Computing Environments

- Mobile Computing
- Distributed Systems
- Client Server Computing
- Peer to Peer Computing
- Virtualisation
- Cloud Computing
- Real Time Embedded Systems

Mobile Computing

- Mobile computing refers to computing on handheld smartphones and tablet computers. These devices share the distinguishing physical features of being portable and lightweight.
- Historically, compared with desktop and laptop computers, mobile systems gave up screen size, memory capacity, and overall functionality in return for handheld mobile access to services such as e-mail and web browsing.
- The memory capacity and processing speed of mobile devices, however, are more limited than those of PCs. Whereas a smartphone or tablet may have 64 GB in storage, it is not uncommon to find 1 TB in storage on a desktop computer.
- Similarly, because power consumption is such a concern, mobile devices often use processors that are smaller, are slower, and offer fewer processing cores than processors found on traditional desktop and laptop computers.
- Two operating systems currently dominate mobile computing: Apple iOS and Google Android.

Distributed Systems

- A distributed system is a collection of physically separate, possibly heterogeneous, computer systems that are networked to provide users with access to the various resources that the system maintains.
- Access to a shared resource increases computation speed, functionality, data availability, and reliability.
- A network, in the simplest terms, is a communication path between two or more systems. Distributed systems depend on networking for their functionality.
- A network operating system is an operating system that provides features such as file sharing across the network, along with a communication scheme that allows different processes on different computers to exchange messages.

Client Server Computing

- Many of today’s systems act as server systems to satisfy requests generated by client systems. This form of specialized distributed system, is called a client–server system.
- Server systems can be broadly categorized as compute servers and file servers:

1. The **compute-server** system provides an interface to which a client can send a request to perform an action (for example, read data). In response, the server executes the action and sends the results to the client. A server running a database that responds to client requests for data is an example of such a system.

2. The **file-server** system provides a file-system interface where clients can create, update, read, and delete files. An example of such a system is a web server that delivers files to clients running web browsers.

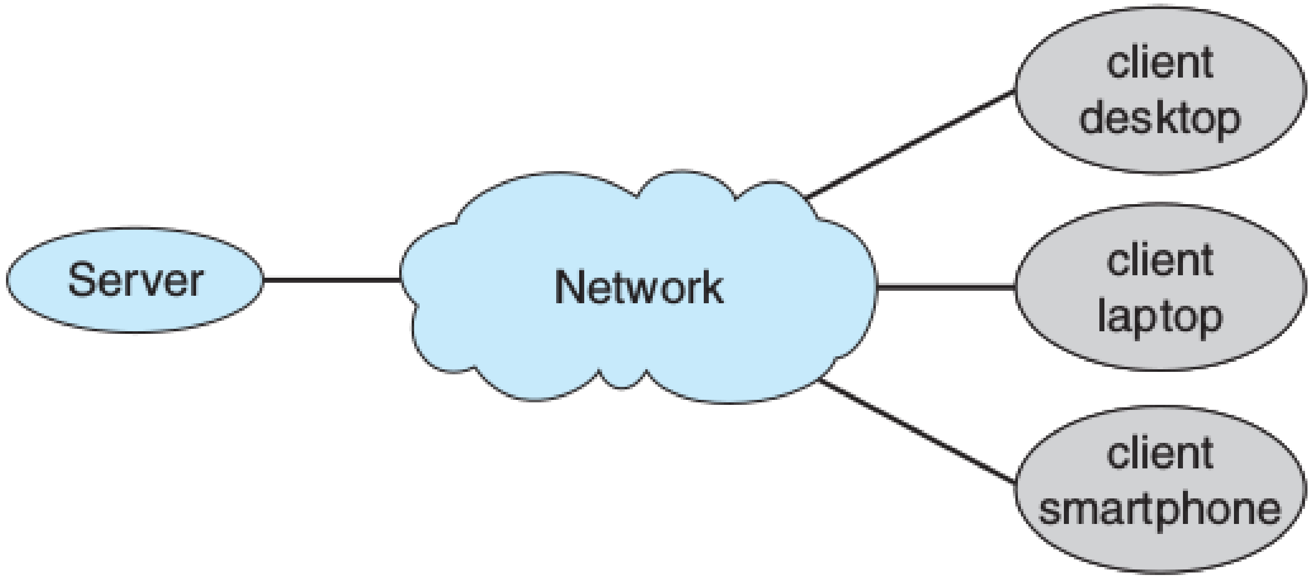


Figure 1.18 General structure of a client–server system.

Peer to Peer Computing

- In this model, clients and servers are not distinguished from one another. Instead, all nodes within the system are considered peers, and each may act as either a client or a server, depending on whether it is requesting or providing a service.
- To participate in a peer-to-peer system, a node must first join the network of peers. Once a node has joined the network, it can begin providing services to—and requesting services from—other nodes in the network.

1. When a node joins a network, it registers its service with a centralized lookup service on the network. Any node desiring a specific service first contacts this centralized lookup service to determine which node provides the service. The remainder of the communication takes place between the client and the service provider.

2. An alternative scheme uses no centralized lookup service. Instead, a peer acting as a client must discover what node provides a desired service by broadcasting a request for the service to all other nodes in the network. The node (or nodes) providing that service responds to the peer making the request.

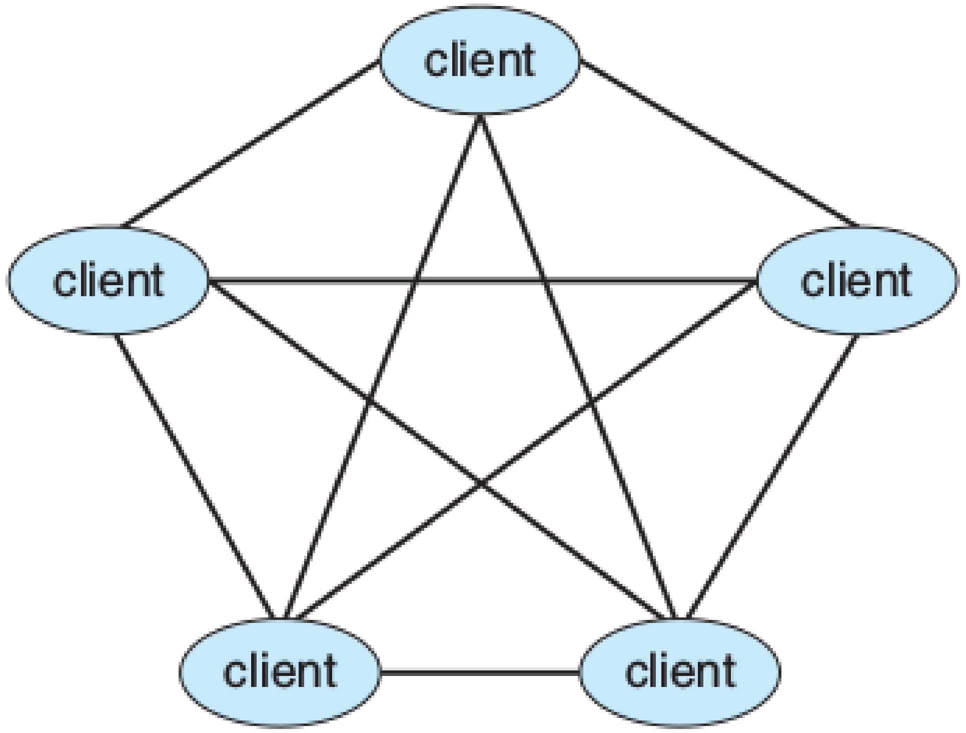


Figure 1.19 Peer-to-peer system with no centralized service.

Virtualization

- Virtualization is a technology that allows operating systems to run as applications within other operating systems.
- With virtualization, in contrast, an operating system that is natively compiled for a particular CPU architecture runs within another operating system also native to that CPU.
- Windows was the host operating system, and the VM ware application was the virtual machine manager VMM . The VMM runs the guest operating systems, manages their resource use, and protects each guest from the others.

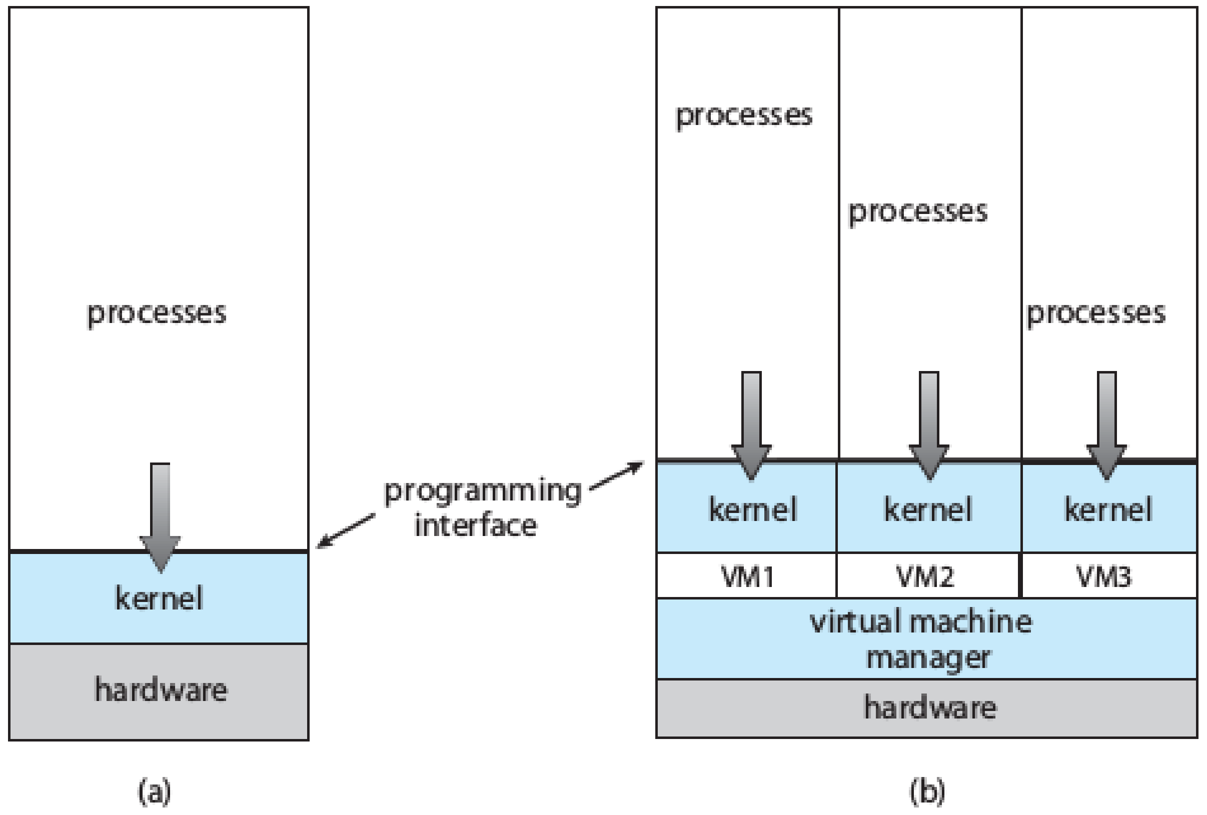


Figure 1.20 VMware.

Cloud Computing

- Cloud computing is a type of computing that delivers computing, storage, and even applications as a service across a network. In some ways, it's a logical extension of virtualization, because it uses virtualization as a base for its functionality.
- There are actually many types of cloud computing, including the following:

1. **Public cloud** —a cloud available via the Internet to anyone willing to pay for the services
2. **Hybrid cloud** —a cloud that includes both public and private cloud components
3. **Software as a service (SaaS)**—one or more applications (such as word processors or spreadsheets) available via the Internet
4. **Platform as a service (PaaS)**—a software stack ready for application use via the Internet (for example, a database server)
5. **Infrastructure as a service (IaaS)**—servers or storage available over the Internet (for example, storage available for making backup copies of production data)

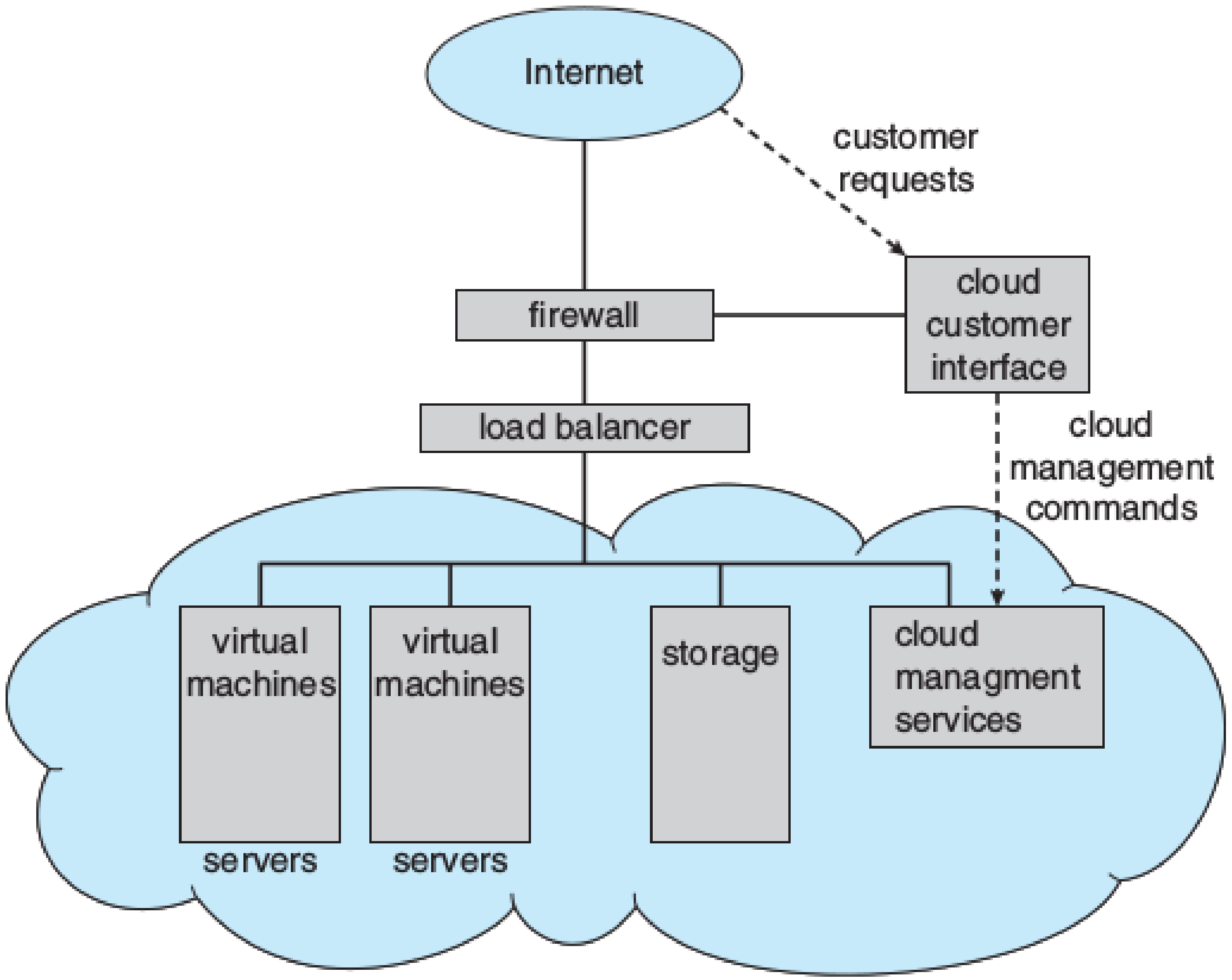


Figure 1.21 Cloud computing.

Real Time Embedded Systems

- They tend to have very specific tasks. The systems they run on are usually primitive, and so the operating systems provide limited features.
- Usually, they have little or no user interface, preferring to spend their time monitoring and managing hardware devices, such as automobile engines and robotic arms.
- Embedded systems almost always run real-time operating systems. A real-time system has well-defined, fixed time constraints. Processing must be done within the defined constraints, or the system will fail. For instance, it would not do for a robot arm to be instructed to halt after it had smashed into the car it was building.
- A real-time system functions correctly only if it returns the correct result within its time constraints. Contrast this system with a time-sharing system, where it is desirable (but not mandatory) to respond quickly, or a batch system, which may have no time constraints at all.

Operating-System Services

1. **User interface**. Almost all operating systems have a user interface (UI). This interface can take several forms. One is a **command-line interface (CLI)**, which uses text commands and a method for entering them (say, a keyboard). Another is a **batch interface**, in which commands and directives to control those commands are entered into files, and those files are executed. Most commonly, a **graphical user interface** (GUI) is used. Here, the interface is a window system with a pointing device to direct I/O , choose from menus, and make selections and a keyboard to enter text.

2. **Program execution**. The system must be able to load a program into memory and to run that program. The program must be able to end its execution, either normally or abnormally (indicating error).

3. **I/O operations**. A running program may require I/O , which may involve a file or an I/O device. For efficiency and protection, users usually cannot control I/O devices directly. Therefore, the operating system must provide a means to do I/O.

4. **File-system manipulation.** Programs need to read and write files and directories. They also need to create and delete them by name, search for a given file, and list file information. Finally, some operating systems include permissions management to allow or deny access to files or directories based on file ownership.

5. **Communications**. There are many circumstances in which one process needs to exchange information with another process. Such communication may occur between processes that are executing on the same computer or between processes that are executing on different computer systems tied together by a computer network. Communications may be implemented via **shared memory,** in which two or more processes read and write to a shared section of memory, or **message passing**, in which packets of information in predefined formats are moved between processes by the operating system.

6. **Error detection.** The operating system needs to be detecting and correcting errors constantly. Errors may occur in the CPU and memory hardware (such as a memory error or a power failure), in I/O devices (such as a parity error on disk, a connection failure on a network, or lack of paper in the printer), and in the user program (such as an arithmetic overflow, an attempt to access an illegal memory location, or a too-great use of CPU time). For each type of error, the operating system should take the appropriate action to ensure correct and consistent computing.

7. **Resource allocation**. When there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them. The operating system manages many different types of resources.

8. **Accounting.** We want to keep track of which users use how much and what kinds of computer resources. This record keeping may be used for accounting (so that users can be billed) or simply for accumulating usage statistics.

9. **Protection and security.** When several separate processes execute concurrently, it should not be possible for one process to interfere with the others or with the operating system itself. Protection involves ensuring that all access to system resources is controlled. Security of the system from outsiders is also important. Such security starts with requiring each user to authenticate himself or herself to the system, usually by means of a password, to gain access to system resources.

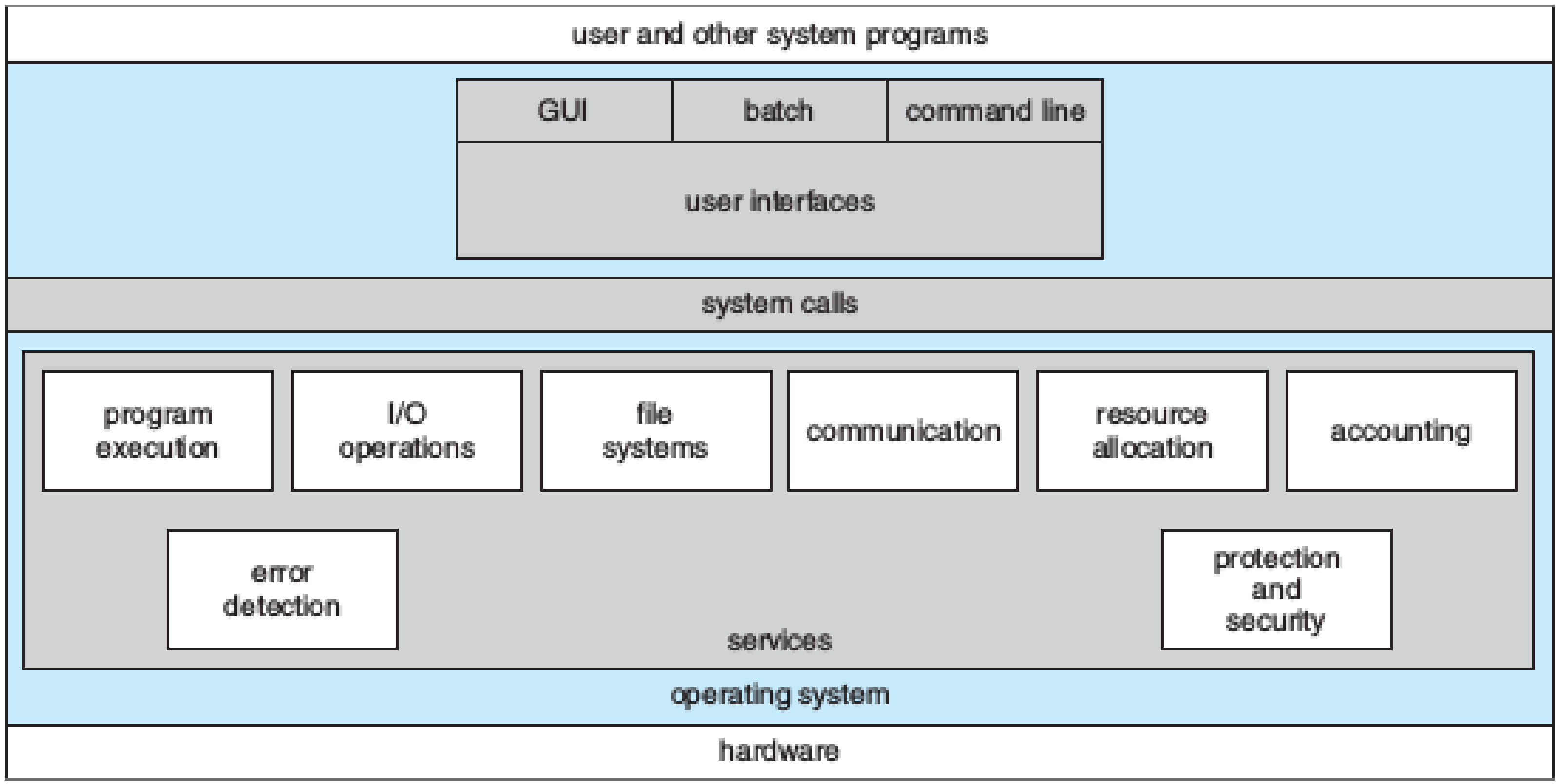
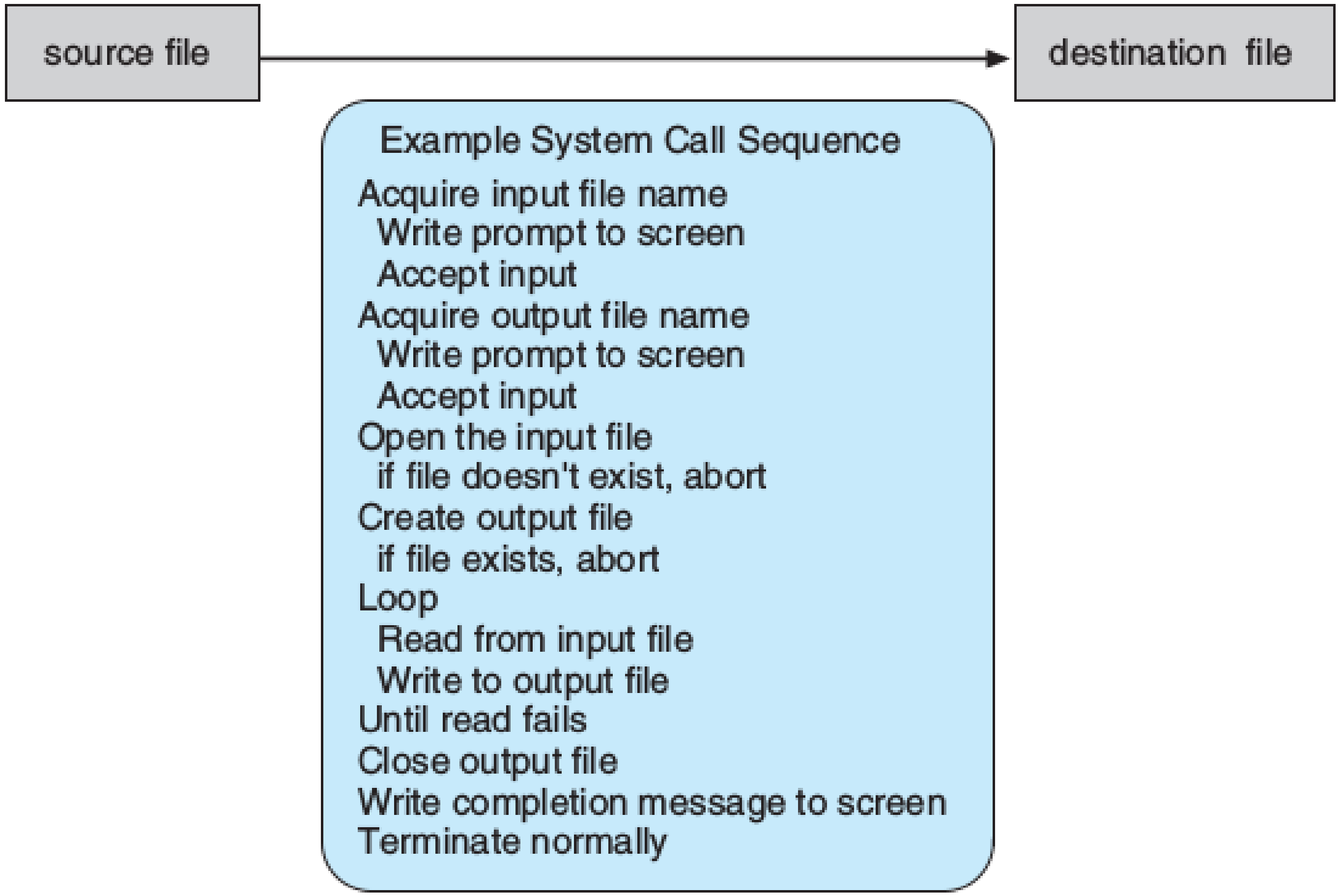


Figure 2.1 A view of operating system services.

System Calls

- System calls provide an interface to the services made available by an operating system.
- Even simple programs may make heavy use of the operating system. Frequently, systems execute thousands of system calls per second.



```
Acquire input file name
  Write prompt to screen
  Accept input
Acquire output file name
  Write prompt to screen
  Accept input
Open the input file
  if file doesn't exist, abort
Create output file
  if file exists, abort
Loop
  Read from input file
  Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally
```

Figure 2.5 Example of how system calls are used.

- An **application programming interface (API)** specifies a set of functions that are available to an application programmer, including the parameters that are passed to each function and the return values the programmer can expect.
- The run-time support system (a set of functions built into libraries included with a compiler) provides a **system-call interface** that serves as the link to system calls made available by the operating system.
- The system-call interface intercepts function calls in the API and invokes the necessary system calls within the operating system.
- Typically, a number is associated with each system call, and the system-call interface maintains a table indexed according to these numbers. The system call interface then invokes the intended system call in the operating-system kernel and returns the status of the system call and any return values.

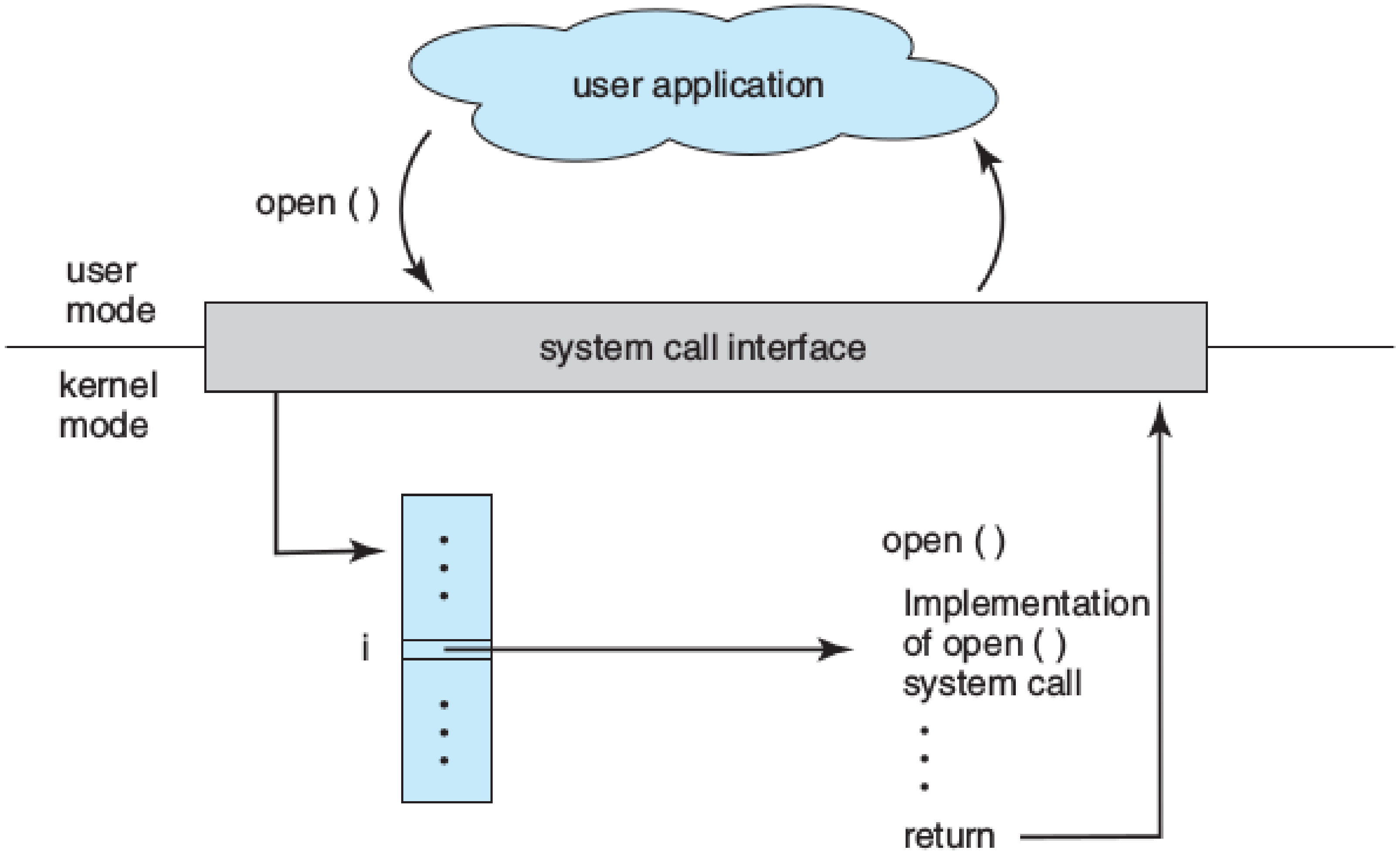


Figure 2.6 The handling of a user application invoking the `open()` system call.

- The caller need know nothing about how the system call is implemented or what it does during execution. Rather, the caller need only obey the API and understand what the operating system will do as a result of the execution of that system call.
- Three general methods are used to pass parameters to the operating system.

  1. The simplest approach is to pass the parameters in registers. In some cases, however, there may be more parameters than registers.

  2. In these cases, the parameters are generally stored in a block, or table, in memory, and the address of the block is passed as a parameter in a register.

  3. Parameters also can be placed, or pushed, onto the stack by the program and popped off the stack by the operating system.

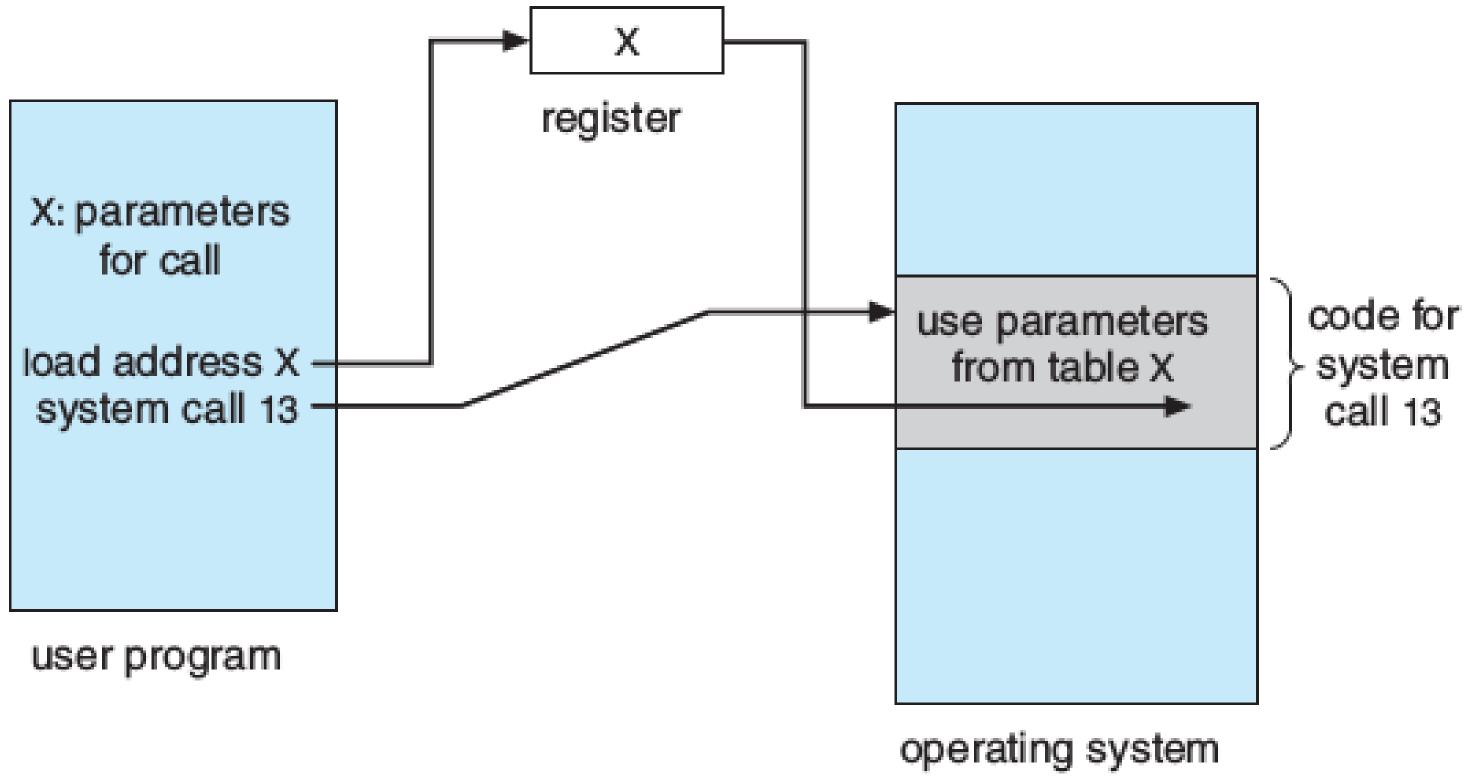


Figure 2.7 Passing of parameters as a table.

Types of System Calls

- System calls can be grouped roughly into six major categories: process control, file manipulation, device manipulation, information maintenance, communications, and protection.

1. **Process Control**

- A running program needs to be able to halt its execution either normally (**end()**) or abnormally (**abort()**).
- A process or job executing one program may want to **load()** and **execute()** another program.
- If both programs continue concurrently, we have created a new job or process (**create_process()**) to be multiprogrammed.
- The ability to determine and reset the attributes of a job or process, including the job’s priority, its maximum allowable execution time, and so on (**get_process_attributes() and set_process_attributes()**).
- Terminate a job or process that we created (**terminate_process()**) if we find that it is incorrect or is no longer needed.

- Having created new jobs or processes, we may need to wait for them to finish their execution. We may want to wait for a certain amount of time to pass (**wait_time()**). More probably, we will want to wait for a specific event to occur (**wait_event()**). The jobs or processes should then signal when that event has occurred (**signal_event()**).
- To ensure the integrity of the data being shared, operating systems often provide system calls allowing a process to lock shared data. Then, no other process can access the data until the lock is released. Typically, such system calls include **acquire_lock()** and **release_lock()** .

2. File Management

- We first need to be able to **create()** and **delete()** files. Once the file is created, we need to **open()** it and to use it. We may also **read()** , **write()** , or **reposition()** (rewind or skip to the end of the file, for example). Finally, we need to **close()** the file, indicating that we are no longer using it.
- File attributes include the file name, file type, protection codes, accounting information, and so on. At least two system calls, **get_file_attributes()** and **set file attributes()** , are required for this function.

3. Device Management

- The various resources controlled by the operating system can be thought of as devices. Some of these devices are physical devices (for example, disk drives), while others can be thought of as abstract or virtual devices (for example, files).
- A system with multiple users may require us to first **request()** a device, to ensure exclusive use of it. After we are finished with the device, we **release()** it.
- Once the device has been requested (and allocated to us), we can **read() , write()** , and (possibly) **reposition()** the device, just as we can with files.

4. Information Maintenance

- Most systems have a system call to return the current time() and date() . Other system calls may return information about the system, such as the number of current users, the version number of the operating system, the amount of free memory or disk space, and so on.

5. Communication

- There are two common models of interprocess communication: the message-passing model and the shared-memory model.
- In the **message-passing model**, the communicating processes exchange messages with one another to transfer information. Messages can be exchanged between the processes either directly or indirectly through a common mailbox. Before communication can take place, a connection must be opened.

- Each computer in a network has a host name by which it is commonly known. Similarly, each process has a process name, and this name is translated into an identifier by which the operating system can refer to the process. The **get_hostid()** and **get_processid()** system calls do this translation.
- The identifiers are then passed to the **open_connection()** and **close_connection()** system calls.
- The recipient process usually must give its permission for communication to take place with an **accept_connection()** call. Most processes that will be receiving connections are special-purpose **daemons**, which are system programs provided for that purpose. They execute a **wait_for_connection()** call and are awakened when a connection is made.
- The source of the communication, known as the client, and the receiving daemon, known as a server, then exchange messages by using **read_message()** and **write_message()** system calls. The **close_connection()** call terminates the communication.

- In the **shared-memory model**, processes use **shared_memory_create()** and **shared_memory_attach()** system calls to create and gain access to regions of memory owned by other processes.
- They can then exchange information by reading and writing data in the shared areas.

7. Protection

- Protection provides a mechanism for controlling access to the resources provided by a computer system.
- System calls providing protection include **set_permission()** and **get_permission()** , which manipulate the permission settings of resources such as files and disks.
- The **allow_user()** and **deny_user()** system calls specify whether particular users can—or cannot—be allowed access to certain resources.

System Programs

- System programs, also known as system utilities, provide a convenient environment for program development and execution.

1. **File management**. These programs create, delete, copy, rename, print, dump, list, and generally manipulate files and directories.

2. **Status information**. Some programs simply ask the system for the date, time, amount of available memory or disk space, number of users, or similar status information. Others are more complex, providing detailed performance, logging, and debugging information.

3. **File modification**. Several text editors may be available to create and modify the content of files stored on disk or other storage devices.

4. Programming-language support. Compilers, assemblers, debuggers, and interpreters for common programming languages (such as C, C++, Java, and PERL) are often provided with the operating system or available as a separate download.

5. Program loading and execution. Once a program is assembled or compiled, it must be loaded into memory to be executed. The system may provide absolute loaders, relocatable loaders, linkage editors, and overlay loaders.

6. Communications. These programs provide the mechanism for creating virtual connections among processes, users, and computer systems. They allow users to send messages to one another’s screens, to browse Web pages, to send e-mail messages, to log in remotely, or to transfer files from one machine to another.