CMRIT
CMR INSTITUTE OF TECHNOLOGY, BENGALURU.
ACCREDITED WITH A+ GRADE BY NAAC
No.132, AECS Layout
I.T. Park Road, Bangalore 560 037
T: +91 80 28524466/77, F : +91 80 28524630
E: info@cmr.ac.in I www.cmr.ac.in

15CS64 – OPERATING SYSTEMS
IAT – 1 SOLUTION

1.a) What is an Operating System? Explain with user and system view points.[5]

• An OS is an intermediary between the user of the computer & the computer hardware.
• It provides a basis for application program & acts as an intermediary between user of computer & computer hardware.
• The purpose of an OS is to provide a environment in which the user can execute the program in a convenient & efficient manner.
• OS is an important part of almost every computer systems.

User Views:- The user view of the computer depends on the interface used.
• SOME USERS MAY USE PC'S. In this the system is designed so that only one user can utilize
the resources and mostly for ease of use where the attention is mainly on performances
and not on the resource utilization.
• Some users may use a terminal connected to a mainframe or minicomputers.
• Other users may access the same computer through other terminals. These users may share resources and exchange information. In this case the OS is designed to maximize resource utilization- so that all available CPU time, memory & I/O are used efficiently.
• Other users may sit at workstations, connected to the networks of other workstation and servers. In this case OS is designed to compromise between individual visibility &resource utilization.

System Views:-We can view system as resource allocator i.e. a computer system has many resources that may be used to solve a problem. The OS acts as a manager of these resources.

The OS must decide how to allocate these resources to programs and the users so that it can operate the computer system efficiently and fairly.
• A different view of an OS is that it need to control various I/O devices & user Programs i.e. an OS is a control program used to manage the execution of user program to prevent errors and improper use of the computer.
Resources can be either CPU Time, memory space, file storage space, I/O devices and so on.

b) Explain any 2 facilities provided for implementing the interacting process in programming language and operating system.[5]

User Operating-System Interface

2.2.1 Command Interpreter

- Gets and processes the next user request, and launches the requested programs.
- In some systems the CI may be incorporated directly into the kernel.
- More commonly the CI is a separate program that launches once the user logs in or otherwise accesses the system.
- UNIX, for example, provides the user with a choice of different shells, which may either be configured to launch automatically at login, or which may be changed on the fly. (Each of these shells uses a different configuration file of initial settings and commands that are executed upon startup.)
- Different shells provide different functionality, in terms of certain commands that are implemented directly by the shell without launching any external programs. Most provide at least a rudimentary command interpretation structure for use in shell script programming (loops, decision constructs, variables, etc.)
- An interesting distinction is the processing of wild card file naming and I/O re-direction. On UNIX systems those details are handled by the shell, and the program which is launched sees only a list of filenames generated by the shell from the wild cards. On a DOS system, the wild cards are passed along to the programs, which can interpret the wild cards as the program sees fit.

2. a) What are system calls? List the categories of system calls.[5]

- System provides interface between the process & the OS.
- The calls are generally available as assembly language instruction & certain system allow system calls to be made directly from a high level language program.
- Several languages have been defined to replace assembly language program.
- System calls occuring different ways depending on the computer. Sometime more information is needed to identify the desired system call. The exact type & amount of information needed may vary according to the Particular OS & call.

System calls may be grouped roughly into 5 categories

- Process control
  - end, abort
  - load, execute
  - create process, terminate process
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory
- File management
  - create file, delete file
  - open, close
  - read, write, reposition
  - get file attributes, set file attributes
- Device management
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices
- Information maintenance
  - get time or date, set time or date
  - get system data, set system data
  - get process, file, or device attributes
  - set process, file, or device attributes
- Communications
  - create, delete communication connection
  - send, receive messages
  - transfer status information
  - attach or detach remote devices

Figure 2.8 Types of system calls.

b) List 3 advantages of multiprocessor systems. Also, bring out the difference between graceful degradation and fault tolerance in this context.[5]

Advantages:

1. Increased Throughput

- By increasing no. of processors, we expect to get more work done in less time.

2. Economy of Scale

- These systems are cheaper because they can share
- → peripherals → mass-storage → power-supply.
- If many programs operate on same data, they will be stored on one disk & all processors can

share them.

3. Increased Reliability

- The failure of one processor will not halt the system.

Graceful Degradation:

The ability to continue providing service proportional to the level of surviving hardware.

Fault Tolerance:

Some systems go beyond graceful degradation is fault tolerance.

3. List and explain the functions and services of an operating system.[10]

An OS provides services for the execution of the programs and the users of such programs. The services provided by one OS may be different from other OS. OS makes the programming task easier.

The common services provided by the OS are

1. Program Execution:- The OS must able to load the program into memory & run that program. The program must end its execution either normally or abnormally.
2. I/O Operation:- A program running may require any I/O. This I/O may be a file or a specific
device users cant control the I/O device directly so the OS must provide a means for controlling I/O devices.
3. File System Interface:- Program need to read or write a file. The OS should provide permission for the creation or deletion of files by names.
4. Communication:- In certain situation one process may need to exchange information with another process. This communication May takes place in two ways.
a. Between the processes executing on the same computer.
b. Between the processes executing on different computer that are connected by anetwork. This communication can be implemented via shared memory or by OS.
5. Error Detection:- Errors may occur in CPU, I/O devices or in M/y H/w. The OS constantly needs to be aware of possible errors. For each type of errors the OS should take appropriate actions to ensure correct & consistent computing.

OS with multiple users provides the following services,

a. Resource Allocation:-When multiple users logs onto the system or when multiple jobs are running, resources must
be allocated to each of them. The OS manages different types of OS resources. Some resources may need some
special allocation codes & others may have some general request & release code.
b. Accounting:-We need to keep track of which users use how many & what kind of resources. This record keeping may be used for accounting. This accounting data may be used for statistics or billing. It can also be used to improve
system efficiency.
c. Protection:-Protection ensures that all the access to the system are controlled. Security starts with each user having authenticated to the system, usually by means of a password. External I/O devices must also be protected from invalid access. In multi process environment it is possible that one process may interface with the other or with the OS, so protection is required.

4. Explain process states with state transition diagram. Also, explain the PCB with a neat diagram.[10]

Process State

- Processes may be in one of 5 states, as shown in Figure 3.2 below.
    - **New** - The process is in the stage of being created.
    - **Ready** - The process has all the resources available that it needs to run, but the CPU is not currently working on this process's instructions.
    - **Running** - The CPU is working on this process's instructions.
    - **Waiting** - The process cannot run at the moment, because it is waiting for some resource to become available or for some event to occur. For example the process may be waiting for keyboard input, disk access request, inter-process messages, a timer to go off, or a child process to finish.
    - **Terminated** - The process has completed.
- The load average reported by the "w" command indicate the average number of processes in the "Ready" state over the last 1, 5, and 15 minutes, i.e. processes who have everything they need to run but cannot because the CPU is busy doing something else.
- Some systems may have other states besides the ones listed here.

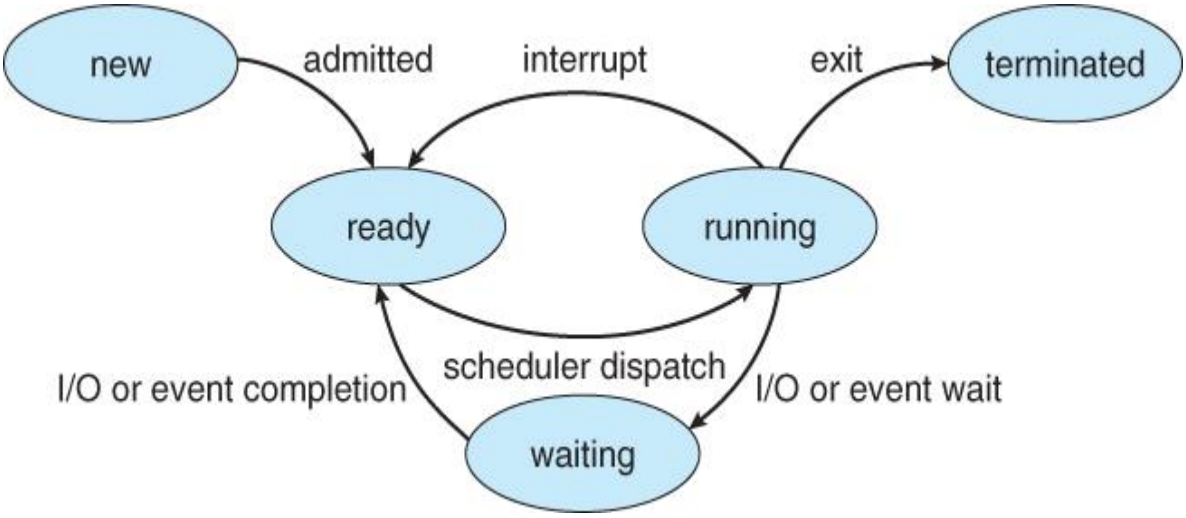


Figure - Diagram of process state

Process Control Block

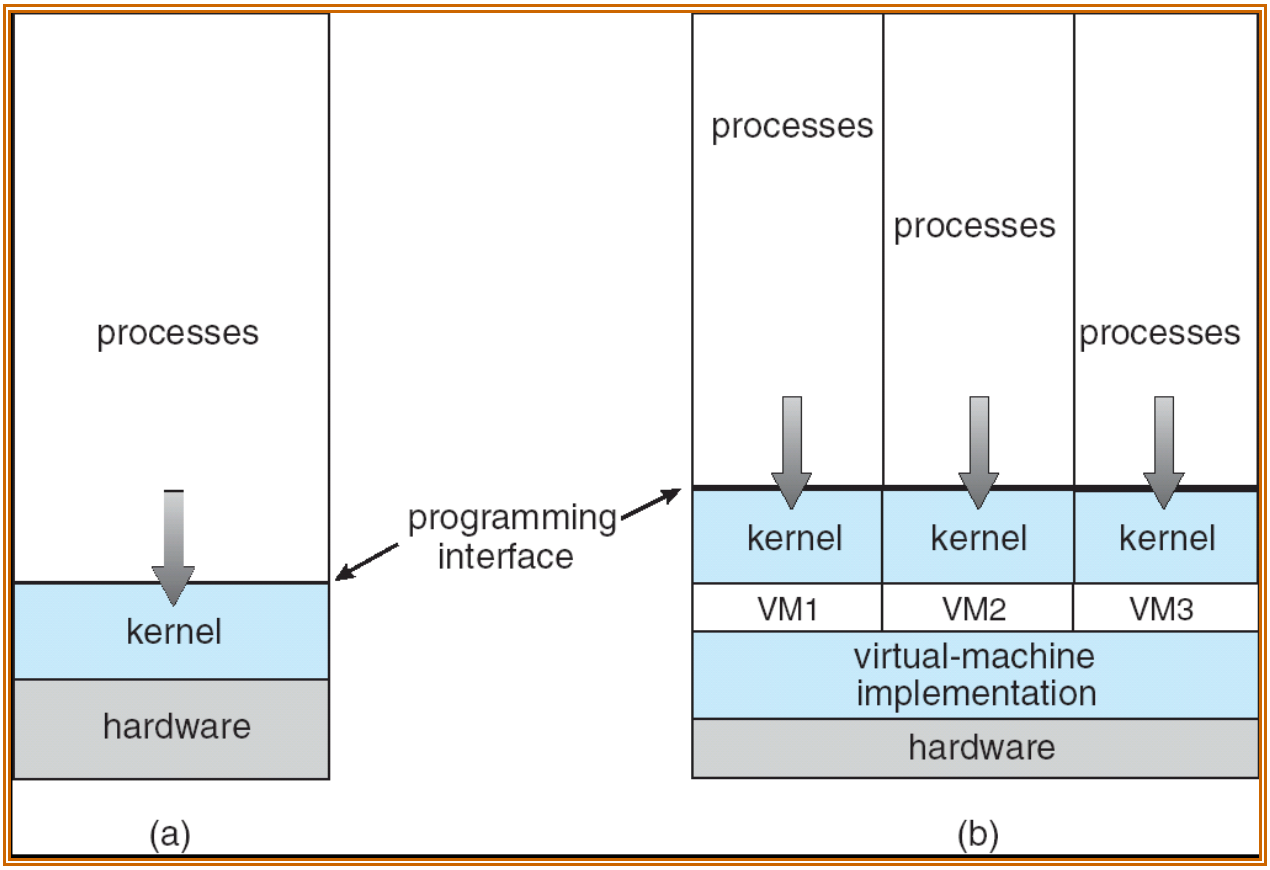
For each process there is a Process Control Block, PCB, which stores the following (types of) process-specific information, as illustrated in Figure 3.1. (Specific details may vary from system to system.)

- **Process State** - Running, waiting, etc., as discussed above.
- **Process ID**, and parent process ID.
- **CPU registers and Program Counter** - These need to be saved and restored when swapping processes in and out of the CPU.
- **CPU-Scheduling information** - Such as priority information and pointers to scheduling queues.
- **Memory-Management information** - E.g. page tables or segment tables.
- **Accounting information** - user and kernel CPU time consumed, account numbers, limits, etc.
- **I/O Status information** - Devices allocated, open file tables, etc.

Figure 3.3 - Process control block (PCB)

5. Define virtual machines. With a neat diagram explain the working of virtual machines and its benefits.[10]

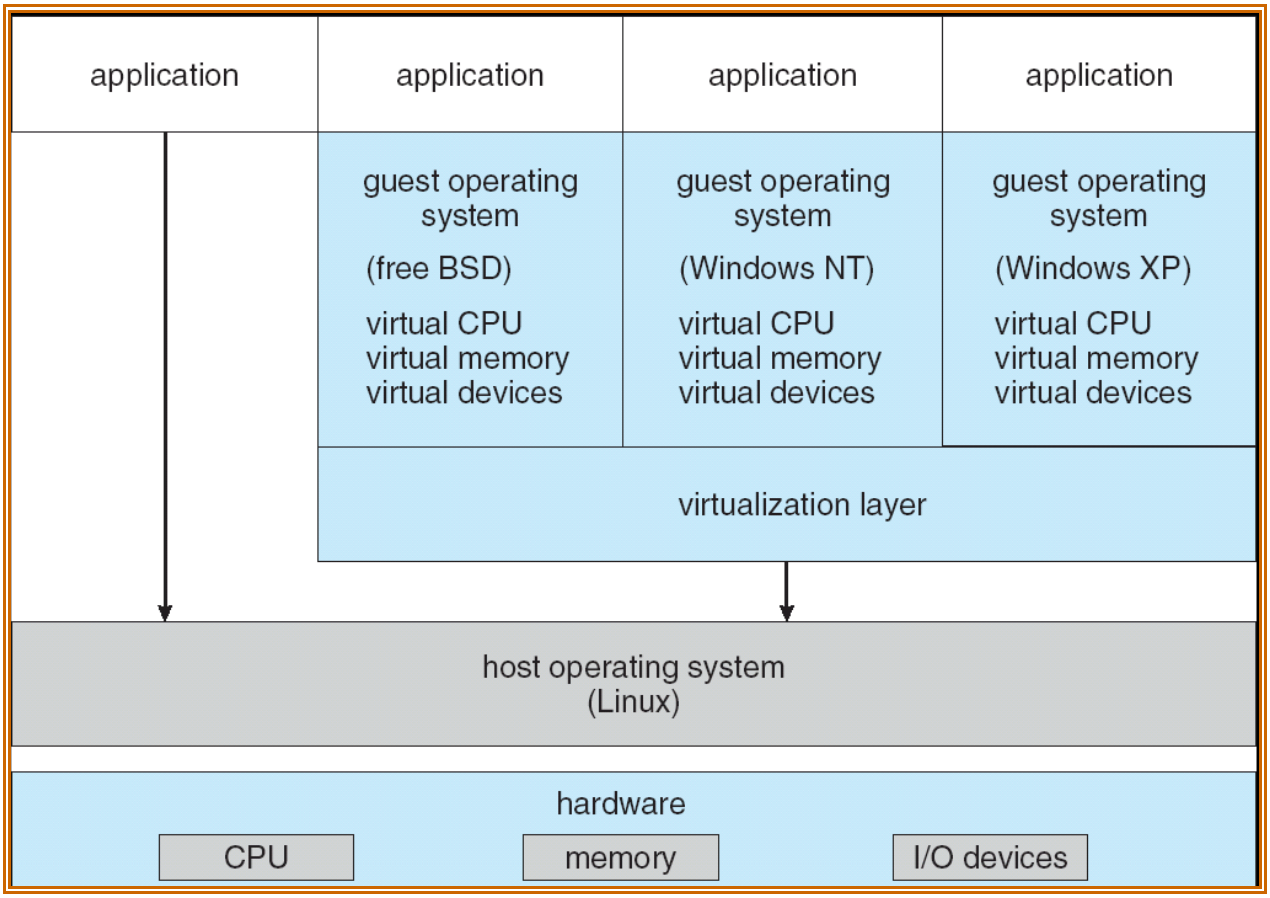
- A *virtual machine* takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware
- A virtual machine provides an interface *identical* to the underlying bare hardware
- The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory
- The resources of the physical computer are shared to create the virtual machines
    - CPU scheduling can create the appearance that users have their own processor
    - Spooling and a file system can provide virtual card readers and virtual line printers
    - A normal user time-sharing terminal serves as the virtual machine operator's console



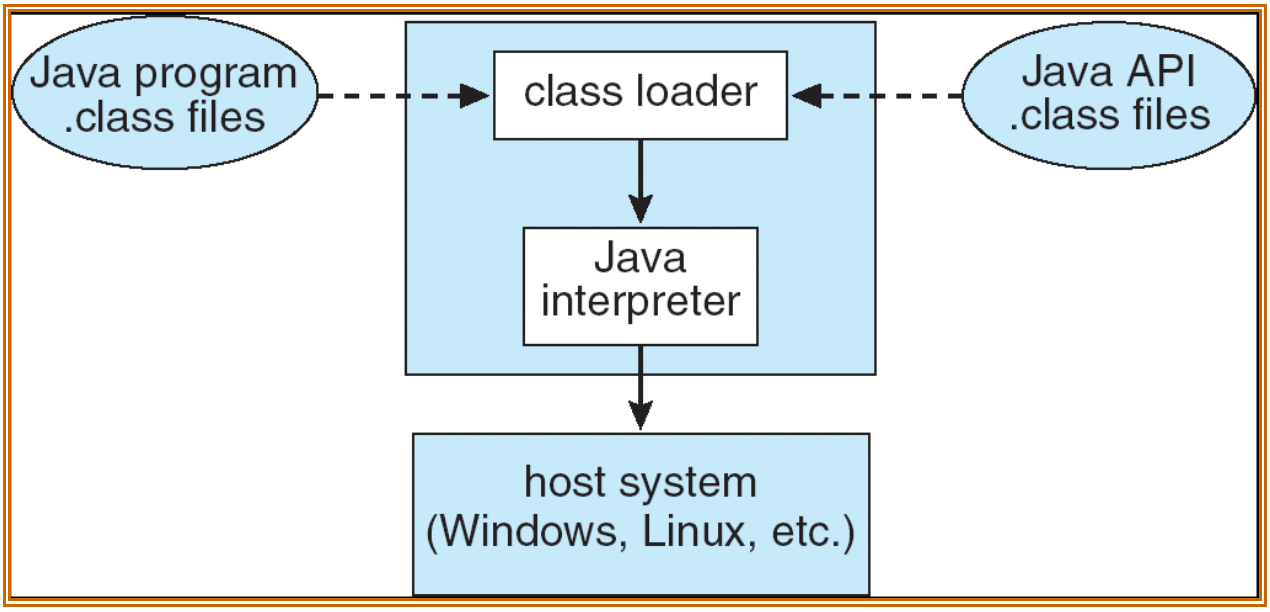
(a) Nonvirtual machine (b) virtual machine

- The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines. This isolation, however, permits no direct sharing of resources.
- A virtual-machine system is a perfect vehicle for operating-systems research and development. System development is done on the virtual machine, instead of on a physical machine and so does not disrupt normal system operation.
- The virtual machine concept is difficult to implement due to the effort required to provide an *exact* duplicate to the underlying machine

VMware Architecture



The Java Virtual Machine



6. a) Discuss 3 common ways of establishing a relationship between the user thread and the kernel thread.[5]

Multithreading Models

- There are two types of threads to be managed in a modern system: User threads and kernel threads.

- User threads are supported above the kernel, without kernel support. These are the threads that application programmers would put into their programs.
- Kernel threads are supported within the kernel of the OS itself. All modern OSes support kernel level threads, allowing the kernel to perform multiple simultaneous tasks and/or to service multiple kernel system calls simultaneously.
- In a specific implementation, the user threads must be mapped to kernel threads, using one of the following strategies.

Many-To-One Model

- In the many-to-one model, many user-level threads are all mapped onto a single kernel thread.
- Thread management is handled by the thread library in user space, which is very efficient.
- However, if a blocking system call is made, then the entire process blocks, even if the other user threads would otherwise be able to continue.
- Because a single kernel thread can operate only on a single CPU, the many-to-one model does not allow individual processes to be split across multiple CPUs.
- Green threads for Solaris and GNU Portable Threads implement the many-to-one model in the past, but few systems continue to do so today.

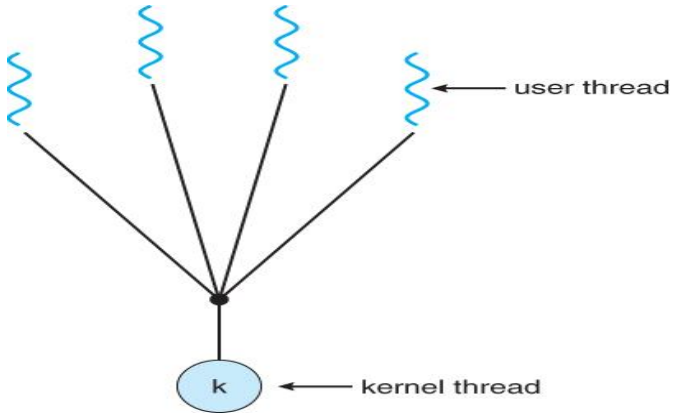


Figure Many-to-one model

One-To-One Model

- The one-to-one model creates a separate kernel thread to handle each user thread.
- One-to-one model overcomes the problems listed above involving blocking system calls and the splitting of processes across multiple CPUs.

- However the overhead of managing the one-to-one model is more significant, involving more overhead and slowing down the system.
- Most implementations of this model place a limit on how many threads can be created.
- Linux and Windows from 95 to XP implement the one-to-one model for threads.

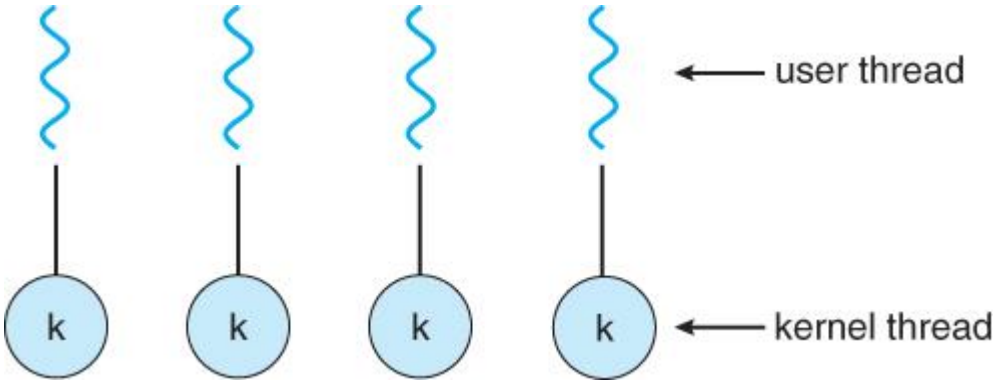


Figure - One-to-one model

4.3.3 Many-To-Many Model

- The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads, combining the best features of the one-to-one and many-to-one models.
- Users have no restrictions on the number of threads created.
- Blocking kernel system calls do not block the entire process.
- Processes can be split across multiple processors.
- Individual processes may be allocated variable numbers of kernel threads, depending on the number of CPUs present and other factors.

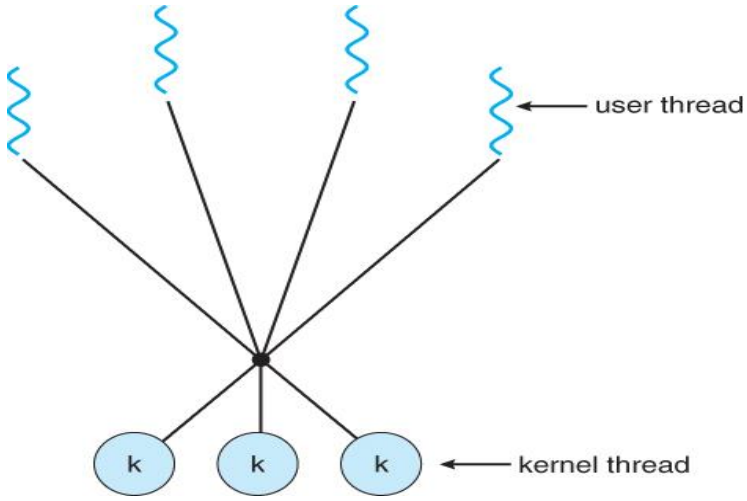
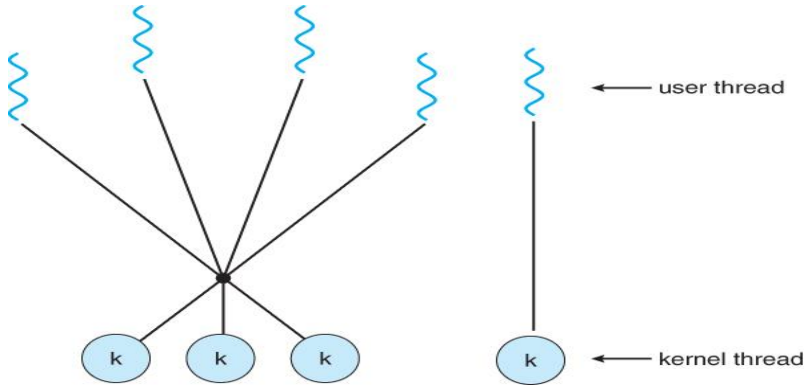


Figure 4.7 - Many-to-many model

- One popular variation of the many-to-many model is the two-tier model, which allows either many-to-many or one-to-one operation.
- IRIX, HP-UX, and Tru64 UNIX use the two-tier model, as did Solaris prior to Solaris 9.



Two-level model

b) Explain the Operating System operations.[5]

Operations on Processes

1. Process Creation and
2. Process Termination

Process Creation

• A process may create a new process via a create-process system-call.
• The creating process is called a parent-process.
The new process created by the parent is called the child-process (Sub-process).
• OS identifies processes by pid (process identifier), which is typically an integer-number.
• A process needs following resources to accomplish the task:
→ CPU time
→ memory and
→ I/0 devices.
• Child-process may
→ get resources directly from the OS or
→ get resources of parent-process. This prevents any process from overloading the system.

• Two options exist when a process creates a new process:
1. The parent & the children execute concurrently.
2. The parent waits until all the children have terminated.

• Two options exist in terms of the address-space of the new process:

1. The child-process is a duplicate of the parent-process (it has the same program and data as the parent).
2. The child-process has a new program loaded into it.

Process creation in UNIX

• In UNIX, each process is identified by its process identifier (pid), which is a unique integer.
• A new process is created by the fork() system-call (Figure 2.7 & 2.8).
• The new process consists of a copy of the address-space of the original process.
• Both the parent and the child continue execution with one difference:

1. The return value for the fork() is zero for the new (child) process.
2. The return value for the fork() is nonzero pid of the child for the parent-process.

• Typically, the exec() system-call is used after a fork() system-call by one of the two processes to replace the process's memory-space with a new program.
• The parent can issue wait() system-call to move itself off the ready-queue.

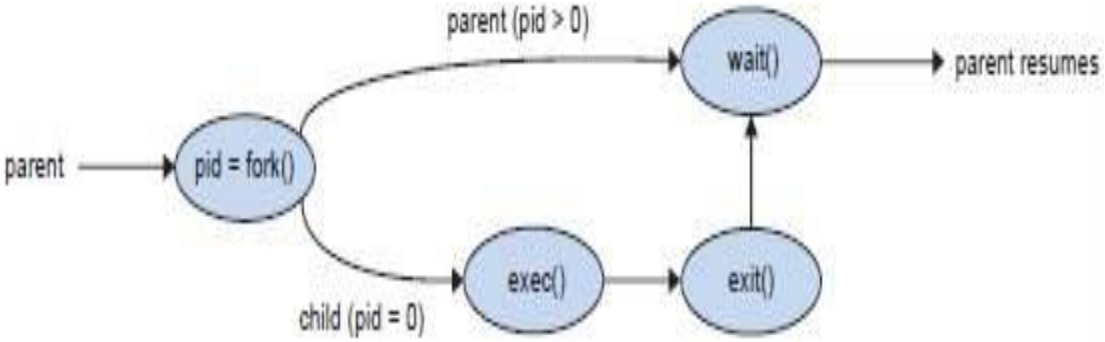
```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid;

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

    if (pid < 0) { /* error occurred */
       fprintf(stderr, "Fork Failed");
       return 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");
    }

    return 0;
}
```



Process Termination

• A process terminates when it executes the last statement (in the program).

• Then, the OS deletes the process by using exit() system-call.

• Then, the OS de-allocates all the resources of the process. The resources include

→ memory

→ open files and

→ I/0 buffers.

• Process termination can occur in following cases:

→ A process can cause the termination of another process via TerminateProcess() system-call.

→ Users could arbitrarily kill the processes.

• A parent terminates the execution of children for following reasons:

1. The child has exceeded its usage of some resources.
2. The task assigned to the child is no longer required.
3. The parent is exiting, and the OS does not allow a child to continue.

• In some systems, if a process terminates, then all its children must also be terminated. This phenomenon is referred to as cascading termination.

7. a) What is IPC? Explain direct and indirect communications with respect to message passing systems.[5]

Interprocess communication is a Mechanism for processes to communicate and to synchronize their actions.

Message system – processes communicate with each other without resorting to shared variables

IPC facility provides two operations:

- send(message) – message size fixed or variable
- receive(message)

If P and Q wish to communicate, they need to:

- establish a communication link between them

- exchange messages via send/receive

Implementation of communication link
- physical (e.g., shared memory, hardware bus)
- logical (e.g., logical properties)

Direct Communication
Processes must name each other explicitly:
- send (P, message) – send a message to process P
- receive(Q, message) – receive a message from process Q

Properties of communication link
- Links are established automatically
- A link is associated with exactly one pair of communicating processes
- Between each pair there exists exactly one link
- The link may be unidirectional, but is usually bi-directional

Indirect Communication
Messages are directed and received from mailboxes (also referred to as ports)
- Each mailbox has a unique id
- Processes can communicate only if they share a mailbox

send (A, message) – send a message to mailbox A
receive(A, message) – receive a message from mailbox B
Properties of communication link
- Link established only if processes share a common mailbox
- A link may be associated with many processes
- Each pair of processes may share several communication links
- Link may be unidirectional or bi-directional

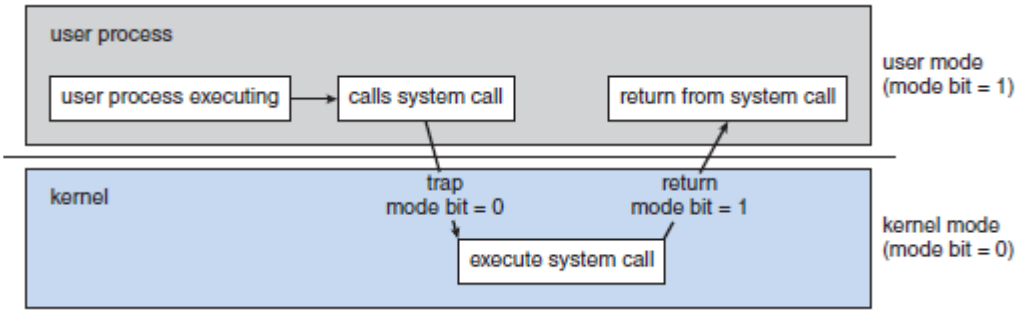
b) Explain dual mode operation in OS with a neat diagram.[5]

Dual Mode Operation
Problem: We must be able to differentiate between the execution of
- → OS code and
- → user-defined code.

Solution: Most computers provide hardware-support.
- We have two modes of operation (Figure 1.9):

1. User mode and
2. Kernel mode

- A mode bit is a bit added to the hardware of the computer to indicate the current mode:
  i.e. kernel (0) or user (1)



Working principle:

1. At system boot time, the hardware starts in kernel-mode.
2. The OS is then loaded and starts user applications in user-mode.
3. Whenever a trap or interrupt occurs, the hardware switches from user-mode to kernel-mode
(that is, changes the state of the mode bit to 0).
4. The system always switches to user-mode (by setting the mode bit to 1) before passing control to a user-program.

• Dual mode protects
→ OS from errant users and
→ errant users from one another.
• Privileged instruction is executed only in kernel-mode.
• If an attempt is made to execute a privileged instruction in user-mode, the hardware treats it as
illegal and traps it to the OS.
• A system calls are called by user-program to ask the OS to perform the tasks on behalf of the user program.

8. Briefly explain threading issues in multithreading.[10]

Threading Issues
The fork( ) and exec( ) System Calls

- Q: If one thread forks, is the entire process copied, or is the new process single-threaded?
- A: System dependant.
- A: If the new process execs right away, there is no need to copy all the other threads. If it doesn't, then the entire process should be copied.
- A: Many versions of UNIX provide multiple versions of the fork call for this purpose.

Signal Handling

- Q: When a multi-threaded process receives a signal, to what thread should that signal be delivered?
- A: There are four major options:
- Deliver the signal to the thread to which the signal applies.
- Deliver the signal to every thread in the process.
- Deliver the signal to certain threads in the process.
- Assign a specific thread to receive all signals in a process.
- The best choice may depend on which specific signal is involved.
- UNIX allows individual threads to indicate which signals they are accepting and which they are ignoring. However the signal can only be delivered to one thread, which is generally the first thread that is accepting that particular signal.
- UNIX provides two separate system calls, kill(pid, signal) and pthread_kill(tid,signal), for delivering signals to processes or specific threads respectively.
- Windows does not support signals, but they can be emulated using Asynchronous Procedure Calls (APCs). APCs are delivered to specific threads, not processes.

Thread Cancellation

- Threads that are no longer needed may be cancelled by another thread in one of two ways:
- Asynchronous Cancellation cancels the thread immediately.
- Deferred Cancellation sets a flag indicating the thread should cancel itself when it is convenient. It is then up to the cancelled thread to check this flag periodically and exit nicely when it sees the flag set.
- (Shared) resource allocation and inter-thread data transfers can be problematic with asynchronous cancellation.

Thread-Specific Data

- Most data is shared among threads, and this is one of the major benefits of using threads in the first place.
- However sometimes threads need thread-specific data also.
- Most major thread libraries (pThreads, Win32, Java) provide support for thread-specific data, known as thread-local storage or TLS. Note that this is more like static data than local variables,because it does not cease to exist when the function ends.

Scheduler Activations

- Many implementations of threads provide a virtual processor as an interface between the user thread and the kernel thread, particularly for the many-to-many or two-tier models.
- This virtual processor is known as a "Lightweight Process", LWP.
- There is a one-to-one correspondence between LWPs and kernel threads.
- The number of kernel threads available, (and hence the number of LWPs) may change dynamically.
- The application (user level thread library) maps user threads onto available LWPs.
- kernel threads are scheduled onto the real processor(s) by the OS.
- The kernel communicates to the user-level thread library when certain events occur (such as a thread about to block) via an upcall, which is handled in the thread library by an upcall handler. The upcall also provides a new LWP for the upcall handler to run on, which it can then use to reschedule the user thread that is about to become blocked. The OS will also issue upcalls when a thread becomes unblocked, so the thread library can make appropriate adjustments.
- If the kernel thread blocks, then the LWP blocks, which blocks the user thread.
- Ideally there should be at least as many LWPs available as there could be concurrently blocked kernel threads. Otherwise if all LWPs are blocked, then user threads will have to wait for one to become available.

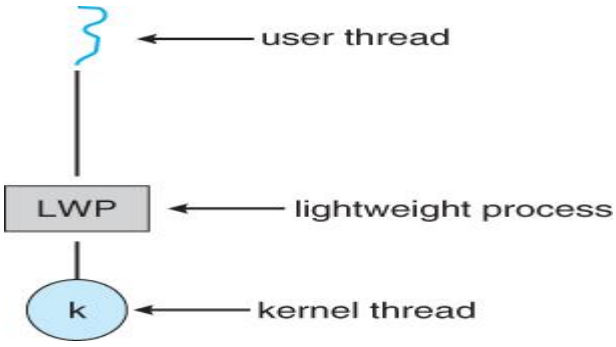


Figure - Lightweight process (LWP)