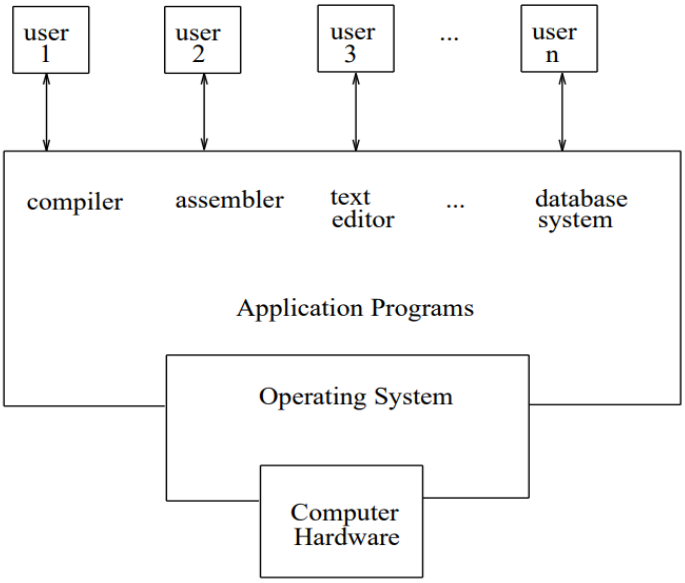
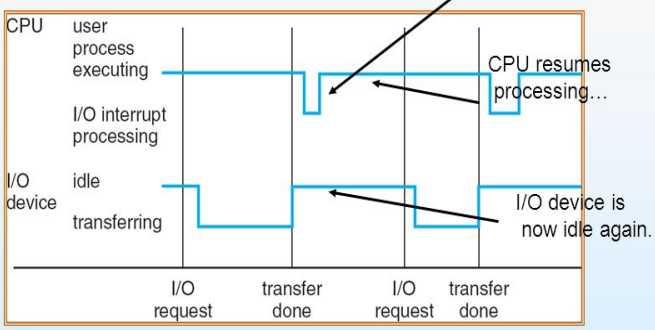


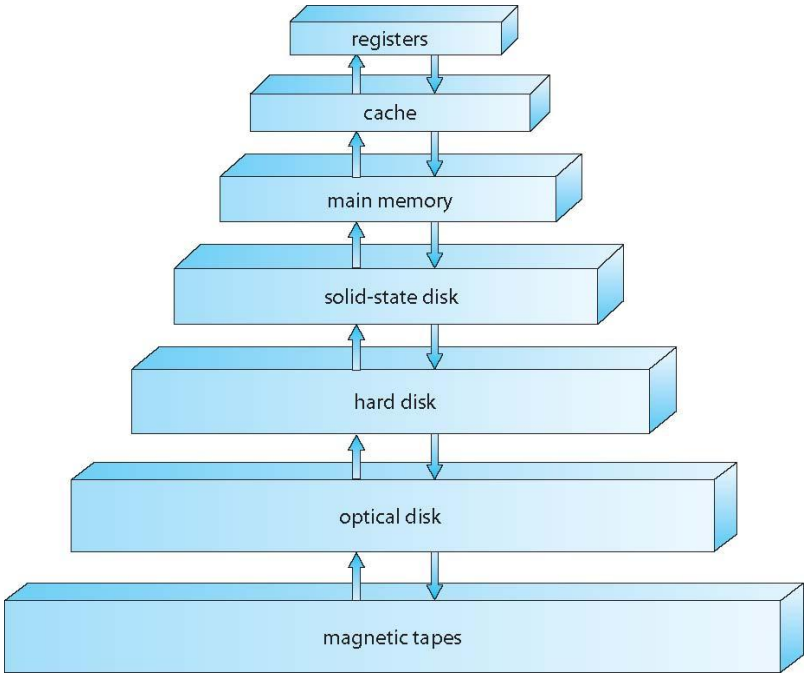

A computer system can be divided roughly into four components: the hardware, the operating system, the application programs, and the users (as shown in figure) Abstract view of the components of a computer system.

# Interrupt Timeline

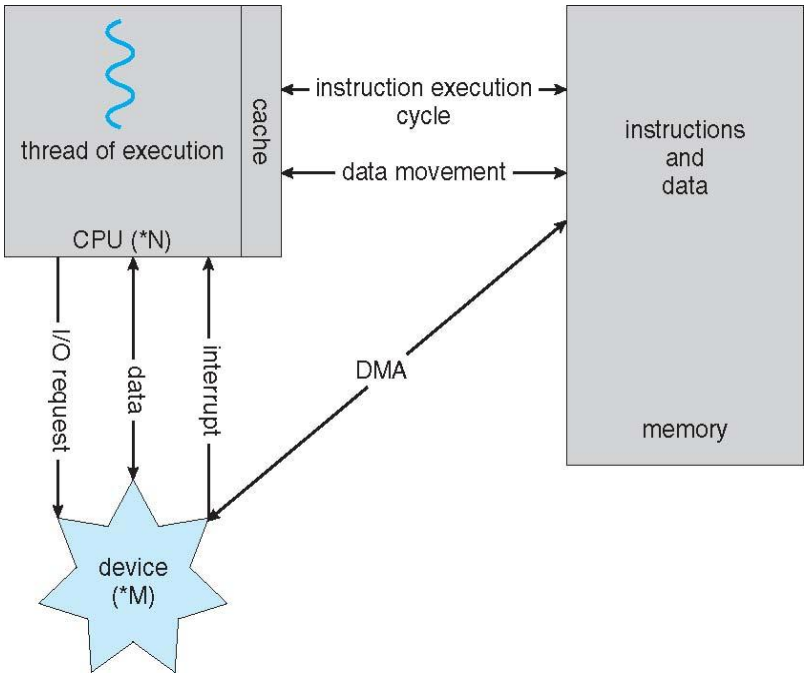



Consider: the CPU is busy executing some process.
In the background, an I/O device, say a disk, is transferring data from itself into primary memory.
When the transfer is complete, a signal (interrupt) is sent to the CPU.
The CPU can suspend 'current' operations, and process the I/O interrupt
Of course at this time, the I/O device becomes idle
Note the CPU time to process the interrupt is very small (much smaller than chart indicates).
Then the CPU resumes normal processing (whatever that might be)
It might resume suspended process or start executing a different process.

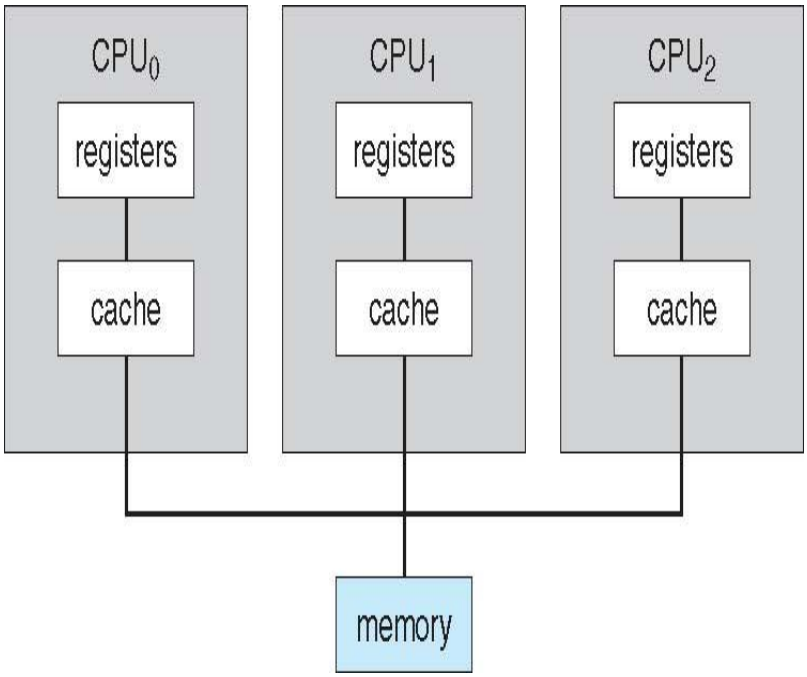
This is sufficient for now. Much more later.



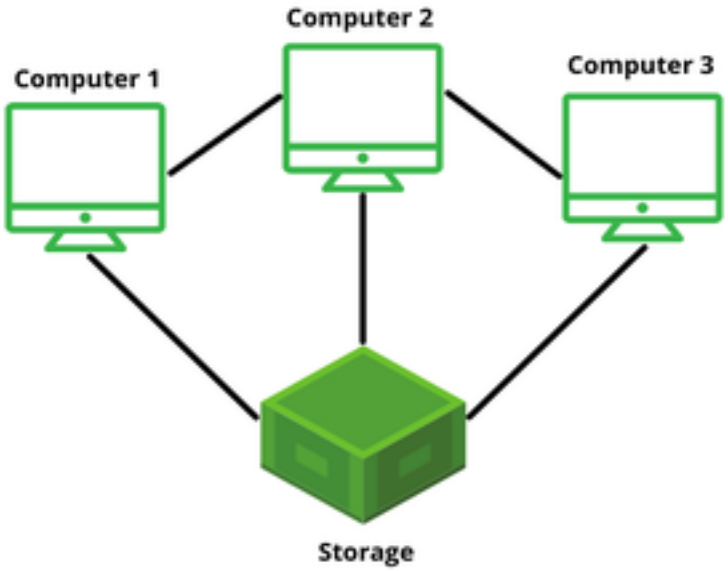

Storage-device hierarchy



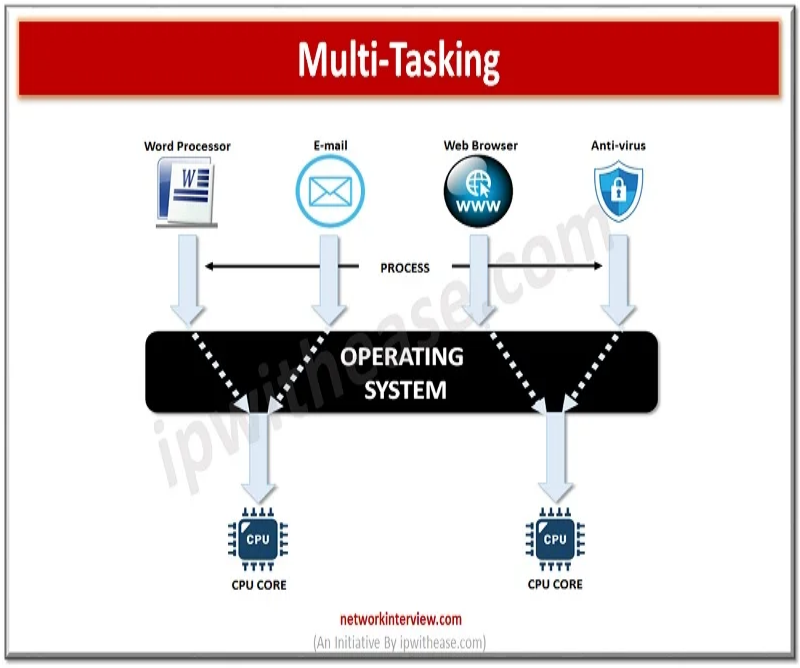

Working of I/O Operation

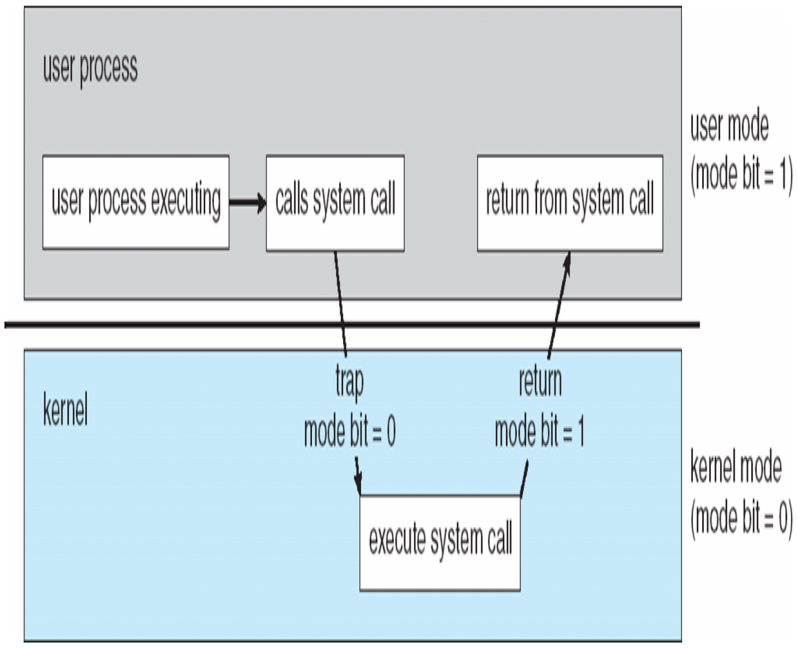


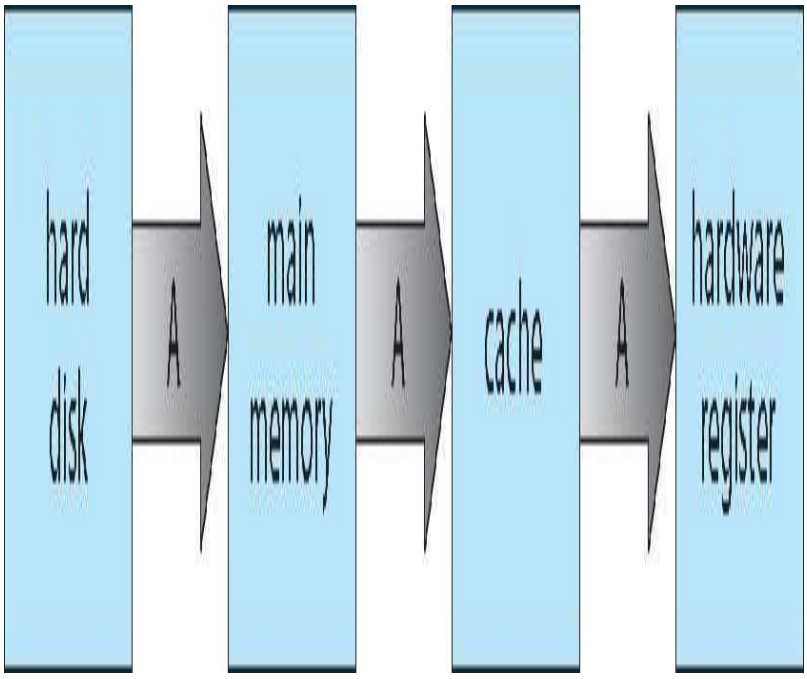

The most common systems use symmetric multiprocessing (SMP), in which each processor performs all tasks within the operating system. SMP means that all processors are peers; no master-slave relationship exists between processors. Figure illustrates a typical SMP architecture. Symmetric multiprocessing architecture.

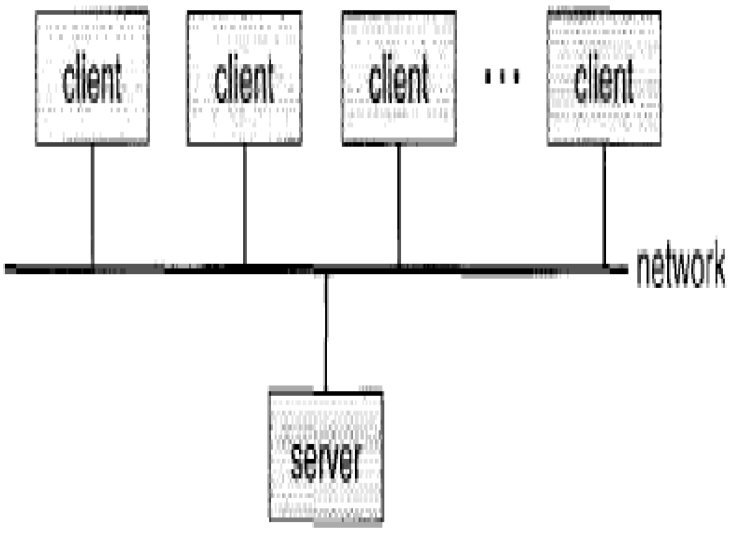


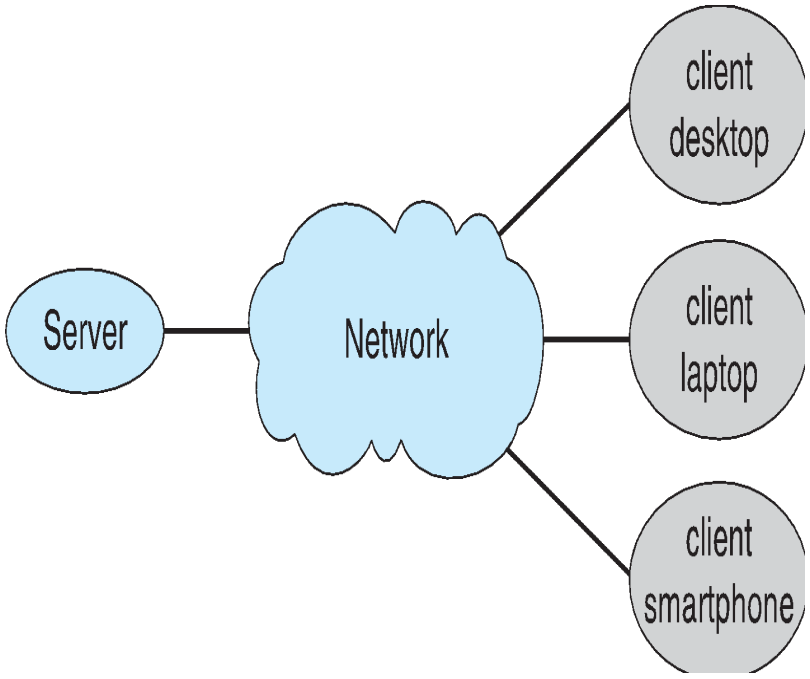

Another type of multiple CPU system is the clustered system. Like multiprocessor systems, clustered systems gather together multiple CPUs to accomplish computational work. Clustered systems differ from multiprocessor systems, however, in that they are composed of two or more individual systems coupled together called as cluster. Provides high availability.

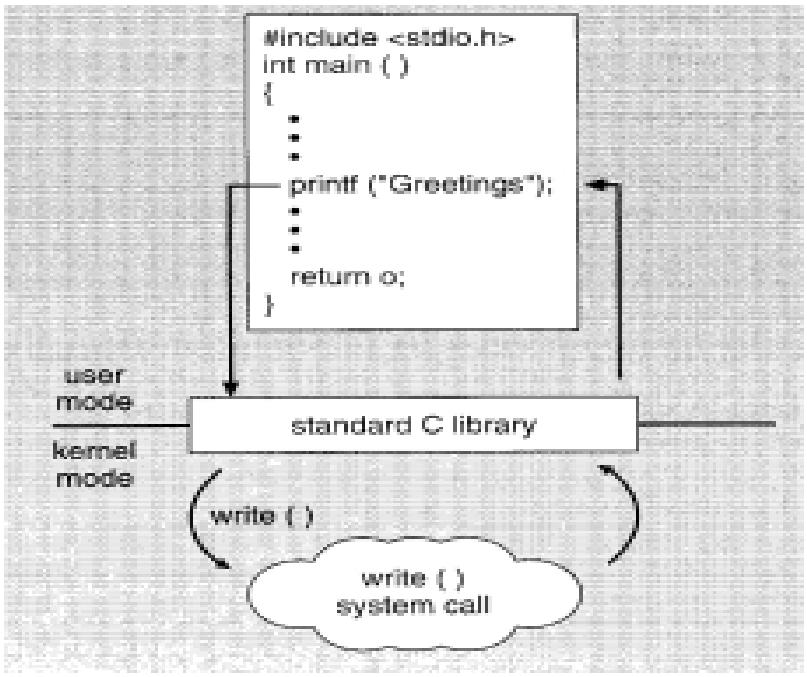

Multi-Tasking
Word Processor
E-mail
Web Browser
Anti-virus
PROCESS
OPERATING SYSTEM
CPU
CPU
CPU CORE
CPU CORE
networkinterview.com
(An Initiative By ipwithease.com)


user process
user process executing
calls system call
return from system call
user mode
(mode bit = 1)
kernel
trap
mode bit = 0
return
mode bit = 1
execute system call
kernel mode
(mode bit = 0)

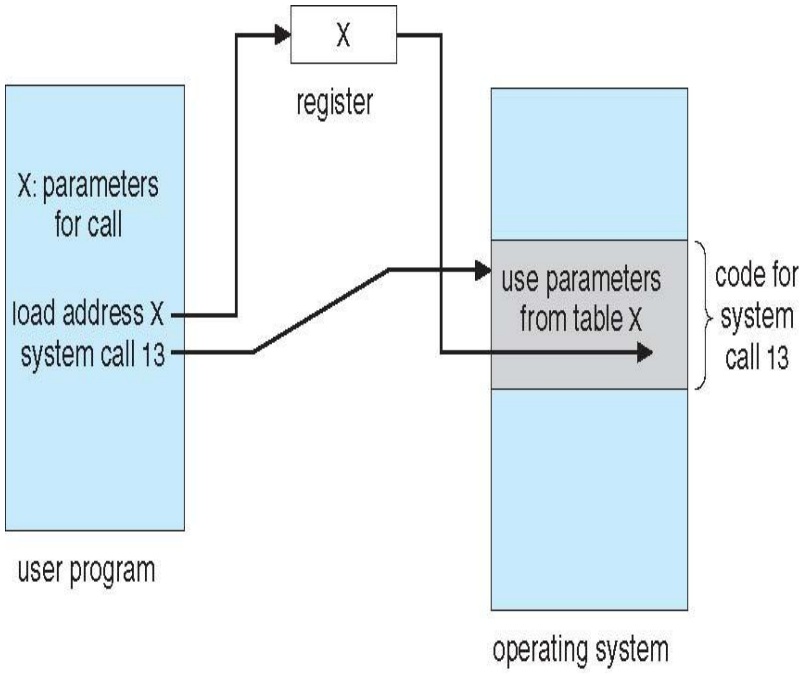

hard disk
A
main memory
A
cache
A
hardware register

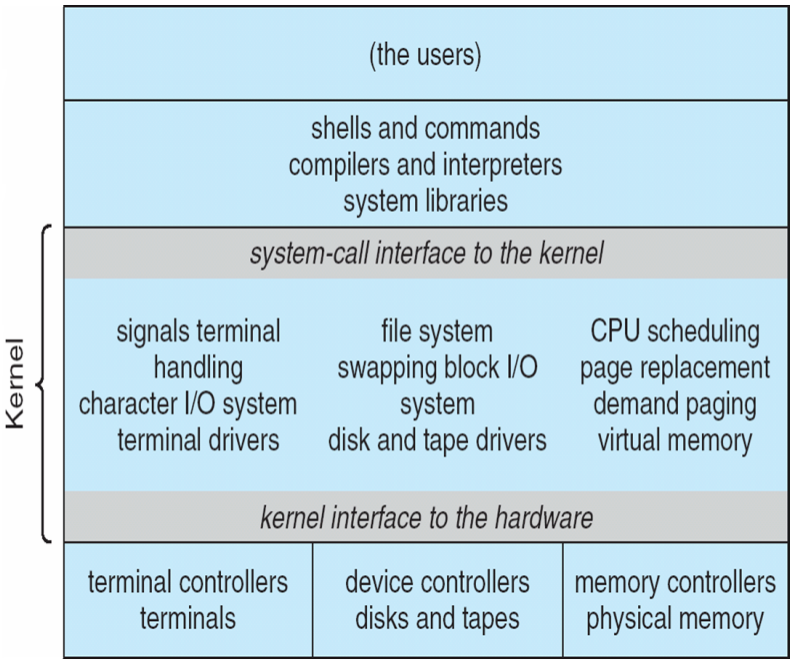

client
client
client
client
network
server


client
desktop
Server
Network
client
laptop
client
smartphone




The relationship between an API, the system-call interface, and the operating system is shown in Figure 1.13, which illustrates how the operating system handles a user application invoking the open ( ) system call.


X
register
X: parameters
for call
load address X
system call 13
user program
use parameters
from table X
code for
system
call 13
operating system


(the users)
shells and commands
compilers and interpreters
system libraries
system-call interface to the kernel
signals terminal handling
character I/O system
terminal drivers
file system
swapping block I/O system
disk and tape drivers
CPU scheduling
page replacement
demand paging
virtual memory
kernel interface to the hardware
terminal controllers
terminals
device controllers
disks and tapes
memory controllers
physical memory
Kernel

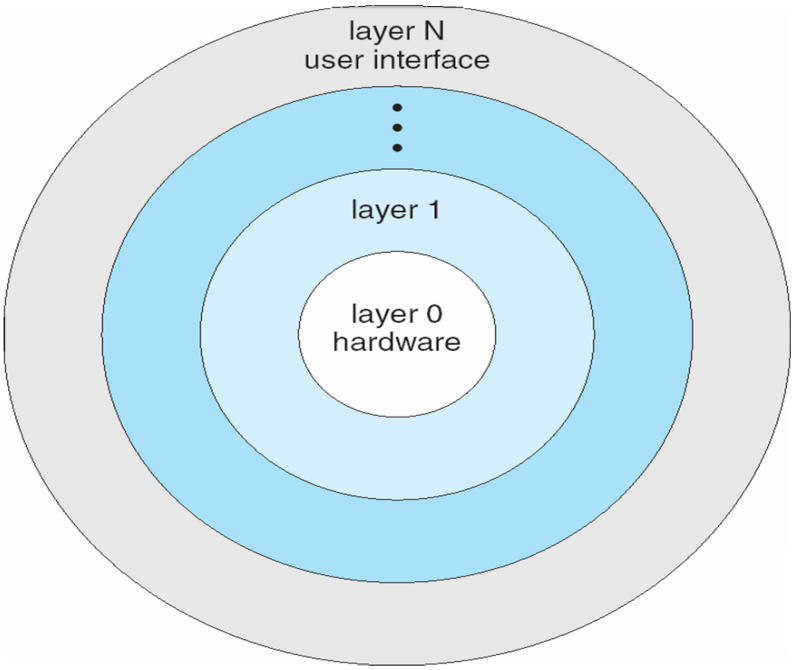
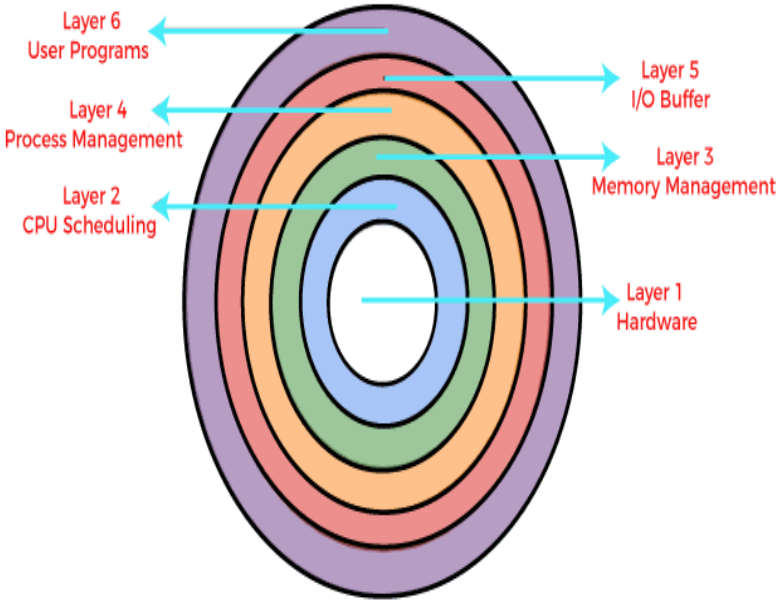
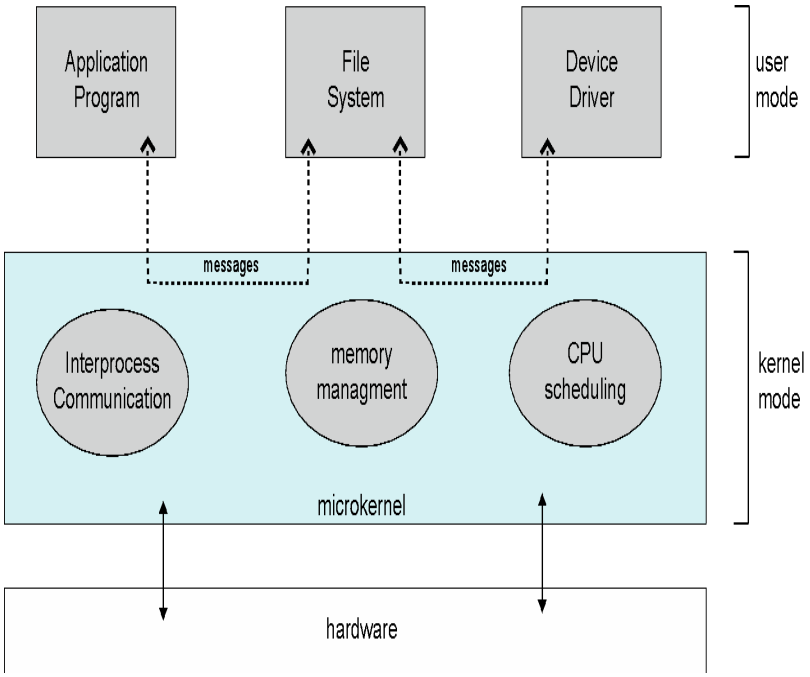
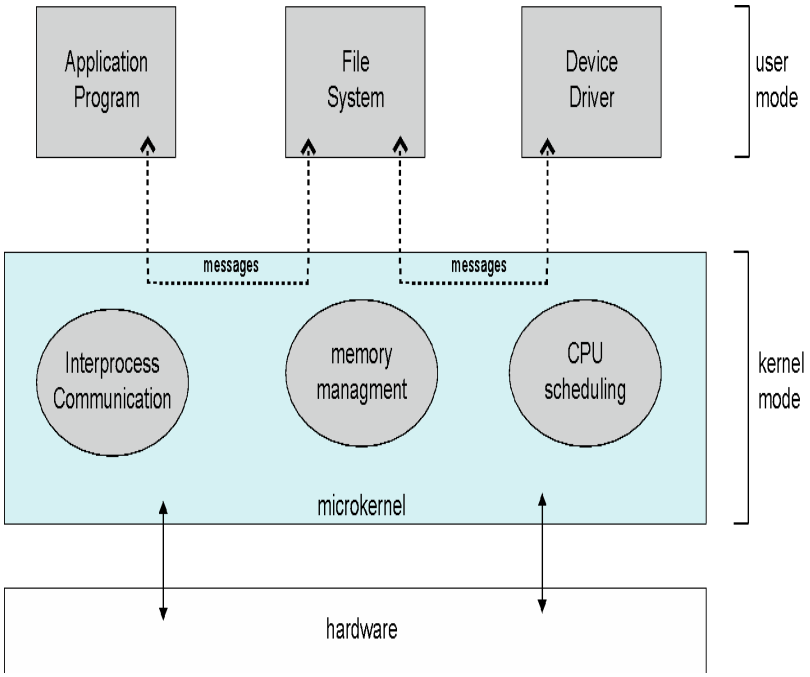


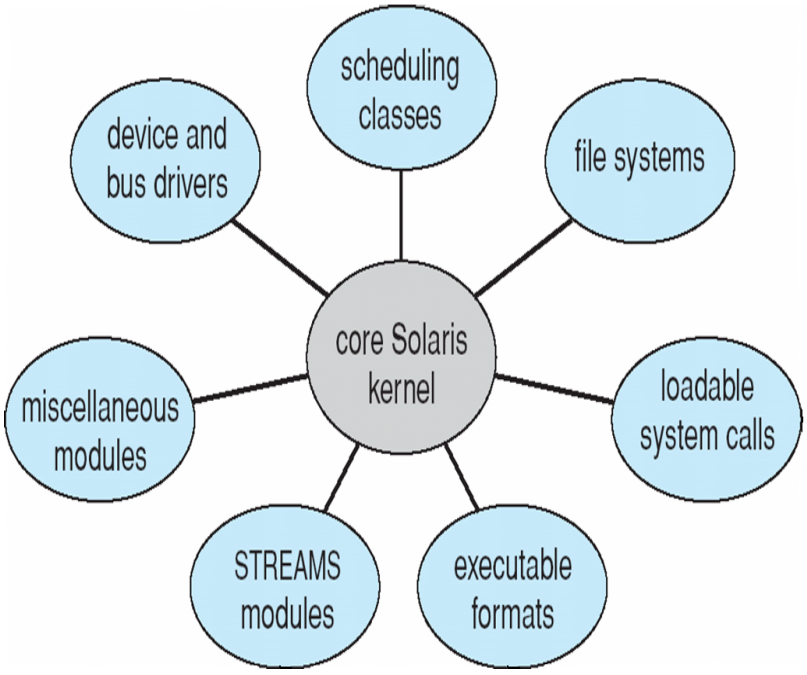

Figure: A layered operating system

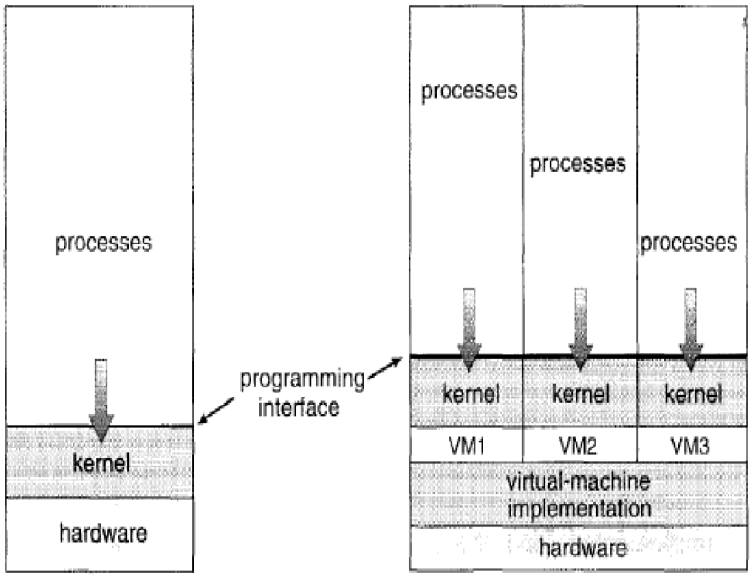


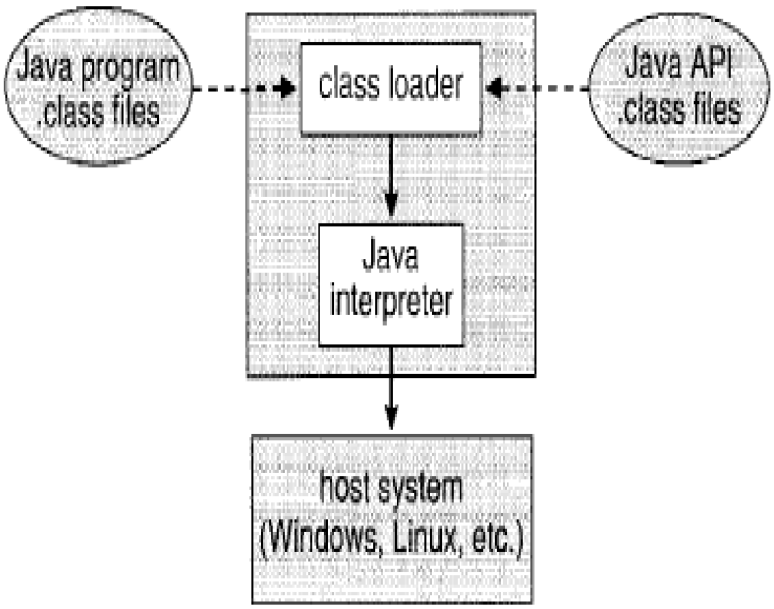

A final problem with layered implementations is that they tend to be less efficient than other types. For instance, when a user program executes an I/O operation, it executes a system call that is trapped to the I/O layer, which calls the memory-management layer, which in turn calls the CPU-scheduling layer, which is then passed to the hardware. At each layer, the parameters may be modified, dat


Application Program
File System
Device Driver
user mode
messages
messages
Interprocess Communication
memory managment
CPU scheduling
kernel mode
microkernel
hardware


Application Program
File System
Device Driver
user mode
messages
messages
Interprocess Communication
memory managment
CPU scheduling
kernel mode
microkernel
hardware


scheduling classes
device and bus drivers
file systems
core Solaris kernel
miscellaneous modules
loadable system calls
STREAMS modules
executable formats


processes
kernel
hardware
programming interface
processes
processes
processes
kernel
kernel
kernel
VM1
VM2
VM3
virtual-machine implementation
hardware
(a)
(b)


Java program
.class files
class loader
Java API
.class files
Java
interpreter
host system
(Windows, Linux, etc.)

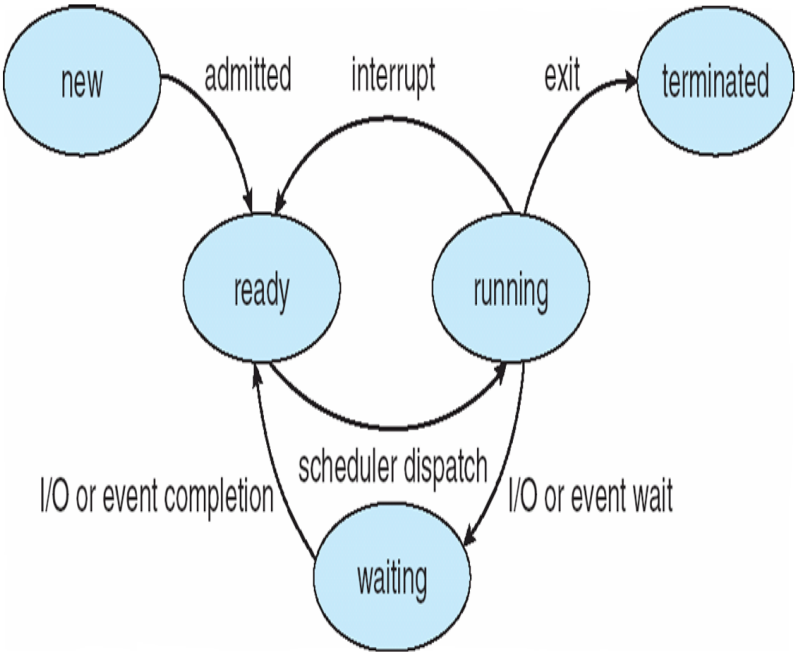
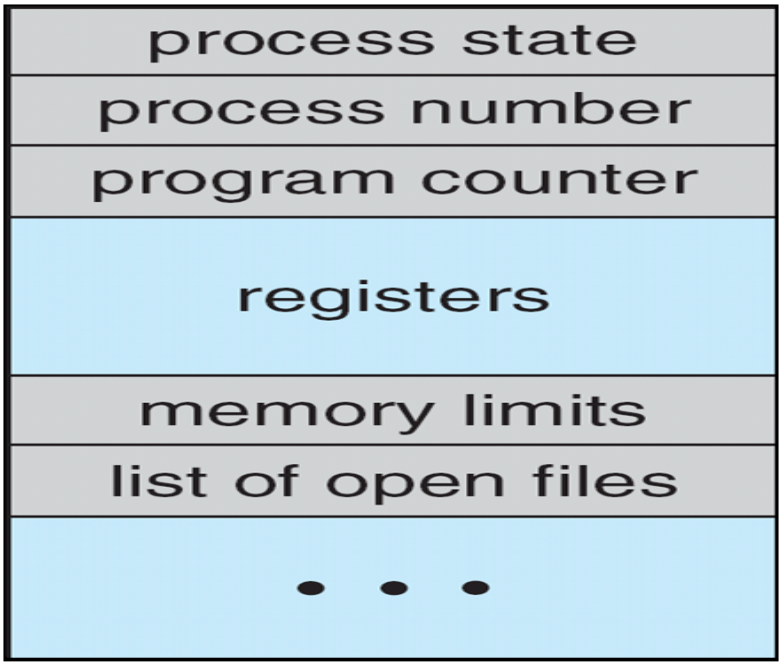
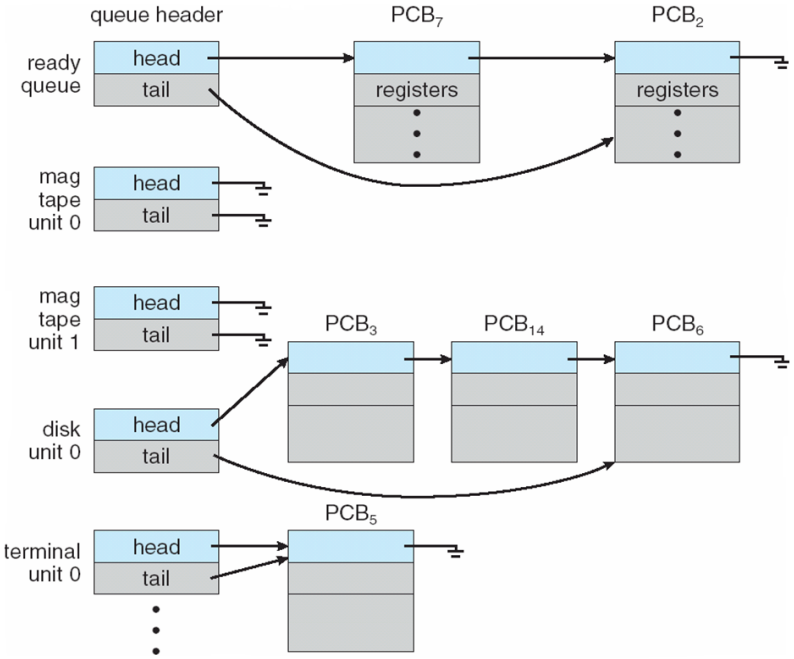



Diagram of process states.




Process Control Block Each process is represented in the operating system by a process control block (PCB) also called a task control block. A PCB is shown in Figure. It contains many pieces of information associated with a specific process, including these:


queue header
PCB7
PCB2
ready queue
head
tail
registers
registers
mag tape unit 0
head
tail
mag tape unit 1
head
tail
PCB3
PCB14
PCB6
disk unit 0
head
tail
PCB5
terminal unit 0
head
tail

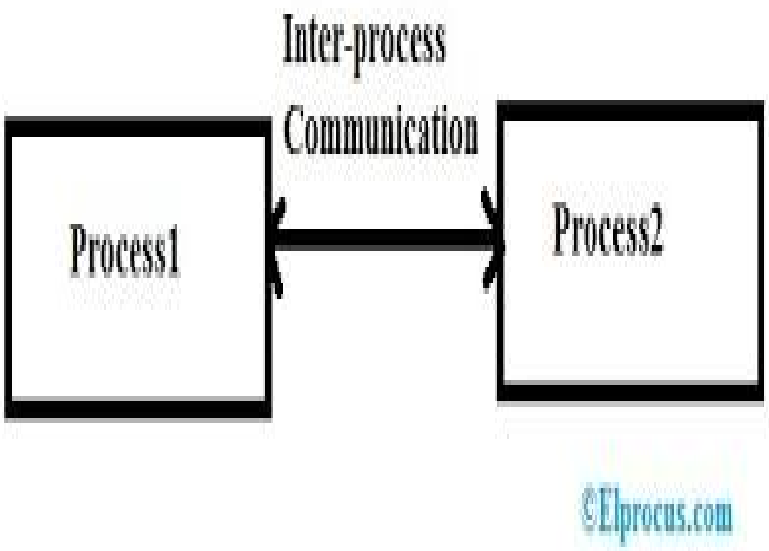
```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid;

    /* fork a child process */
    pid = fork();

    if (pid < 0) {/* error occurred */
       fprintf(stderr, "Fork Failed");
       exit(-1);
    }
    else if (pid == 0) {/* child process */
       execlp("/bin/ls","ls",NULL);
    }
    else {/* parent process */
       /* parent will wait for the child to complete */
       wait(NULL);
       printf("Child Complete");
       exit(0);
    }
}
```

To illustrate these differences, let's first consider the UNIX operating system. In UNIX, as we've seen, each process is identified by its process identifier,


Inter-process
Communication
Process1
Process2
©Elprocus.com