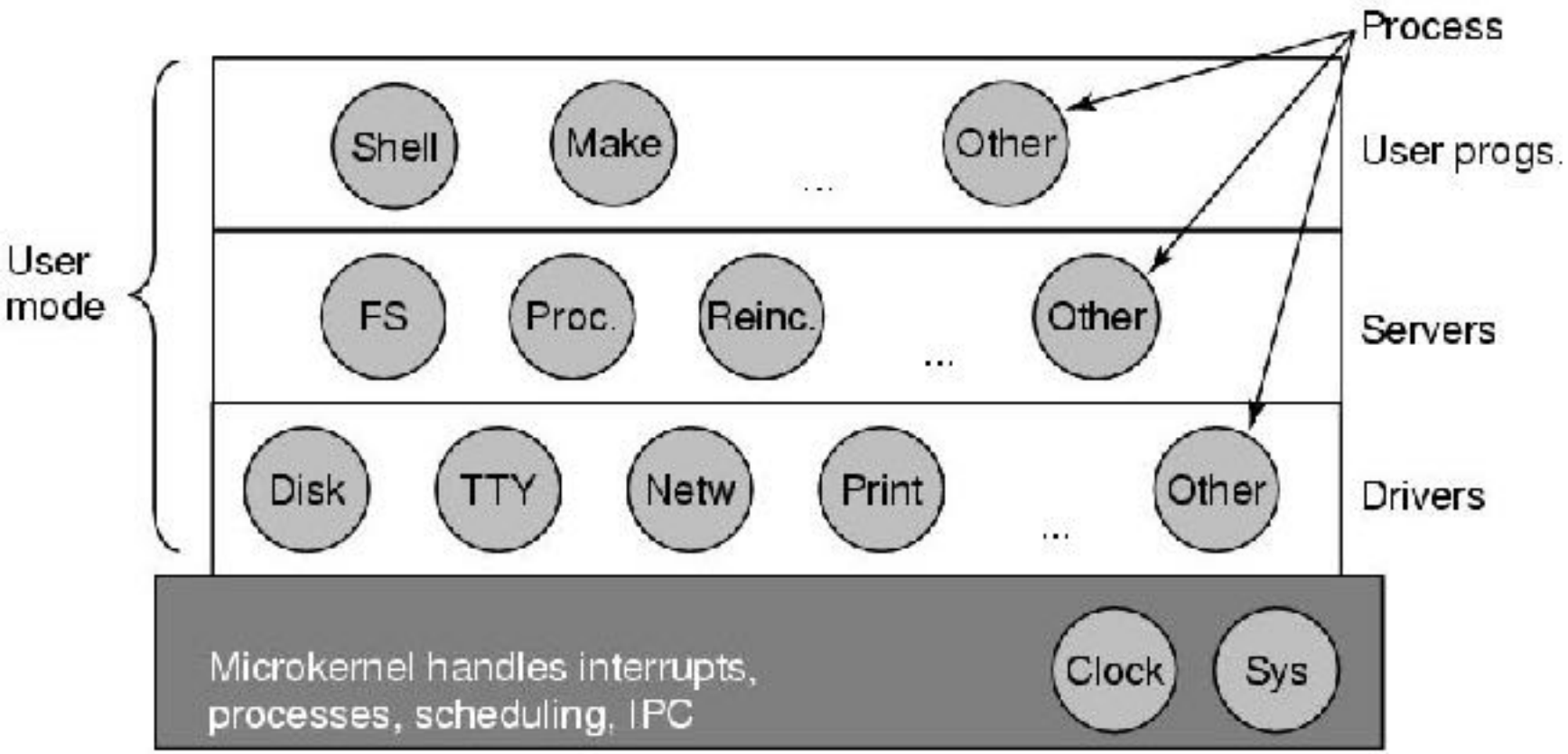
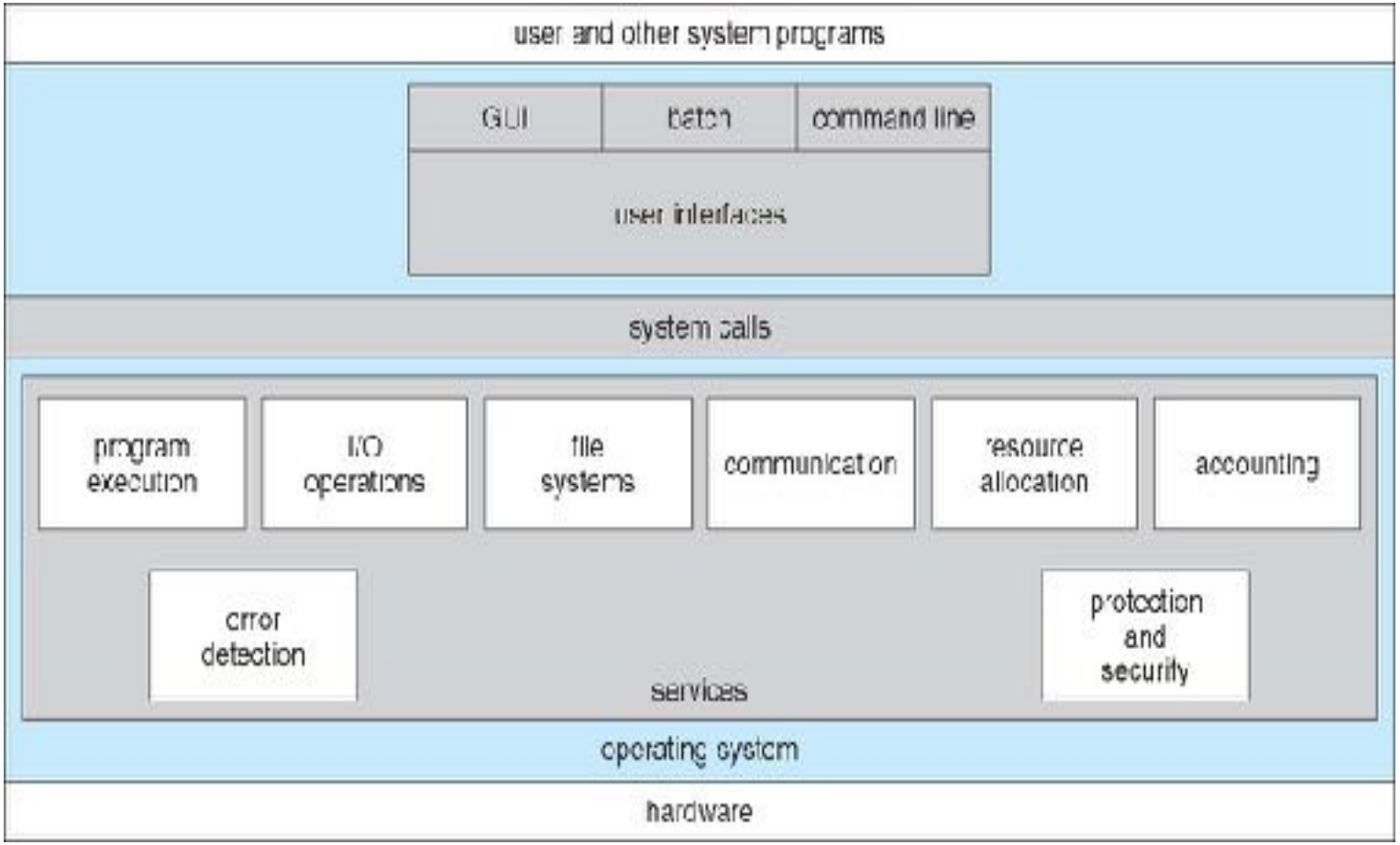


Figure 1-26. Structure of the MINIX 3 system.

Exokernel

- Rather than cloning the actual machine, as is done with virtual machines, another strategy is partitioning it, giving each user a subset of the resources.
- At the bottom layer, running in kernel mode, is a program called the exokernel.
- Its job is to allocate resources to virtual machines and then check attempts to use them to make sure no machine is trying to use somebody else's resources. Example: VM/360

A View of Operating System Services



Operating System Services (Cont)

- One set of operating-system services provides functions that are helpful to the user (Cont):
  - Communications – Processes may exchange information, on the same computer or between computers over a network
  - Error detection – OS needs to be constantly aware of possible errors
    - For each type of error, OS should take the appropriate action to ensure correct and consistent computing

Operating System Services (Cont)

- **Resource allocation -** When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
- **Accounting -** To keep track of which users use how much and what kinds of computer resources
- **Protection and security -** The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other
  - **Protection** involves ensuring that all access to system resources is controlled
  - **Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts