Module 1

INTRODUCTION TO OPERATING SYSTEMS

Program

A program is a collection or group or set of statements or instructions which performs a particular task.

Software

Software is a collection or set of programs which collectively performs a set of activities.

Types of software

System software
Application software

System software

Software which provides services to other software is called as system software.

System Software is used for operating computer hardware.

System Software is installed on the computer when the operating system is installed.

The user does not interact with the system software because it works in the background.

System software provides platform for running application software.

Example system software are: Operating Systems, Compilers, Linkers, Loaders, Assemblers, debugger, driver and so on.

Application software

Software which depends on other software for services is called as application software.

Application software is used by the user to perform a specific task.

Application software are installed according to user's requirements.

The user interacts with application software.

Application software can't run independently. They can't run without the presence of system software.

Example application software are: VLC player, web browser, word processor and so on.

Computer

A computer contains number of hardware and software components.

Each component of the computer system is called a resource.

Some of the hardware resources are: processor, HD, RAM, keyboard, monitor and so on.

Some of the software resources are: operating system, compiler, interpreter, loader, assembler and so on.

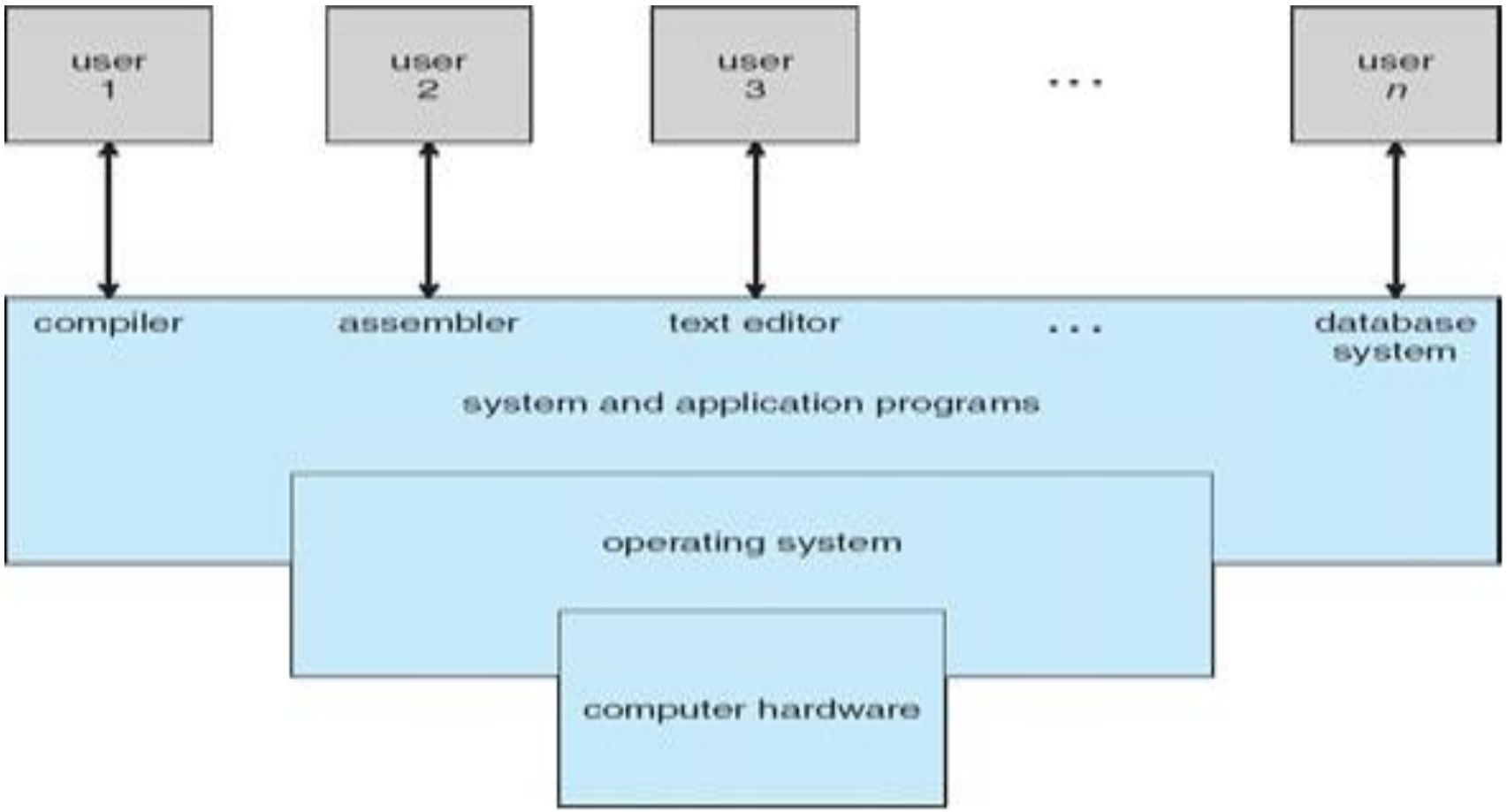
Operating System

An Operating System is a software that manages the computer hardware.

It also provides a basis for application programs and acts as an intermediary between the computer user and the computer hardware.

Some operating systems are designed to provide convenient communication between user and computer system, others for efficient utilization of resources, and others to provide both.

Role of Operating System



A computer system can be logically divided into four components: the hardware, the operating system, the application programs, and the users as shown in above figure.

The **hardware** (the central processing unit, the memory and the I/O devices) provides the computing resources for the computer system.

The **system programs** (compilers, assemblers and so on) and **application programs** (word processors, spreadsheets, web browsers and so on) help the user programs in their execution.

The **operating system** controls the hardware and coordinates the use of hardware among various user programs.

The role of operating system is fully explained by considering two views: user view and system view.

User view

Computer systems are divided into four categories from user point of view.

1) Personal computers

Used by single person for home applications. The Operating System used in personal computers should focus on the following

Easy use of the system
Utilization of resources
Performance of the system

2) Thin clients

A thin client is a system which do not have any processing capability and storage capacity.

It is used for typing the program and displaying the output.

Thin clients are connected to a server system which has storage capacity and execution capability.

The programs typed in the thin clients are stored and executed in server system. The results of execution are passed to client systems.

Ex: Oracle server

The operating system used in thin clients should concentrate on resource utilization.

3) Client – Server system

Number of client systems are connected to a server system.

A client system makes a request to a server system for any service.

The server system provides service to all client systems connected to it.

The Operating System used in client system should use resources of the client system as well as resources available at the server system.

4) Home devices or devices used in automobiles

An example for home device is a microwave oven.

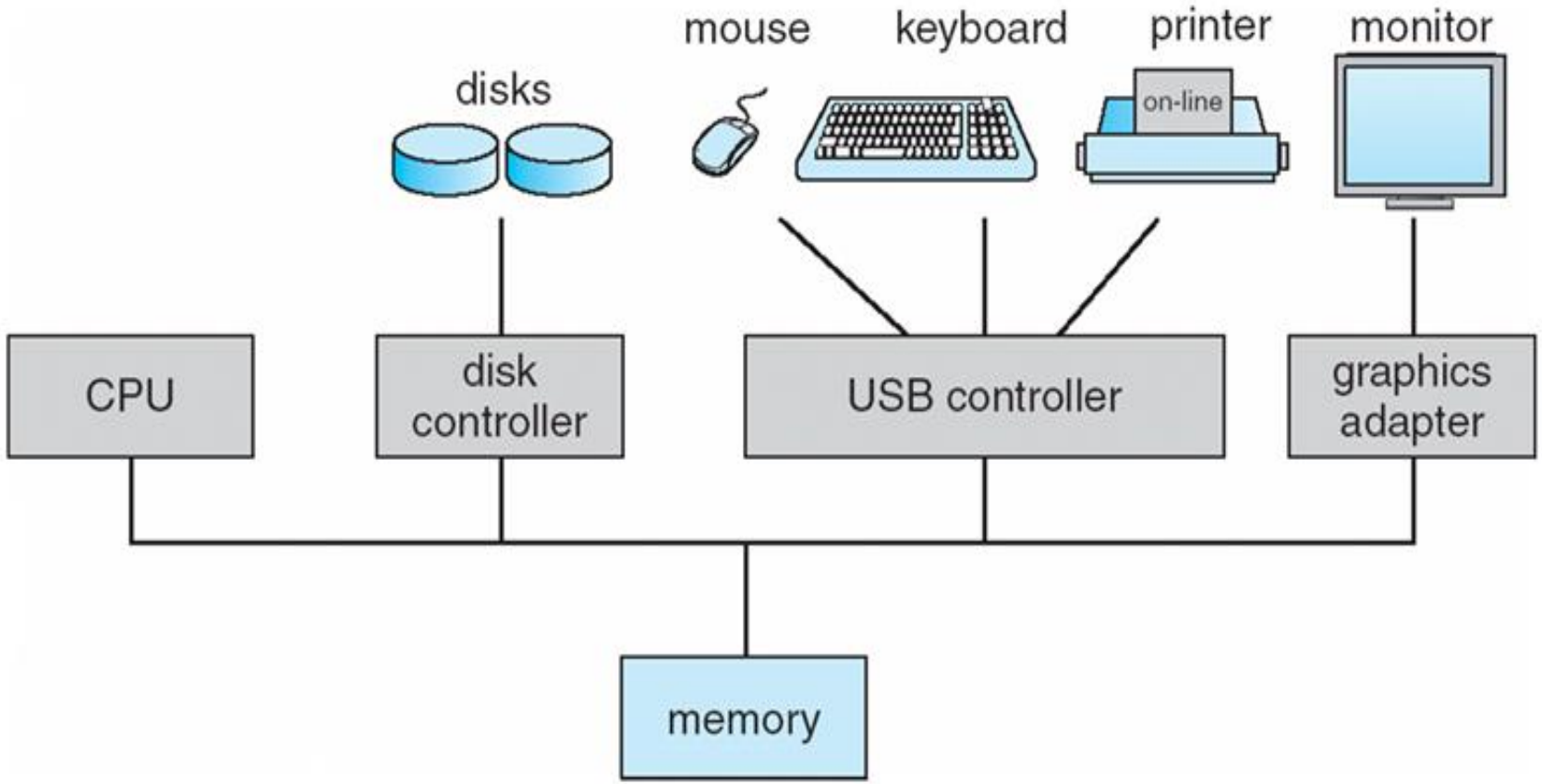
The Operating System used in home devices should work automatically without any intervention of user.

System view

From system point of view, the Operating System can be viewed as resource allocator.

The Operating System allocates the resources of the computer system to the programs that are currently running in the system.

Computer System Organization



A modern computer system consists of one or more CPUs and number of devices.

The CPUs and devices are connected to the system bus as shown in diagram.

The devices are connected to the system bus through device controllers.

Each device controller controls the operations of a device or a set of devices.

When the computer system is booted, a program called bootstrap program starts running.

The bootstrap program resides in ROM or EEPROM.

Role of bootstrap program is locating the operating system, loading the operating system into RAM and starting the execution of operating system.

When the operating system is started then the operating system controls the operations of computer system.

Operating System Operations

The following are the major operations or functions of an operating system

1. Process Management
2. Memory Management
3. File Management
4. I/O Management
5. Protection
6. Security
7. Networking

Process Management

To mange processes, the Operating System performs the following activities

1) creating and deleting processes

2) suspending and resuming processes

3) providing synchronization

4) providing communication

5) handling deadlocks

Creating and deleting processes

To create a new process, the operating system has to do the following activities

1) Move the program from disk to RAM

2) Allocate CPU or processor to the program

To delete a process, the Operating System has to move the program form RAM to disk.

Suspending and resuming processes

Before suspending the process, the Operating System has to save the status of the execution of the process.

Before resuming the process, the Operating System has to restore the saved status of the process.

Providing synchronization

Sharable resource

Ex: RAM, File etc.

Non sharable resource

Ex: CPU, printer.

When number of processes requests a non sharable resource at the same time, then the Operating System has to provide synchronous access to the resource by allocating the resource to only one process at a time.

Providing commutation

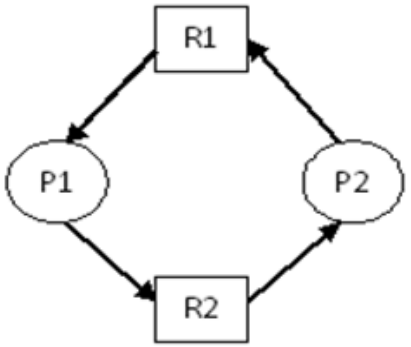
Programs or processes communicate for sharing or accessing data.

Two methods used for providing communication are:

1) Shared memory
2) Message passing

Handling deadlocks

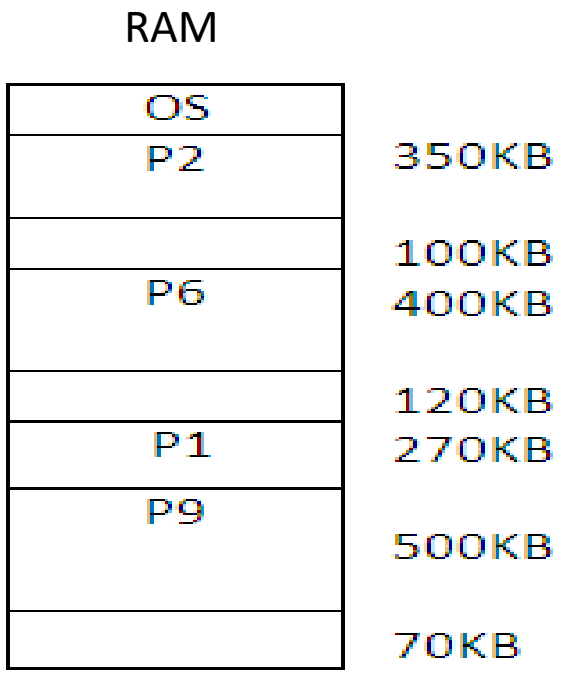
A set of processes is said to be in deadlock state if each process in the set is waiting for another process.



The Operating System has to allocate the resources to processes such that the system should not go into the deadlock state.

Memory Management

Operating system has to manage both primary memory (RAM) and secondary memory (Hard disk).



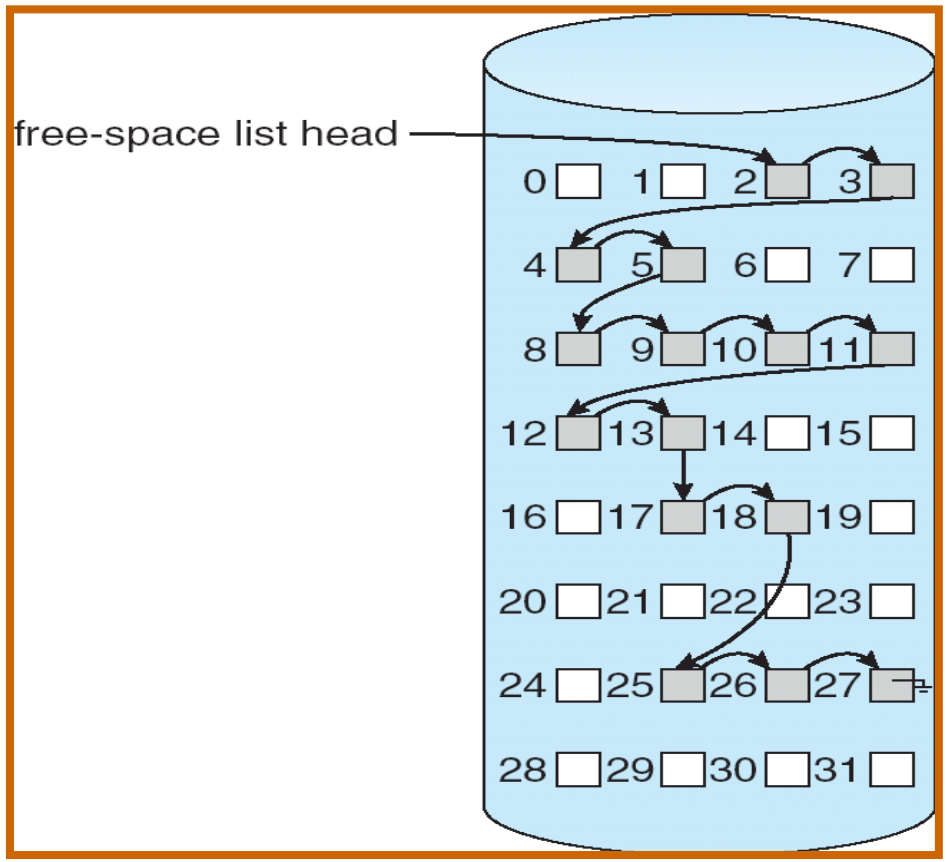
For managing the primary memory, the Operating System is responsible for

1) Knowing which parts of the RAM are in use and by whom

2) Allocating & de allocating space when required

3) Deciding which process needs to be moved between RAM and Hard disk

The hard disk contains number of storage blocks. At any time, some of the blocks contains programs and remaining blocks are free.



For managing the secondary memory, the Operating System is responsible for

1) Free – space management

2) Storage allocation

3) Disk scheduling

File Management

File is a logical storage unit.

The Operating System maps files onto storage devices.

Files are organized into directories to make them easier to use.

The Operating System is responsible for the following activities for managing files

1) Creating and deleting files
2) Creating and deleting directories
3) Supporting primitives for manipulating files and directories
4) Mapping files onto secondary storage
5) Backing up files on stable storage

I/O device Management

A computer system contains number of input and output devices like keyboard, mouse, monitor, printer and so on.

Each device is controlled by a device driver.

The Operating System has to manage all these device drivers.

Networking

A network is a collection of systems connected to each other.

The Operating System used in a computer which is connected to a network must provide support

1) For sharing the resources available in the network.
2) Dividing the program into parts and distributing them to different systems in the network.
3) Providing transparency to the users.

Protection and Security

Protection: restricting access to the resources of the system from the processes that are running in the system.

For example, if two or more processes request the CPU at the same time then the Operating System has to restrict the access to the CPU by allocating the CPU to only one process at a time.

Security: restricting access to the system by the users.

Username and password are used to provide security.

Types of Operating Systems

1) Batch processing operating system
2) Multiprogramming operating system
3) Timesharing operating system
4) Distributed operating system
5) Real time operating system
6) Multi user operating system
7) Multi tasking operating system
8) Embedded operating system
9) Mobile operating system

Batch processing operating system

If the computer system is running with batch processing operating system then the execution of programs in the computer system is as follows:

The programs that users want to execute and the input data required for executing the programs are collected into a batch.

The batch of programs is then loaded into computer system for execution.

The batch of programs is then executed in sequential manner.

There is no interaction between users and computer during execution of the programs.

The outputs generated after execution of programs is collected into a batch and then distributed to users.

IBM’s Z/OS is an example Batch operating system.

Multiprogramming operating system

When only one program is loaded into main memory and if that program requires any i/o operation during its execution then the CPU will be in idle state until the i/o operation is completed.

The utilization of resources (CPU, i/o devices) is low.

To increase the utilization of resources, number of programs is loaded into memory.

During execution of program, if the program requires any i/o operation then the CPU is switched to another program so that the CPU is busy at all times.

Loading number of programs into main memory is called **multiprogramming.**

Multiprogramming operating system allows loading of number of programs at a time into RAM.

Multiprogramming operating system switches the CPU from current program to another program when the current process is completed or goes to the waiting state.

Windows and LINUX are examples for multiprogramming operating systems.

Timesharing operating system

Time sharing operating system loads number of programs at a time into main memory.

Allow each program to execute for a certain amount of time only. After that the CPU is switched to another program.

Each program has equal chance of getting the CPU.

Timesharing operating system switches the CPU from current program to another program

1) when the current process is completed
2) when the current process goes to the waiting state
3) when the allocated time slice is over.

The user can interact with his program while it is running.

A time-shared operating system allows many users to share the computer simultaneously.

Windows and LINUX are examples for timesharing operating systems.

Distributed operating system

Distributed operating system manages a group of independent computers and makes them appear to be a single computer.

The group of computers is connected through a network.

Following are the features of distributed operating systems:

Resource sharing

Resources can be shared by the computer systems connected to the network.

For example, if a printer is connected to the network then all systems in the network can share the printer.

Reliability of resources

If any resource of any system fails then the resource of other system in the network can be used.

Speed up of computations

Programs can be executed in less amount of time by dividing the program into number of parts and executing these parts on different systems in the network.

Communication between systems

If more number of persons is involved in the development of any project then there should be some communication between the systems that are being used by the persons.

This communication can be provided easily by the distributed operating system.

Providing Transparency

The distributed operating system hides the details of execution of the program from the user.

Amoeba and LOCUS are examples for distributed operating systems.

Real time operating system

Real time operating systems are used in computer systems which executes real time applications.

Real time applications have fixed time constraints on processing.

For example, if there is a satellite which sends some data for every 100 seconds and if a computer system receives and stores that data then the operating system used in that computer system is a real time operating system.

In this example, the operating system has to receive and store the data within 99 seconds.

Windows CE and Symbian are examples for real time operating systems.

Multi user operating system

Multi user operating system allows multiple users to access a computer system simultaneously.

Time-sharing systems can be classified as multi-user systems as they enable multiple user access to a computer through time sharing.

UNIX and open VMS are examples for multi user operating systems.

Multi tasking operating system

Multitasking operating system allows execution of multiple tasks at a time.

There are two versions of multi tasking: pre-emptive and co-operative.

In pre-emptive multitasking, the operating system slices the CPU time and dedicates one slot to each of the programs.

Unix-like operating systems such as Solaris and Linux support pre-emptive multitasking.

In co-operative multi tasking, CPU is switched to another process when the current process is completed or goes to the waiting state.

MS Windows prior to Windows 95 used to support cooperative multitasking.

Embedded operating system

The operating systems designed for being used in embedded computer systems are known as embedded operating systems.

Embedded operating systems are designed to operate on small machines like PDAs with less autonomy.

Embedded operating systems are able to operate with a limited number of resources.

Embedded operating systems are very compact and extremely efficient.

Windows CE and FreeBSD are some examples of embedded operating systems.

Mobile operating system

A mobile operating system controls a mobile device and its design supports wireless communication and mobile applications.

Tablet PCs and smart phones run on mobile operating systems.

Blackberry OS, Google's Android and Apple's iOS are some of the most known names of mobile operating systems.

System calls

The services provided by an operating system are made available to user programs through system calls.

System calls are generally developed in C and C++ languages.

Some system calls which directly interacts with hardware devices are developed in assembly language.

System calls are executed in kernel of operating system when a user program requests the operating system for any service.

For example, when a user program calls a scanf() function for reading data from keyboard then the system call ‘read()’ is invoked and executed in the kernel.

Number of system calls is invoked during execution of even a small program.

For example, assume that there is a program for copying data from one file to other file. Some of the system calls invoked during the execution of this program are:

1. A system call to display the prompt message
2. A system call to read the name of source file
3. A system call to display the prompt message
4. A system call to read the name of destination file
5. A system call for opening the source file
6. A system call for opening the destination file
7. A sequence of system calls for reading data from the source file and writing data into destination file
8. A system call for closing the source file
9. A system call for closing the destination file

The programmers develop programs using Application Programming Interface (API).

The API contains set of functions/methods using which the programmer develops programs.

The API also specifies the parameters that need to be passed to each function and also the values returned by each function.

Java API is one example.

Each function or method in the API is linked to a system call.

When any API function or method is executed in the program then the corresponding system call is invoked.

There are two main reasons for writing programs using API instead of system calls directly.

1. Program written using API can be executed on any system that supports the same API.

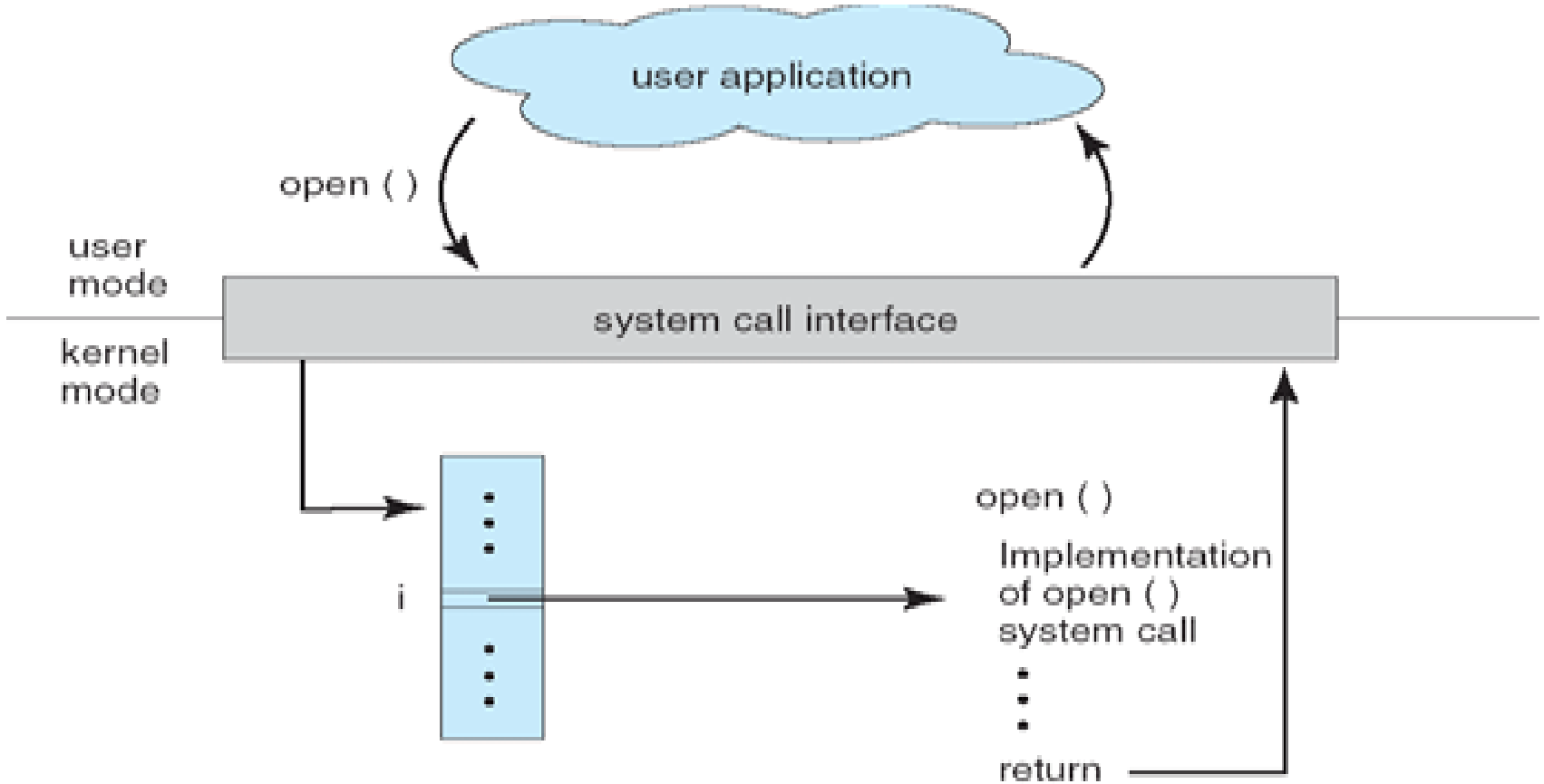
1. Working with system calls is more difficult than API.

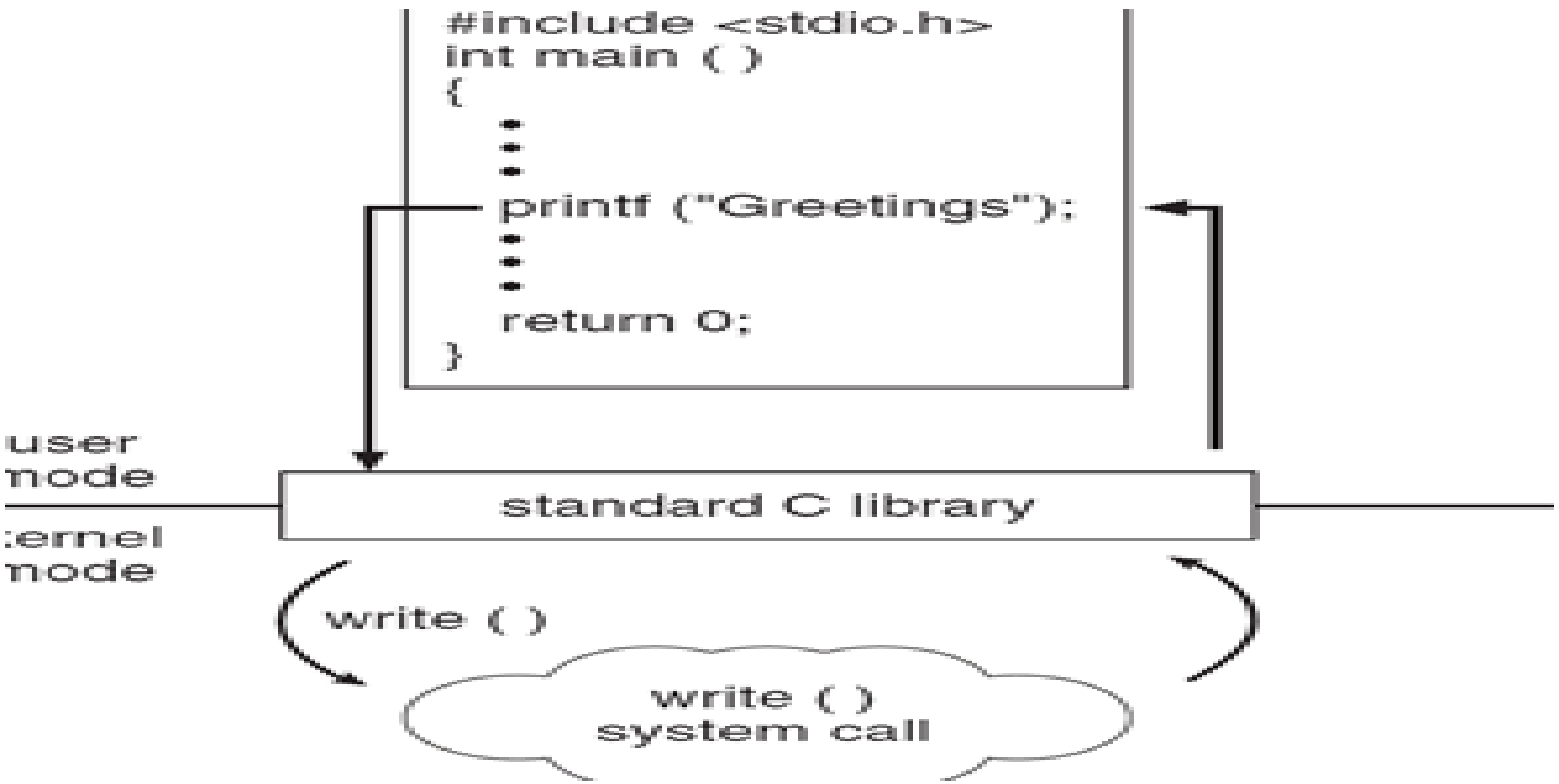
System call interface

The run-time support system for most programming languages provides a system-call interface that serves as the link to system calls made available by the operating system.

During execution of user program, when any API function is executed then the system-call interface identifies the corresponding system call and activates that system call within the operating system.

The system-call interface also returns the status of the system call and any return values.


user application
open ( )
user
mode
kernel
mode
system call interface
i
open ( )
Implementation
of open ( )
system call
return


#include <stdio.h>
int main ( )
{
printf ("Greetings");
return 0;
}
user
node
standard C library
write ( )
write ( )
system call

Types of system calls

1) Process control system calls

2) File management system calls

3) Device management system calls

4) Information maintenance system calls

5) Communication maintenance system calls

6) Protection and security maintenance system calls

Process control system calls

The various system calls for controlling processes are

1. end, abort

2. load, execute

3. create process , terminate process

4. get process attributes , set process attributes

5. wait, signal

File management system calls

1. create, delete

2. open, close

3. read, write, reposition

4. get file attributes, set file attributes

Device management system calls

1. request device, release device

2. read, write

3. get attributes, set attributes

Information maintenance system calls

1. get time or date, set time or date
2. get system data, set system data

Communication maintenance system calls

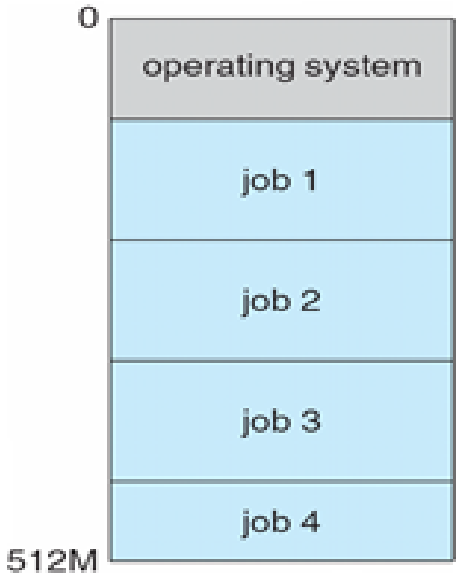
1. open connection, close connection
2. get hostid, get process id
3. send message, receive message
4. shared memory create, shared memory attach

protection

1. set permission, get permission
2. allow user, deny user

Dual mode operation of the system

A modern computer system operates in 2 modes in order to protect the operating system code from user processes and also the code of each process from other processes.



The two modes are:

1. User mode
2. Kernel mode (supervisor mode, privileged mode or system mode)

A bit called ‘mode bit’ indicates the current mode.

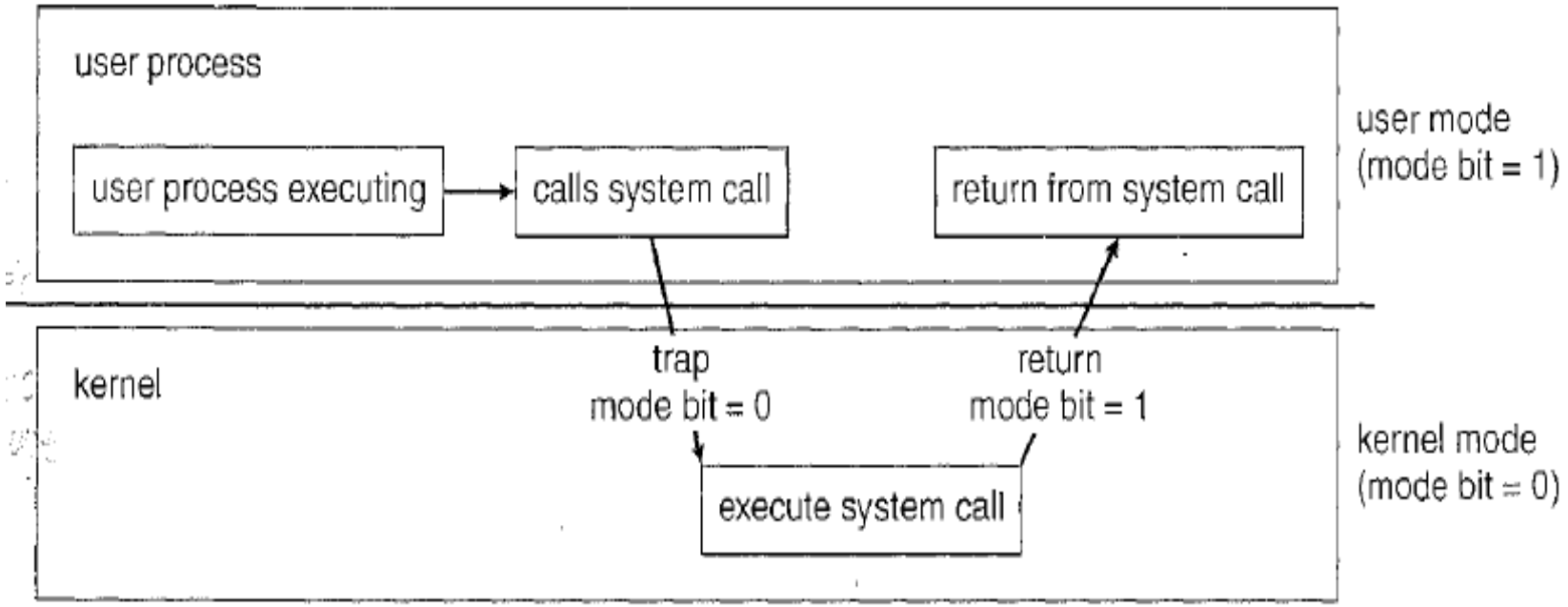
The value of mode bit is ‘0’ for kernel mode and ‘1’ for user mode.

When the system is booted, the system is running in kernel mode.

When the operating system starts any user program then the mode of system is switched to user mode.

During execution, if the user program requests any service from operating system by calling a system call then the mode is changed to kernel mode.

After executing system call, the mode is switched to user mode by setting mode bit to ‘1’ before passing the control to user program.



To protect the operating system code, some of the machine instructions are designated as privileged instructions.

These privileged instructions are executed only in kernel mode.

The instruction used to change the mode bit is an example for privileged instruction.

When any privileged instruction is executed in user mode then the hardware informs to the operating system.

Fork() System Call

Fork system call is used for creating a new process, which is called ***child process***, which runs concurrently with the process that makes the fork() call (parent process).

After a new child process is created, both processes will execute the next instruction following the fork() system call.

Separate copy of the variables is maintained in parent and child processes. Changes made to the values of variables by parent process will not be reflected in the child process and vice versa.

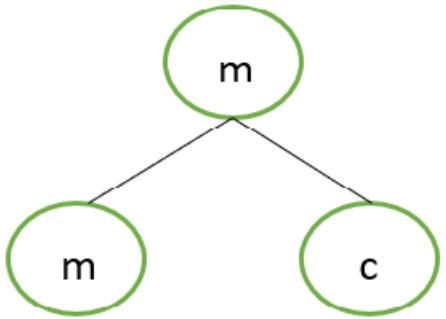
Fork system call takes no parameters and returns an integer value. Below are different values returned by fork().

Negative Value: creation of a child process was unsuccessful.
Zero: Returned to the newly created child process.
Positive value: Returned to parent or caller. The value contains process ID of newly created child process.

Predict the Output of the following program:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
        fork();
        printf("Hello world!\n");
        return 0;
}
```

Process hierarchy



Output:

```
Hello world!
Hello world!
```

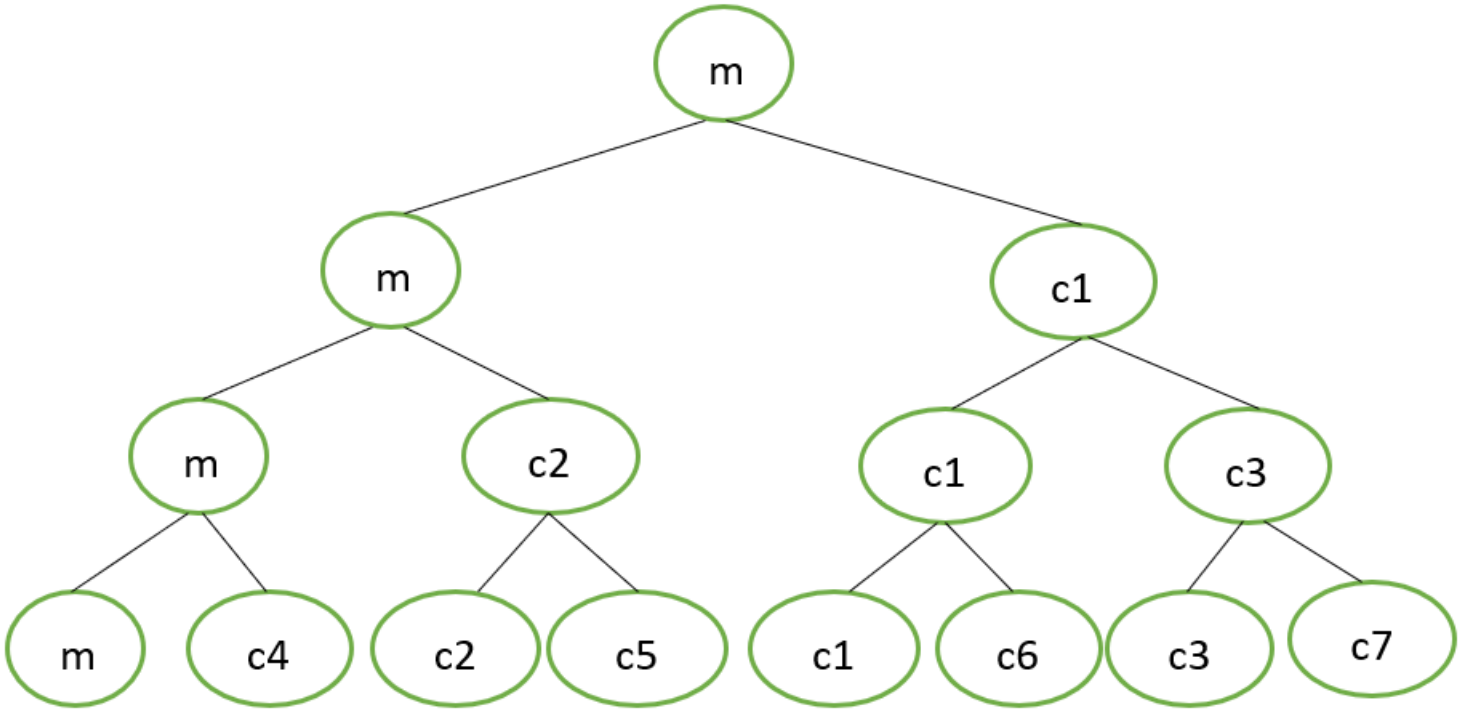
Calculate number of times hello is printed:

```
#include <stdio.h>
#include <sys/types.h>
int main()
{
        fork();
        fork();
        fork();
        printf("hello\n");
        return 0;
}
```

Output:

```
hello
hello
hello
hello
hello
hello
hello
hello
```

Process hierarchy

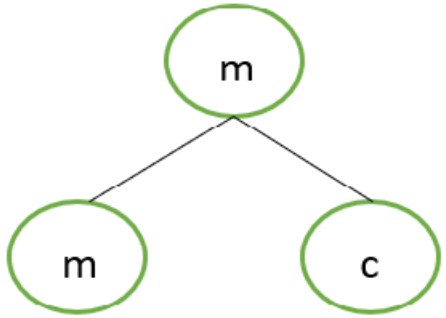


Predict the Output of the following program:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
void forkexample()
{
        if (fork() == 0)
                printf("Hello from Child!\n");

        else
                printf("Hello from Parent!\n");
}
int main()
{
        forkexample();
        return 0;
}
```

Process hierarchy



Output:

```
Hello from Child!
Hello from Parent!

(or)

Hello from Parent!
Hello from Child!
```

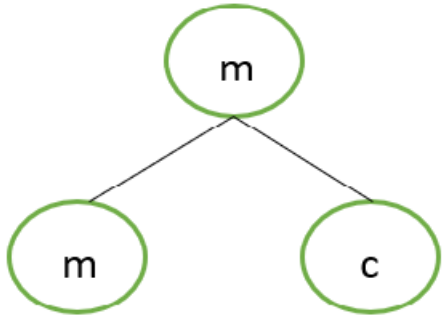
Predict the Output of the following program:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

void forkexample()
{
        int x = 1;

        if (fork() == 0)
                printf("Child has x = %d\n", ++x);
        else
                printf("Parent has x = %d\n", --x);
}
int main()
{
        forkexample();
        return 0;
}
```

Process hierarchy



Output:

```
Parent has x = 0
Child has x = 2

(or)

Child has x = 2
Parent has x = 0
```

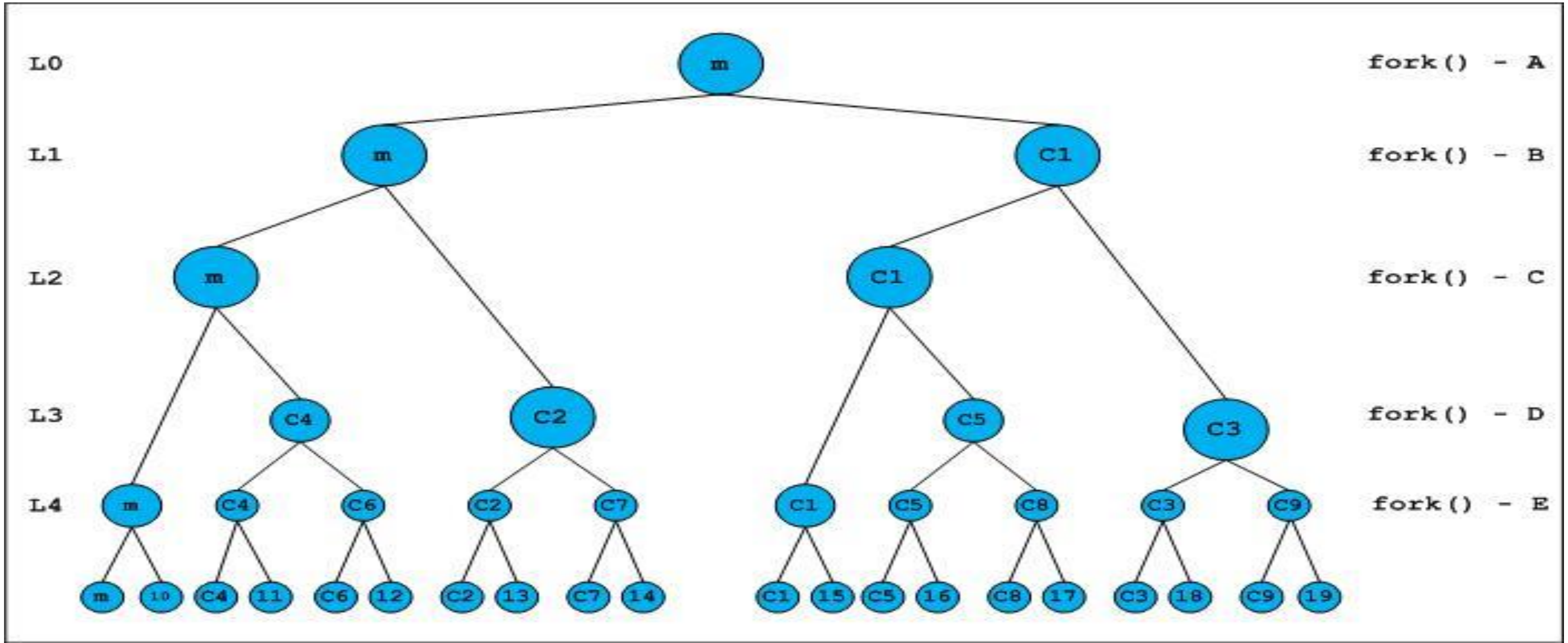
Predict output of below program.

```
#include <stdio.h>
#include <unistd.h>
int main()
{
        fork();
        ((fork() && fork()) || fork());
        fork();

        printf("forked\n");
        return 0;
}
```

```
#include <stdio.h>
int main()
{
            fork(); /* A */
            (( fork()  /* B */ && fork()  /* C */ ) || fork(); /* D */ )
            fork(); /* E */
            printf("forked\n");
            return 0;
}
```

Process hierarchy



Output:
Forked
Forked
Forked
Forked
Forked
Forked
Forked
Forked
Forked
Forked
Forked
Forked
Forked
Forked
Forked
Forked
Forked
Forked
Forked
forked

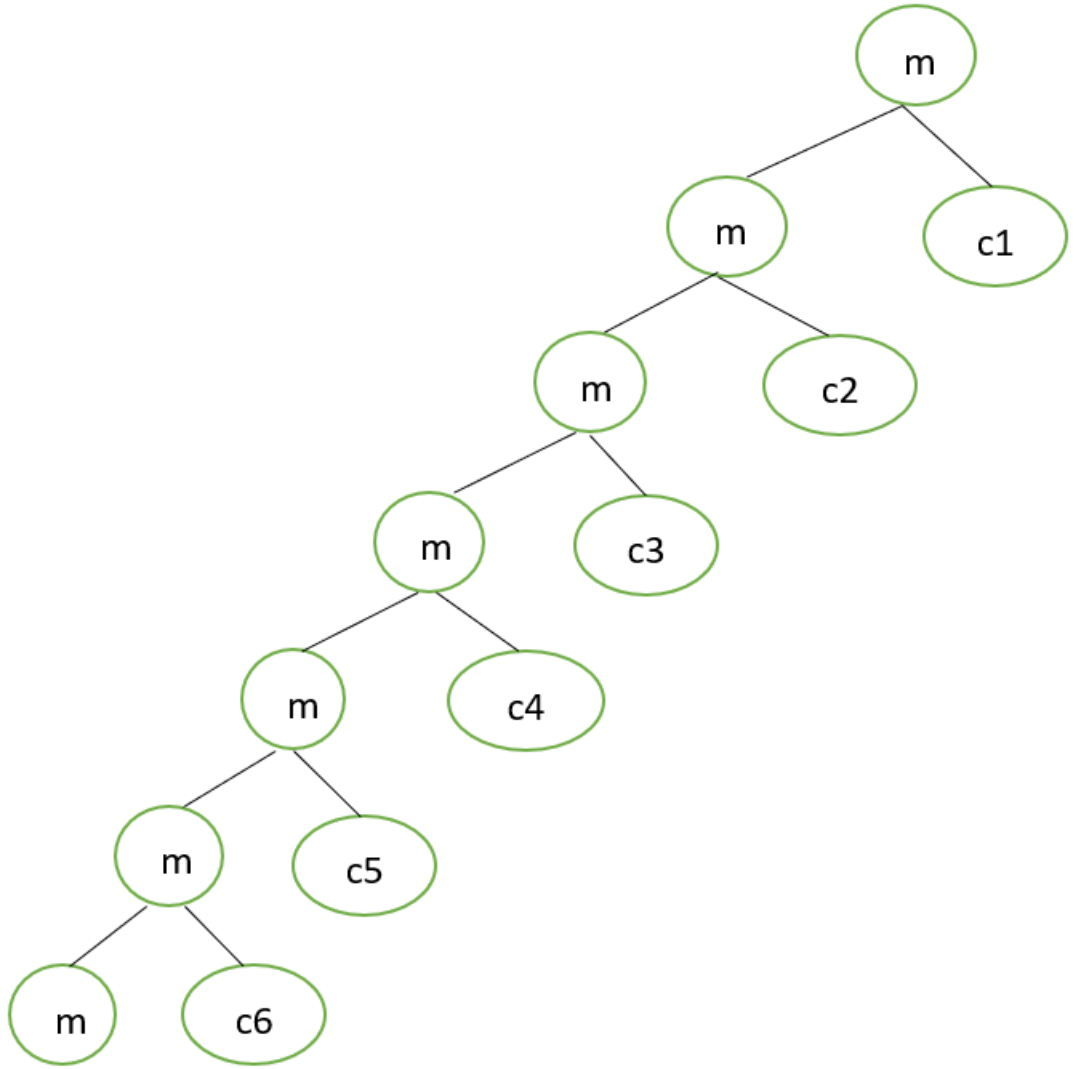
How many times **fork()** is invoked in the following program?

```
void main()
{
        int i = 0;
        for (i = 0; fork();i++)
        {
                if(i == 5)
                {
                        exit();
                }
        }
}
```

6 times

Final Clarification:
The value of i does not reset to 0 in any process because each process works on its own copy of i, which is independent of the others.
Control does not return to earlier processes like C1 after a later process (e.g., C6) terminates. Instead, once a process executes exit(), it is completely removed from the process tree.
The diagram showing the hierarchy of processes (M, C1, C2, ... C6) is accurate in depicting how processes are created and terminate, but no process resumes or resets after another terminates.

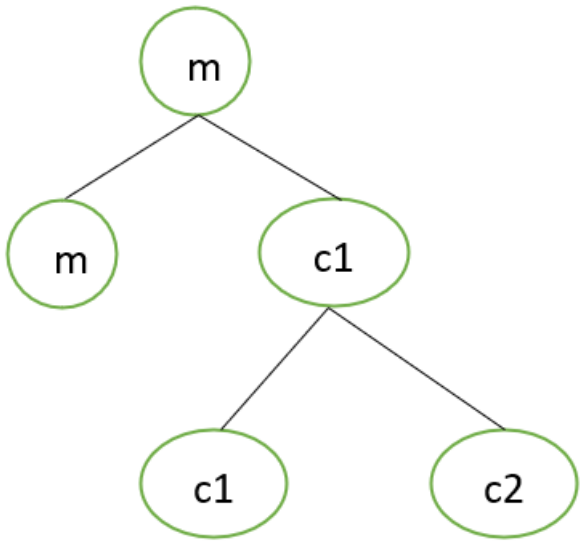
Process hierarchy



Mention the process hierarchy and output.

```
main()
{
    print("a\n");
    if(fork() == 0)
    {
            print("b\n");
            if(fork() == 0)
            {
                    print("c\n");
                    exit();
            }
            print("d\n");
            print("e\n");
            exit();
    }
    print("f\n");
    print("g\n");
    exit();
}
```

Process hierarchy



Output:

a
f
g
b
d
e
c

Draw the process hierarchy to represent the process creation and mention how many times “OS” will be printed.

```
void main()
{
 if(fork() && fork() || fork())
 {
    fork();
    fork();
    print(“OS”);
 }
}
```

Output:
OS
OS
OS
OS
OS
OS
OS
OS
OS
OS
OS
OS
OS
OS

Process hierarchy