Indira Gandhi
National Open University
School of Computer and
Information Sciences

MCS-203
Operating Systems

Block

3

ADVANCED TOPICS IN OPERATING SYSTEMS

PROGRAMME/COURSE DESIGN COMMITTEE

Prof. Sanjeev K. Aggarwal, IIT, Kanpur
Shri Navneet Aggarwal
Trinity BPM, New Delhi
Prof. M. Balakrishnan, IIT, Delhi
Prof. Pandurangan, C., IIT, Madras
Ms. Bhoomi Gupta
Sirifort College of Computer and Technology Management, New Delhi
Shri Manoj Kumar Gupta
Keane India Ltd., New Delhi
Shri Sachin Gupta
Delhi Institute of Advanced Studies, New Delhi
Prof. Harish Karnick, IIT, Kanpur
Shri Anil Kumar
Amity School of Engineering and Technology New Delhi
Dr. Kapil Kumar, IIMT, Meerut
Dr. Sachin Kumar,
CCS University, Meerut
Ms. Manjulata
Amity School of Engineering and Technology New Delhi
Shri Ajay Rana
Amity School of Computer Sciences, Noida
Dr. Divya Sharma
Bharati College, Delhi
Shri Neeraj Sharma
Havard Institute of Management Technology Noida
Shri Sanjeev Thakur
Amity School of Computer Sciences, Noida
Shri Amrit Nath Thulal
Amity School of Engineering and Technology New Delhi
Dr. Om Vikas(Retd), Ex-Sr. Director, Ministry of ICT, Delhi
Shri Vishwakarma
Amity School of Engineering and Technology New Delhi
Prof (Retd) S. K. Gupta, IIT Delhi
Prof. T.V. Vijaya Kumar, Dean, SC&SS, JNU, New Delhi
Prof. Ela Kumar, Dean, CSE, IGDTUW, Delhi
Prof. Gayatri Dhingra, GVMITM, Sonipat
Sh. Milind Mahajani Vice President, Impressico Business Solutions, Noida, UP
Prof. V. V. Subrahmanyam, Director SOCIS, New Delhi
Prof. P. V. Suresh,
SOCIS, IGNOU, New Delhi
Dr. Shashi Bhushan
SOCIS, IGNOU, New Delhi
Shri Akshay Kumar, Associate Prof.
SOCIS, IGNOU, New Delhi
Shri M. P. Mishra, Associate Prof.
SOCIS, IGNOU, New Delhi
Dr. Sudhansh Sharma, Asst. Prof
SOCIS, IGNOU, New Delhi

BLOCK PREPARATION TEAM

Prof. D.P. Vidyarthi *(Content Editor)*
Jawaharlal Nehru University
New Delhi

Shri Sachin Gupta
Delhi Institute of Advanced Studies
New Delhi

Dr. Divya Sharma
Bharati College, University of Delhi,
Delhi

Prof. V.V. Subrahmanyam
(Course Writer-Unit-3)
SOCIS, IGNOU

Prof. A.K. Vermar *(Laguage Editor)*
Professor & Head (Retired)
Mass Communication, Delhi

Course Coordinator: Prof. V. V. Subrahmanyam

(The Units of Block-3, Unit 1 and Unit-2 were adopted from MCS-041 Operating Systems)

PRINT PRODUCTION

Mr. Tilak Raj
Assistant Registrar,
MPDD, IGNOU, New Delhi

July, 2021

ISBN : 978-93-91229-15-3

Further information on the Indira Gandhi National Open University courses may be obtained from the University's office at Maidan Garhi, New Delhi-110068.

Printed and published on behalf of the Indira Gandhi National Open University, New Delhi by the Registrar, MPDD, IGNOU, New Delhi

Laser Typesetting : Akashdeep Printers, 20-Ansari Road, Daryaganj, New Delhi-110002

Printed at : Akashdeep Printers, 20-Ansari Road, Daryaganj, New Delhi-110002

BLOCK INTRODUCTION

This block introduces you with the advanced topics in Operating Systems like the Mulitprocessor systems, Distributed Operating Systems and Mobile Operating Systems.

The block is organised into 3 units.

Unit 1 covers the OS technology to support the Multiprocessor systems. This unit discusses the different types of microprocessor interconnections, types of multiprocessor operating systems, functions of multiprocessor OS and synchronization aspects in the multiprocessor systems.

Unit 2 covers the OS technology to support the distributed computing. The unit starts with the discussion on the need of the distributed systems, design goals and design issues pertaining to the distributed systems. Also it focuses on various models of distributed systems, Remote procedure calls and Distributed shared memory.

Unit 3 covers the Mobile Operating Systems

ignou
THE PEOPLE'S
UNIVERSITY

UNIT 1 MULTIPROCESSOR SYSTEMS

Structure

1.0 INTRODUCTION

A multiprocessor system is a collection of a number of standard processors put together in an innovative way to improve the performance / speed of computer hardware. The main feature of this architecture is to provide high speed at low cost in comparison to uniprocessor. In a distributed system, the high cost of multiprocessor can be offset by employing them on a computationally intensive task by making it compute server. The multiprocessor system is generally characterised by - *increased system throughput* and *application speedup* - parallel processing.

Throughput can be improved, in a *time-sharing environment*, by executing a number of unrelated user processor on different processors in parallel. As a result a large number of different tasks can be completed in a unit of time without explicit user direction. On the other hand application speedup is possible by creating a multiple processor scheduled to work on different processors.

The scheduling can be done in two ways:

1) *Automatic means*, by parallelising compiler.

2) *Explicit-tasking approach*, where each programme submitted for execution is treated by the operating system as an independent process.

Multiprocessor operating systems aim to support high performance through multiple CPUs. An important goal is to make the number of CPUs transparent to the application. Achieving such transparency is relatively easy because the communication between different (parts of) applications uses the same primitives as those in multitasking uni-processor operating systems. The idea is that all communication is done by manipulating data at shared memory locations, and that we only have to protect that data against simultaneous access. Protection is done through synchronization primitives like semaphores and monitors.

In this unit we will study multiprocessor coupling, interconnections, types of multiprocessor operating systems and synchronization.

1.1 OBJECTIVES

After going through this unit, you should be able to:

- define a multiprocessor system;
- describe the architecture of multiprocessor and distinguish among various types of architecture;
- become familiar with different types of multiprocessor operating systems;
- discuss the functions of multiprocessors operating systems;
- describe the requirement of multiprocessor in Operating System, and
- discuss the synchronization process in a multiprocessor system.

1.2 MULTIPROCESSING AND PROCESSOR COUPLING

Multiprocessing is a general term for the use of two or more CPUs within a single computer system. There are many variations on this basic theme, and the definition of multiprocessing can vary with context, mostly as a function of how CPUs are defined. The term multiprocessing is sometimes used to refer to the execution of multiple concurrent software processes in a system as opposed to a single process at any one instant. However, the term multiprogramming is more appropriate to describe this concept, which is implemented mostly in software, whereas multiprocessing is more appropriate to describe the use of multiple hardware CPUs. A system can be both multiprocessing and multiprogramming, only one of the two, or neither of the two.

Processor Coupling is a type of multiprocessing. Let us see in the next section.

Processor Coupling

Tightly-coupled multiprocessor systems contain multiple CPUs that are connected at the bus level. These CPUs may have access to a central shared memory (SMP), or may participate in a memory hierarchy with both local and shared memory (NUMA). *The IBM p690 Regatta* is an example of a high end SMP system. Chip multiprocessors, also known as multi-core computing, involve more than one

processor placed on a single chip and can be thought of the most extreme form of tightly-coupled multiprocessing. Mainframe systems with multiple processors are often tightly-coupled.

Loosely-coupled multiprocessor systems often referred to as clusters are based on multiple standalone single or dual processor commodity computers interconnected via a high speed communication system. *A Linux Beowulf* is an example of a loosely-coupled system.

Tightly-coupled systems perform better and are physically smaller than loosely-coupled systems, but have historically required greater initial investments and may depreciate rapidly; nodes in a loosely-coupled system are usually inexpensive commodity computers and can be recycled as independent machines upon retirement from the cluster.

Power consumption is also a consideration. Tightly-coupled systems tend to be much more energy efficient than clusters. This is due to fact that considerable economies can be realised by designing components to work together from the beginning in tightly-coupled systems, whereas loosely-coupled systems use components that were not necessarily intended specifically for use in such systems.

1.3 MULTIPROCESSOR INTERCONNECTIONS

As learnt above, a multiprocessor can speed-up and can improve throughput of the computer system architecturally. The whole architecture of multiprocessor is based on the principle of parallel processing, which needs to synchronize, after completing a stage of computation, to exchange data. For this the multiprocessor system needs an efficient mechanism to communicate. This section outlines the different architecture of multiprocessor interconnection, including:

- Bus-oriented System
- Crossbar-connected System
- Hyper cubes
- Multistage Switch-based System.

1.3.1 Bus-oriented System

Figure1 illustrates the typical architecture of a bus oriented system. As indicated, processors and memory are connected by a common bus. Communication between processors (P1, P2, P3 and P4) and with globally shared memory is possible over a shared bus. Other then illustrated many different schemes of bus-oriented system are also possible, such as:

1) Individual processors may or may not have private/cache memory.
2) Individual processors may or may not attach with input/output devices.
3) Input/output devices may be attached to shared bus.
4) Shared memory implemented in the form of multiple physical banks connected to the shared bus.

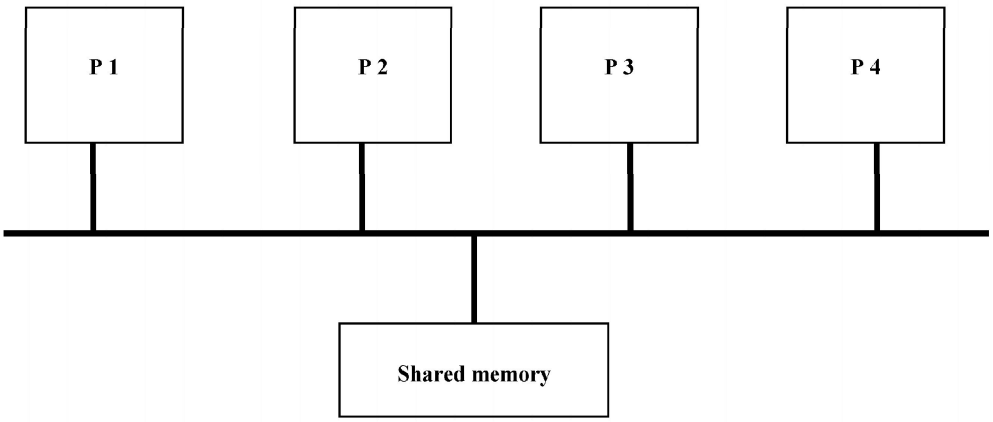

Figure 1. Bus-oriented multiprocessor interconnection

The above architecture gives rise to a problem of *contention* at two points, one is shared bus itself and the other is shared memory. Employing private/cache memory in either of two ways, explained below, the problem of contention could be reduced;

- with shared memory; and
- with cache associated with each individual processor

1) **With shared memory**

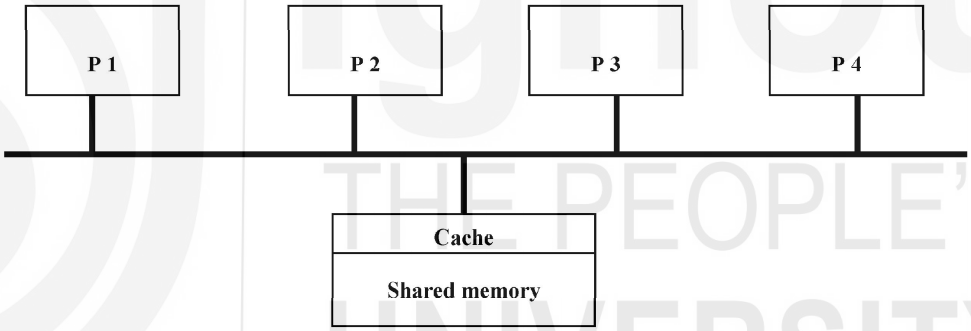

Figure 2: With shared Memory

2) **Cache associated with each individual processor.**

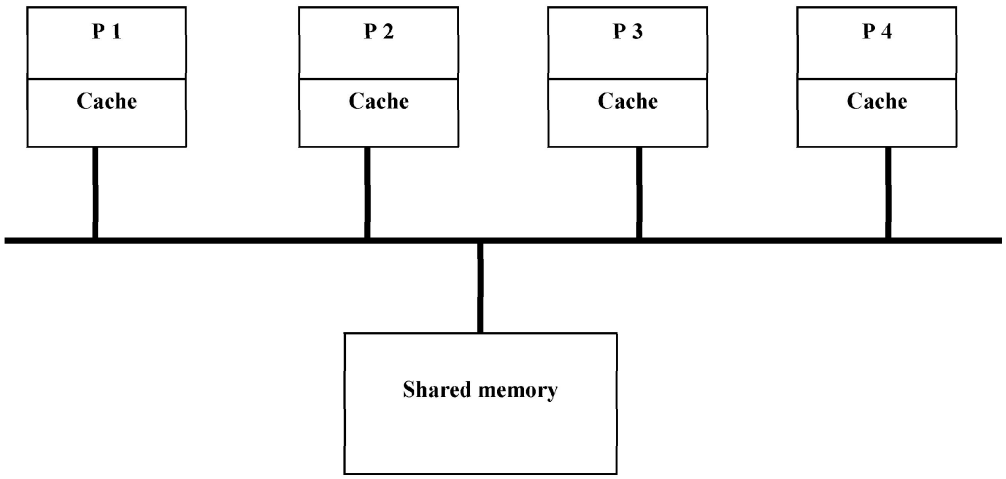

Figure 3: Individual processors with cache

The second approach where the cache is associated with each individual processor is the most popular approach because it reduces contention more effectively.

Cache attached with processor an capture many of the local memory references for example, with a cache of 90% hit ratio, a processor on average needs to access the shared memory for 1 (one) out of 10 (ten) memory references because 9 (nine) references are already captured by private memory of processor. In this case where memory access is uniformly distributed a 90% cache hit ratio can support the shared bus to speed-up 10 times more than the process without cache. The negative aspect of such an arrangement arises when in the presence of multiple cache the shared writable data are cached. In this case Cache Coherence is maintained to consistency between multiple physical copies of a single logical datum with each other in the presence of update activity. Yes, the cache coherence can be maintained by attaching additional hardware or by including some specialised protocols designed for the same but unfortunately this special arrangement will increase the bus traffic and thus reduce the benefit that processor caches are designed to provide.

Cache coherence refers to the integrity of data stored in local caches of a shared resource. Cache coherence is a special case of memory coherence.

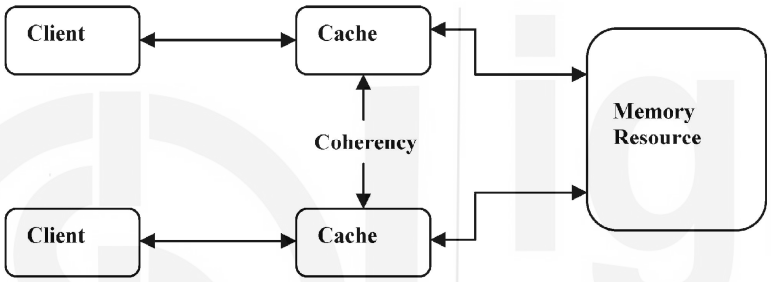

Figure 4: Multiple Caches of Common Resource

When clients in a system, particularly CPUs in a multiprocessing system, maintain caches of a common memory resource, problems arise. Referring to the *Figure 4*, if the top client has a copy of a memory block from a previous read and the bottom client changes that memory block, the top client could be left with an invalid cache of memory without it knowing any better. Cache coherence is intended to manage such conflicts and maintain consistency between cache and memory.

Let us discuss some techniques which can be employed to decrease the impact of bus and memory saturation in bus-oriented system.

1) **Wider Bus Technique**: As suggested by its name a bus is made wider so that more bytes can be transferred in a single bus cycle. In other words, a wider parallel bus increases the bandwidth by transferring more bytes in a single bus cycle. The need of bus transaction arises when lost or missed blocks are to fetch into his processor cache. In this transaction many system supports, block-level reads and writes of main memory. In the similar way, a missing block can be transferred from the main memory to his processor cache only by a single main-memory (block) read action. The advantage of *block level access to memory* over *word-oriented busses* is the amortization of overhead addressing, acquisition and arbitration over a large number of items in a block. Moreover, it also enjoyed specialised burst bus cycles, which makes their transport more efficient.

2) **Split Request/Reply Protocols**: The memory request and reply are split into two individual works and are treated as separate bus transactions. As soon as

a processor requests a block, the bus released to other user, meanwhile it takes for memory to fetch and assemble the related group of items.

The bus-oriented system is the simplest architecture of multiprocessor system. In this way it is believed that in a tightly coupled system this organisation can support on the order of 10 processors. However, the two contention points (the bus and the shared memory) limit the scalability of the system.

1.3.2 Crossbar-Connected System

Crossbar-connected System (illustrated in *Figure 5*) is a grid structure of processor and memory modules. The every cross point of grid structure is attached with switch. By looking at the *Figure* it seems to be a contention free architecture of multiprocessing system, it shows simultaneous access between processor and memory modules as *N* number of processors are provided with *N* number of memory modules. Thus each processor accesses a different memory module. Crossbar needs N^2 switches for fully connected network between processors and memory. Processors may or may not have their private memories. In the later case the system supports uniform-memory-access.

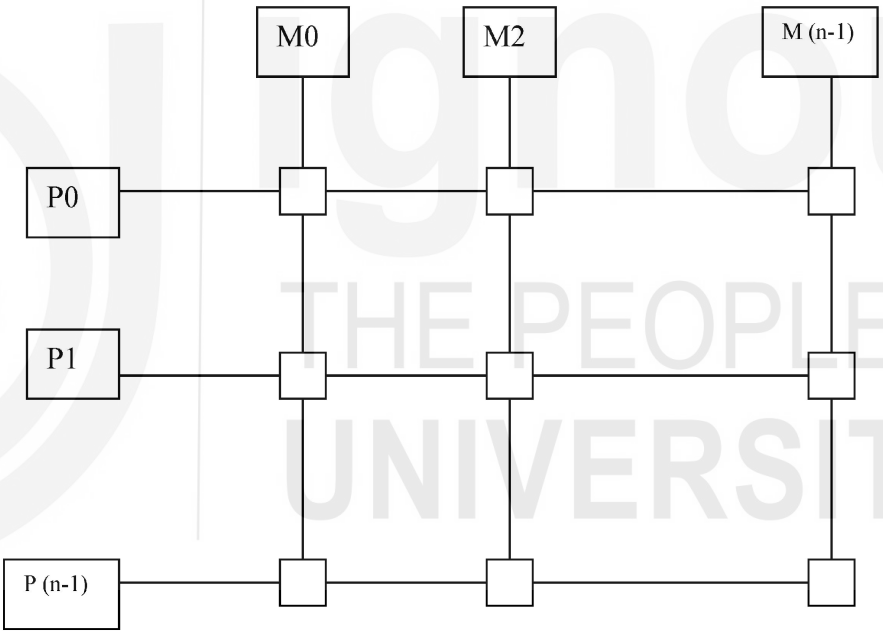

Figure 5: Crossbar-connected system

Where P= processor, M=Memory Module and =Switch

But Unfortunately this system also faces the problem of contention when,

1) More than two processors (P1 and P2) attempt to access the one memory module (M1) at the same time. In this condition, contention can be avoided by making any one processor (P1) deferred until the other one (P2) finishes this work or left the same memory module (M1) free for processor P1.

2) More than two processor attempts to access the same memory module. This problem cannot be solved by above-mentioned solution.

Thus in crossbar connection system the problem of contention occurs on at memory neither at processor nor at interconnection networks level. The solution of this problem includes quadratic growth of its switches as well as its complexity

level. Not only this, it will make the whole system expensive and also limit the scalability of the system.

1.3.3 Hypercubes System

Hypercube base architecture is not an old concept of multiprocessor system. This architecture has some advantages over other architectures of multiprocessing system. As the *Figure 6* indicates, the system topology can support large number of processors by providing increasing interconnections with increasing complexity. In an *n-degree* hypercube architecture, we have:

1) 2^n nodes (Total number of processors)
2) Nodes are arranged in n-dimensional cube, i.e. each node is connected to *n* number of nodes.
3) Each node is assigned with a unique address which lies between 0 to$2^n - 1$
4) The adjacent nodes (n−1) are differing in 1 bit and the *nth* node is having maximum '*n*' internode distance.

Let us take an example of **3-degree hypercube** to understand the above structure:

1) 3-degree hypercube will have 2^n nodes i.e., $2^3 = 8$ nodes
2) Nodes are arranged in 3-dimensional cube, that is, each node is connected to 3 number of nodes.
3) Each node is assigned with a unique address, which lies between 0 to 7 ($2^n - 1$), i.e., 000, 001, 010, 011, 100, 101, 110, 111
4) Two adjacent nodes differing in 1 bit (001, 010) and the 3^{rd} (n^{th}) node is having maximum '3'internode distance (100).

Hypercube provide a good basis for scalable system because its communication length grows logarithmically with the number of nodes. It provides a bi-directional communication between two processors. It is usually used in loosely coupled multiprocessor system because the transfer of data between two processors goes through several intermediate processors. The longest internodes delay is *n-degree*. To increase the input/output bandwidth the input/output devices can be attached with every node (processor).

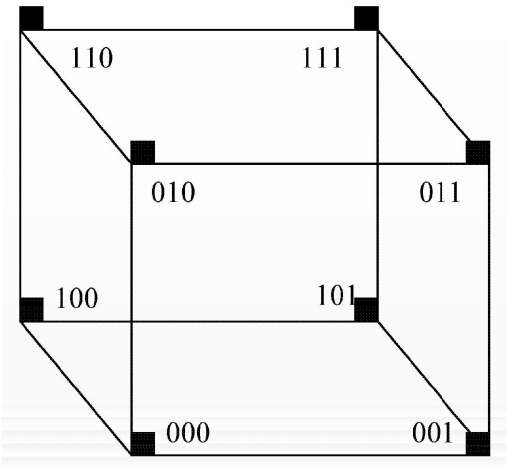

Figure 6: Hypercubes System

1.3.4 Multistage Switch Based system

Multistage Switch Based System permits simultaneous connection between several input-output pairs. It consists of several stages of switches which provide multistage interconnection network. A *N* input-output connections contains K= $\log_2 N$ stages of $N/2$ switches at each stage. In simple words, *N*N* processor-memory interconnection network requires $(N/2) \times \log_2 N$ switches.

For example, in a 8×8 process-memory interconnection network requires $(8/2 * \log_2 8) = 4*3 = 12$ switches. Each switch acts as 2×2 crossbar.

This network can connect any processor to any memory module by making appropriate connection of each of the '*K*' stages. The binary address of processor and memory gives binary string routing path between module pairs of input and output. the routing path *id* decided by on the basis of destination binary address, that the sender includes with each request for connection. Various combinations of paths between sources to destination are possible by using different switch function (straight, swap, copy, etc.)

In multistage switch based system all inputs are connected to all outputs in such a way that no two-processor attempt to access the same memory at the same time. But the problem of contention, at a switch, arises when some memory modules are contested by some fixed processor. In this situation only one request is allowed to access and rest of the requests are dropped. The processor whose requests were dropped can retry the request or if buffers are attached with each switch the rejected request is forwarded by buffer automatically for transmission. This Multistage interconnection networks also called store-and-forward networks.

1.4 TYPES OF MULTIPROCESSOR OPERATING SYSTEMS

The multiprocessor operating systems are complex in comparison to multiprograms on an uniprocessor operating system because multiprocessor executes tasks concurrently.

Therefore, it must be able to support the concurrent execution of multiple tasks to increase processors performance.

Depending upon the control structure and its organisation the three basic types of multiprocessor operating system are:

1) Separate supervisor
2) Master-slave
3) Symmetric Supervision

1.4.1 Separate Supervisors

In separate supervisor system each process behaves independently. Each system has its own operating system which manages local input/output devices, file system and memory well as keeps its own copy of kernel, supervisor and data structures, whereas some common data structures also exist for communication between

processors. The access protection is maintained, between processor, by using some synchronization mechanism like semaphores. Such architecture will face the following problems:

1) Little coupling among processors.
2) Parallel execution of single task.
3) During process failure it degrades.
4) Inefficient configuration as the problem of replication arises between supervisor/kernel/data structure code and each processor.

1.4.2 Master-Slave

In master-slave, out of many processors one processor behaves as a master whereas others behave as slaves. The master processor is dedicated to executing the operating system. It works as scheduler and controller over slave processors. It schedules the work and also controls the activity of the slaves. Therefore, usually data structures are stored in its private memory. Slave processors are often identified and work only as a schedulable pool of resources, in other words, the slave processors execute application programmes.

This arrangement allows the parallel execution of a single task by allocating several subtasks to multiple processors concurrently. Since the operating system is executed by only master processors this system is relatively simple to develop and efficient to use. Limited scalability is the main limitation of this system, because the master processor become a bottleneck and will consequently fail to fully utilise slave processors.

1.4.3 Symmetric

In symmetric organisation all processors configuration are identical. All processors are autonomous and are treated equally. To make all the processors functionally identical, all the resources are pooled and are available to them. This operating system is also symmetric as any processor may execute it. In other words there is one copy of kernel that can be executed by all processors concurrently. To that end, the whole process is needed to be controlled for proper interlocks for accessing scarce data structure and pooled resources.

The simplest way to achieve this is to treat the entire operating system as a critical section and allow only one processor to execute the operating system at one time. This method is called ‘floating master’ method because in spite of the presence of many processors only one operating system exists. The processor that executes the operating system has a special role and acts as a master. As the operating system is not bound to any specific processor, therefore, it floats from one processor to another.

Parallel execution of different applications is achieved by maintaining a queue of ready processors in shared memory. Processor allocation is then reduced to assigning the first ready process to first available processor until either all processors are busy or the queue is emptied. Therefore, each idled processor fetches the next work item from the queue.

1.5 MULTIPROCESSOR OPERATING SYSTEM - FUNCTIONS AND REQUIREMENTS

A multiprocessor operating system manages all the available resources and schedule functionality to form an abstraction it will facilitates programme execution and interaction with users.

A processor is one of the important and basic types of resources that need to be manage. For effective use of multiprocessors the processor scheduling is necessary.

Processors scheduling undertakes the following tasks:

(i) Allocation of processors among applications in such a manner that will be consistent with system design objectives. It affects the system throughput. Throughput can be improved by co-scheduling several applications together, thus availing fewer processors to each.

(ii) Ensure efficient use of processors allocation to an application. This primarily affects the speedup of the system.

The above two tasks are somehow conflicting each other because maximum speedup needs dedication of a large proportion of a systems processors to a single application which will decrease throughput of the system. Due to the difficulties of automating the process, the need for explicit programmer direction arises to some degree. Generally the language translators and preprocessors provide support for explicit and automatic parallelism. The two primary facets of OS support for multiprocessing are:

(i) Flexible and efficient interprocess and interprocessor synchronization mechanism, and

(ii) Efficient creation and management of a large number of threads of activity, such as processes or threads.

The latter aspect is important because parallelism is often accomplished by splitting an application into separate, individually executable subtasks that may be allocated to different processors.

The **Memory management** is the second basic type of resource that needs to be managed. In multiprocessors system memory management is highly dependent on the architecture and inter-connection scheme.

- In loosely coupled systems memory is usually handled independently on a pre-processor basis whereas in multiprocessor system shared memory may be simulated by means of a message passing mechanism.
- In shared memory systems the operating system should provide a flexible memory model that facilitates safe and efficient access to share data structures and synchronization variables.

A multiprocessor operating system should provide a hardware independent, unified model of shared memory to facilitate porting of applications between different multiprocessor environments. The designers of the mach operating system exploited the duality of memory management and inter-process communication.

The third basic resource is **Device Management** but it has received little attention in multiprocessor systems to date, because earlier the main focus point is speedup of compute intensive application, which generally do not generate much input/output after the initial loading. Now, multiprocessors are applied for more balanced general-purpose applications, therefore, the input/output requirement increases in proportion with the realised throughput and speed.

1.6 MULTIPROCESSOR SYNCHRONIZATION

Multiprocessor system facilitates parallel program execution and read/write sharing of data and thus may cause the processors to concurrently access location in the shared memory. Therefore, a correct and reliable mechanism is needed to serialize this access. This is called synchronization mechanism. The mechanism should make access to a shared data structure appear atomic with respect to each other. In this section, we introduce some new mechanism and techniques suitable for synchronization in multiprocessors.

1.6.1 Test-and-Set

The test-and-set instruction automatically reads and modifies the contents of a memory location in one memory cycle. It is as follows:

function test-and-set *(var m: Boolean); Boolean;*

begin

test-and set:=m;

m:=rue

end;

The test-and-set instruction returns the current value of variable m (memory location) and sets it to true. This instruction can be used to implement *P* and *V* operations (Primitives) on a binary semaphore, *S*, in the following way (*S* is implemented as a memory location):

P(S): while Test-and-Set (S) do nothing;

V(S): S:=false;

Initially, *S* is set to false. When a *P(S)* operation is executed for the first time, test-and-set*(S)* returns a false value (and sets *S* to true) and the "while" loop of the *P(S)* operation terminates. All subsequent executions of *P(S)* keep looping because *S* is true until a *V(S)* operation is executed.

1.6.2 Compare-and-Swap

The compare and swap instruction is used in the optimistic synchronization of concurrent updates to a memory location. This instruction is defined as follows (*r1and r2* are to registers of a processor and *m* is a memory location):

function test-and-set *(var m: Boolean); Boolean;*

var temp: integer;

begin

temp:=m;

```
    if temp = r1 then  {m:= r2;z:=1}
    else [r1:= temp; z:=0}
end;
```

If the contents of *r1* and *m* are identical, this instruction assigns the contents of *r2* to *m* and sets *z* to *1*. Otherwise, it assigns the contents of *m* to *r1* and set *z* to *0*. Variable *z* is a flag that indicates the success of the execution. This instruction can be used to synchronize concurrent access to a shared variable.

1.6.3 Fetch-and-Add

The fetch and add instruction is a multiple operation memory access instruction that automatically adds a constant to a memory location and returns the previous contents of the memory location. This instruction is defined as follows:

```
Function Fetch-and-add (m: integer; c: integer);
Var temp: integer;
Begin
    Temp:= m;
    M:= m + c;
    Return (temp)
end;
```

This instruction is executed by the hardware placed in the interconnection network not by the hardware present in the memory modules. When several processors concurrently execute a fetch-and-add instruction on the same memory location, these instructions are combined in the network and are executed by the network in the following way:

- A single increment, which is the sum of the increments of all these instructions, is added to the memory location.
- A single value is returned by the network to each of the processors, which is an arbitrary serialisation of the execution of the individual instructions.
- If a number of processors simultaneously perform fetch-and-add instructions on the same memory location, the net result is as if these instructions were executed serially in some unpredictable order.

The fetch-and-add instruction is powerful and it allows the implementation of P and V operations on a general semaphore, S, in the following manner:

```
P(S): while (Fetch-add-Add(S, -1)< 0) do
begin
    Fetch-and-Add (S, 1);
    while (S<0) do nothing;
end;
```

The outer "while-do" statement ensures that only one processor succeeds in decrementing S to 0 when multiple processors try to decrement variable S. All the unsuccessful processors add 1 back to S and try again to decrement it. The "while-do" statement forces an unsuccessful processor to wait (before retrying)

until S is greater then 0.

V(S): *Fetch-and-Add (S, 1).*

☞ **Check Your Progress 1**

1) What is the difference between a loosely coupled system and a tightly coupled system? Give examples.

.....

.....

.....

.....

2) What is the difference between symmetric and asymmetric multiprocessing?

.....

.....

.....

.....

1.7 SUMMARY

Multiprocessors systems architecture provides higher computing power and speed. It consists of multiple processors that putting power and speed to the system. This system can execute multiple tasks on different processors concurrently. Similarly, it can also execute a single task in parallel on different processors. The design of interconnection networks includes the bus, the cross-bar switch and the multistage interconnection networks. To support parallel execution this system must effectively schedule tasks to various processors. And also it must support primitives for process synchronization and virtual memory management. The three basic configurations of multiprocessor operating systems are: Separate supervisors, Master/slave and Symmetric. A multiprocessor operating system manages all the available resources and schedule functionality to form an abstraction. It includes Process scheduling, Memory management and Device management. Different mechanism and techniques are used for synchronization in multiprocessors to serialize the access of pooled resources. In this section we have discussed Test-and-Set, Compare-and-Swap and Fetch-and-Add techniques of synchronization.

1.8 SOLUTIONS/ANSWERS

1) One feature that is commonly characterizing tightly coupled systems is that they share the clock. Therefore, multiprocessors are typically tightly coupled but distributed workstations on a network are not.

   Another difference is that: in a tightly-coupled system, the delay experienced when a message is sent from one computer to another is short, and data rate is high; that is, the number of bits per second that can be transferred is large.

In a loosely-coupled system, the opposite is true: the intermachine message delay is large and the data rate is low.

For example, two CPU chips on the same printed circuit board and connected by wires etched onto the board are likely to be tightly coupled, whereas two computers connected by a 2400 bit/sec modem over the telephone system are certain to be loosely coupled.

2) The difference between symmetric and asymmetric multiprocessing: all processors of symmetric multiprocessing are peers; the relationship between processors of asymmetric multiprocessing is a master-slave relationship. More specifically, each CPU in symmetric multiprocessing runs the same copy of the OS, while in asymmetric multiprocessing, they split responsibilities typically, therefore, each may have specialised (different) software and roles.

1.9 FURTHER READINGS

1) Singhal Mukesh and Shivaratri G. Niranjan, *Advanced Concepts in Operating Systems*, TMGH, 2003, New Delhi.

2) Hwang, K. and F Briggs, *Multiprocessors Systems Architectures*, McGraw-Hill, New York.

3) Milenkovic, M., *Operating Systems: Concepts and Design*, TMGH, New Delhi.

4) Silberschatz, A., J. Peterson, and D. Gavin, *Operating Systems Concepts*, 3rd ed., Addison Wesley, New Delhi.

UNIT 2 DISTRIBUTED OPERATING SYSTEMS

Structure

2.0 INTRODUCTION

In the earlier unit we have discussed the Multiprocessor systems. In the process coupling we have come across the tightly coupled systems and loosely coupled systems. In this unit we concentrate on the loosely coupled systems called as the distributed systems. Distributed computing is the process of aggregating the power of several computing entities to collaboratively run a computational task in a transparent and coherent way, so that it appears as a single, centralised system.

The easiest way of explaining what distributed computing is all about, is by naming a few of its properties:

- Distributed computing consists of a network if more or less independent or autonomous nodes.
- The nodes do *not* share primary storage (i.e., RAM) or secondary storage (i.e., disk) - in the sense that the nodes cannot directly access another node's disk or RAM. Think about it in contrast to a multiprocessors machine where the different "nodes" (CPUs)) share the same RAM and secondary storage by using a common bus.
- A well designed distributed system does not crash if a node goes down.
- If you are to perform a computing task which is parallel in nature, scaling your system is a lot cheaper by adding extra nodes, compared to getting a faster single machine. Of course, if your processing task is highly non-parallel (every result depends on the previous), using a distributed computing system may not be very beneficial.

Before going into the actual details of the distributed systems let us study how distributed computing evolved.

2.1 OBJECTIVES

After going through this unit, you should be able to:

- define a distributed system, key features and design goals of it;
- explain the design issues involved in the distributed systems;
- describe the models of distributed system structure;
- explain the mutual exclusion in distributed systems, and
- define RPC, its usage and limitations.

2.2 HISTORY OF DISTRIBUTED COMPUTING

Distributed computing began around 1970 with the emergence of two technologies:

- minicomputers, then workstations, and then PCs.
- computer networks (eventually Ethernet and the Internet).

With the minicomputer (e.g., Digital's PDP-11) came the timesharing operating system (e.g., MULTICS, Unix, RSX, RSTS) - many users using the same machine but it looks to the users as if they each have their own machine.

The problem with mini-computers was that they were slower than the mainframes made by IBM, Control Data, Univac, etc. As they became popular they failed to scale to large number of users as the mainframes could. The way to scale mini-computers was to buy more of them. The trend toward cheaper machines made the idea of having many minis as a feasible replacement for a single mainframe and made it possible to contemplate a future computing environment where every user had their own computer on their desk, which is a computer workstation.

Work on the first computer workstation began in 1970 at Xerox Corporation's Palo Alto Research Center (PARC). This computer was called the Alto. Over the next 10 years, the computer system's group at PARC would invent almost everything that's interesting about the computer workstations and personal computers we use today. The Alto introduced ideas like the workstation, bit-mapped displays (before that computer interfaces were strictly character-based) and the mouse.

Other innovations that came from PARC from 1970 to 1980 include Ethernet, the first local-area network, window- and icon-based computing (the inspiration for the Apple Lisa, the progenitor of the Macintosh, which in turn inspired Microsoft Windows and IBM OS/2 came from a visit to PARC), the first distributed fileserver XDFS , the first print server, one of the first distributed services: Grapevine (a messaging and authentication system), object-oriented programming (Smalltalk) and Hoare's condition variables and monitors were invented as part of the Mesa programming language used in the Cedar system.

2.2.1 Workstations-Networks

The vision of the PARC research (and the commercial systems that followed) was to replace the timesharing mini-computer with single-user workstations. The genesis of this idea is that it made computers (i.e., workstations and PCs) into a commodity item that like a TV or a car could be produced efficiently and cheaply. The main costs of a computer are the engineering costs of designing it and designing the manufacturing process to build it. If you build more units, you can amortised the engineering costs better and thus make the computers cheaper. This is a very important idea and is the main reason that distribution is an excellent way to build a cheap scalable system.

2.2.2 Wide-Scale Distributed Computing and the Internet

Another type of distributed computing that is becoming increasingly important today is the sort of wide-scale distribution possible with the Internet.

At roughly the same time as the workstation and local-area network were being invented, the U.S. Department of Defence was putting tons of U.S. taxpayer money to work to set up a world-wide communication system that could be used to support distributed science and engineering research needed to keep the Defence Dept. supplied with toys. They were greatly concerned that research assets not be centrally located, as doing so would allow one well-placed bomb to put them out

of business. This is an example of another benefit of distributed systems: fault tolerance through replication.

2.3 DISTRIBUTED SYSTEMS

Multiprocessor systems have more than one processing unit sharing memory/ peripheral devices. They have greater computing power, and higher reliability. Multiprocessor systems are classified into two:

- **Tightly-coupled**: Each processor is assigned a specific duty but processors work in close association, possibly sharing one memory module.
- **Loosely-coupled (distributed)**: Each processor has its own memory and copy of the OS.

A distributed computer system is a loosely coupled collection of autonomous computers connected by a network using system software to produce a single integrated computing environment.

A distributed operating system differs from a network of machines each supporting a network operating system in only one way: The machines supporting a distributed operating system are all running under a single operating system that spans the network. Thus, the print spooler might, at some instant, be running on one machine, while the file system is running on others, while other machines are running other parts of the system, and under some distributed operating systems, these software parts may at times migrate from machine to machine.

With network operating systems, each machine runs an entire operating system. In contrast, with distributed operating systems, the entire system is itself distributed across the network. As a result, distributed operating systems typically make little distinction between remote execution of a command and local execution of that same command. In theory, all commands may be executed anywhere; it is up to the system to execute commands where it is convenient.

2.4 KEY FEATURES AND ADVANTAGES OF A DISTRIBUTED SYSTEM

The following are the key features of a distributed system:

- They are Loosely coupled
  - remote access is many times slower than local access
- Nodes are autonomous
  - workstation resources are managed locally
- Network connections using system software
  - remote access requires explicit message passing between nodes
  - messages are CPU to CPU
  - protocols for reliability, flow control, failure detection, etc., implemented in software

- the *only* way two nodes can communicate is by sending and receiving network messages–this differs from a hardware approach in which hardware signalling can be used for flow control or failure detection.

2.4.1 Advantages of Distributed Systems over Centralised Systems

- Better price/performance than mainframes
- More computing power
  - sum of the computing power of the processors in the distributed system may be greater than any single processor available (parallel processing)
- Some applications are inherently distributed
- Improved reliability because system can survive crash of one processor
- Incremental growth can be achieved by adding one processor at a time
- Shared ownership facilitated.

2.4.2 Advantages of Distributed Systems over Isolated PCs

- Shared utilisation of resources.
- Communication.
- Better performance and flexibility than isolated personal computers.
- Simpler maintenance if compared with individual PC's.

2.4.3 Disadvantages of Distributed Systems

Although we have seen several advantages of distributed systems, there are certain disadvantages also which are listed below:

- Network performance parameters.
- *Latency*: Delay that occurs after a send operation is executed before data starts to arrive at the destination computer.
- *Data Transfer Rate*: Speed at which data can be transferred between two computers once transmission has begun.
- *Total network bandwidth*: Total volume of traffic that can be transferred across the network in a give time.
- Dependency on reliability of the underlying network.
- Higher security risk due to more possible access points for intruders and possible communication with insecure systems.
- Software complexity.

2.5 DESIGN GOALS OF DISTRIBUTED SYSTEMS

In order to design a good distributed system. There are six key design goals. They are:

- Concurrency
- Scalability
- Openness
- Fault Tolerance
- Privacy and Authentication
- Transparency.

Let us discuss them one by one.

2.5.1 Concurrency

A server must handle many client requests at the same time. Distributed systems are naturally concurrent; that is, there are multiple workstations running programs independently and at the same time. Concurrency is important because any distributed service that isn't concurrent would become a bottleneck that would serialise the actions of its clients and thus reduce the natural concurrency of the system.

2.5.2 Scalability

The goal is to be able to use the same software for different size systems. A distributed software system is scalable if it can handle increased demand on any part of the system (i.e., more clients, bigger networks, faster networks, etc.) without a change to the software. In other words, we would like the engineering impact of increased demand to be proportional to that increase. Distributed systems, however, can be built for a very wide range of scales and it is thus not a good idea to try to build a system that can handle everything. A local-area network file server should be built differently from a Web server that must handle millions of requests a day from throughout the world. The key goal is to understand the target system's expected size and expected growth and to understand how the distributed system will scale as the system grows.

2.5.3 Openness

Two types of openness are important: non-proprietary and extensibility.

Public protocols are important because they make it possible for many software manufacturers to build clients and servers that will be able to talk to each other. Proprietary protocols limit the "players" to those from a single company and thus limit the success of the protocol.

A system is extensible if it permits customisations needed to meet unanticipated requirements. Extensibility is important because it aids scalability and allows a system to survive over time as the demands on it and the ways it is used change.

2.5.4 Fault Tolerance

It is critically important that a distributed system be able to tolerate "partial failures". Why is it so important? Two reasons are as follows:

- *Failures are more harmful:* Many clients are affected by the failure of a

distributed service, unlike a non-distributed system in which a failure affects only a single node.

- *Failures are more likely:* A distributed service depends on many components (workstation nodes, network interfaces, networks, switches, routers, etc.) all of which must work. Furthermore, a client will often depend on multiple distributed services (e.g., multiple file systems or databases) in order to function properly. The probability that such a client will experience a failure can be approximated as the sum of the individual failure probabilities of everything that it depends on. Thus, a client that depends on *N* components (hardware or software) that each have failure probability *P* will fail with probability roughly *N*P*. (This approximation is valid for small values of *P*. The exact failure probability is (1−(1−P)^N).)

There are two aspects of failure tolerance to be studied as shown below:

Recovery

- A failure shouldn't cause the loss (or corruption) of critical data, or computation.
- After a failure, the system should recover critical data, even data that was being modified when the failure occurred. Data that survives failures is called "persistent" data.
- Very long-running computations must also be made recoverable in order to restart them where they left off instead of from the beginning.
- For example, if a fileserver crashes, the data in the file system it serves should be intact after the server is restarted.

Availability

- A failure shouldn't interrupt the service provided by a critical server.
- This is a bit harder to achieve than recovery. We often speak of a highly-available service as one that is almost always available even if failures occur.
- The main technique for ensuring availability is service replication.
- For example, a fileserver could be made highly available by running two copies of the server on different nodes. If one of the servers fails, the other should be able to step in without service interruption.

2.5.5 Privacy and Authentication

Privacy is achieved when the sender of a message can control what other programs (or people) can read the message. The goal is to protect against eavesdropping. For example, if you use your credit card to buy something over the Web, you will probably want to prevent anyone but the target Web server from reading the message that contains your credit card account number. Authentication is the process of ensuring that programs can know who they are talking to. This is important for both clients and servers.

For clients authentication is needed to enable a concept called trust. For example, the fact that you are willing to give your credit card number to a merchant when

you buy something means that you are implicitly trusting that merchant to use your number according to the rules to which you have both agreed (to debit your account for the amount of the purchase and give the number to no one else). To make a Web purchase, you must trust the merchant's Web server just like you would trust the merchant for an in-person purchase. To establish this trust, however, you must ensure that your Web browser is really talking to the merchant's Web server and not to some other program that's just pretending to be their merchant.

For servers authentication is needed to enforce access control. For a server to control who has access to the resources it manages (your files if it is a fileserver, your money if it is a banking server), it must know who it is talking to. A Unix login is a crude example of an authentication used to provide access control. It is a crude example because a remote login sends your username and password in messages for which privacy is not guaranteed. It is thus possible, though usually difficult, for someone to eavesdrop on those messages and thus figure out your username and password.

For a distributed system, the only way to ensure privacy and authentication is by using cryptography.

2.5.6 Transparency

The final goal is transparency. We often use the term **single system image** to refer to this goal of making the distributed system look to programs like it is a tightly coupled (i.e., single) system.

This is really what a distributed system software is all about. We want the system software (operating system, runtime library, language, compiler) to deal with all of the complexities of distributed computing so that writing distributed applications is as easy as possible.

Achieving complete transparency is difficult. There are eight types, namely:

- ***Access Transparency*** enables local and remote resources to be accessed using identical operations
- ***Location Transparency*** enables resources to be accessed without knowledge of their (physical) location. Access transparency and location transparency are together referred to as network transparency.
- ***Concurrency Transparency*** enables several processes to operate concurrently using shared resources without interference between them.
- ***Replication Transparency*** enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers.
- ***Failure Transparency*** enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components.
- ***Mobility Transparency*** allows the movement of resources and clients within a system without affecting the operation of users or programs.

- ***Performance Transparency*** allows the system to be reconfigured to improve performance as loads change
- ***Scaling Transparency*** Transparency allows the system and applications to expand in scale without change to the system structure or the application algorithms

2.6 DESIGN ISSUES INVOLVED IN DISTRIBUTED SYSTEMS

There are certain design issues to be considered for distributed systems. They are:

a) Naming

b) Communication

c) Software Structure

d) Workload Allocation

e) Consistency Maintenance

Let us discuss the issues briefly:

2.6.1 Naming

- A name is a string of characters used to identify and locate a distributed resource.
- An identifier is a special kind of name that is used directly by the computer to access the resource.

  For example the identifier for a Unix server would include at least (1) an IP address and (2) a port number. The IP address is used to find the node that runs the server and the port number identifies the server process on that node.
- Resolution is the process of turning a name into an identifier. Resolution is performed by a Name Server. It is also called "binding" (as in binding a name to a distributed service).

For example, an IP domain name (e.g., cs.ubc.ca) is turned into a IP address by the IP Domain Name Server (DNS), a distributed hierarchical server running in the Internet.

2.6.2 Communication

- Getting different processes to talk to each other
- Messages
- Remote method invocation.

2.6.3 Software Structure

- The main issues are to choose a software structure that supports our goals, particularly the goal of openness.

- We thus want structures that promote extensibility and otherwise make it easy to program the system.
- Alternatives are:
    - A monolithic structure is basically a big pile of code; it is not so desirable because it is hard to extend or reason about a system like that.
    - A modular structure divides the system into models with well-defined interfaces that define how the models interact. Modular systems are more extensible and easier to reason about than monolithic systems.
    - A layered structure is a special type of modular structure in which modules are organised into layers (one on top of the other). The interaction between layers is restricted such that a layer can communicate directly with only the layer immediately above and the layer immediately below it. In this way, each layer defines an abstraction that is used by the layer immediately above it. Clients interact only with the top layer and only the bottom layer deals with the hardware (e.g., network, disk, etc.) Network protocols have traditionally been organised as layers (as we will see in the next class) and for this reason we often refer to these protocols as "protocol stacks".
    - Operating systems can be either monolithic (e.g., UNIX and Windows) or modular (e.g., Mach and Windows NT). A modular operating system is called a micro kernel. Amicro-kernel OS has a small minimalist kernel that includes as little functionality as possible. OS services such as VM, file systems, and networking are added to the system as separate servers and they reside in their own user-mode address spaces outside the kernel.

Let us study more details regarding the software structure in section 2.7 of this unit.

2.6.4 Workload Allocation

- The key issue is load balancing: The allocation of the network workload such that network resources (e.g., CPUs, memory, and disks) are used efficiently.

For CPUs, there are two key approaches. They are*:*

Processor Pools

Put all of the workstations in a closet and give the users cheap resource-poor X terminals. User processes are allocated to the workstations by a distributed operating system that controls all of the workstations. Advantage is that process scheduling is simplified and resource utilisation more efficient because workstations aren't "owned" by anyone. The OS is free to makes scheduling decisions based solely on the goal of getting the most work done. Disadvantage is that, these days, powerful workstations aren't much more expensive than cheap X terminals. So if each user has a powerful workstation instead of a cheap X terminal, then we don't need the machines in the closet and we thus have the idle-workstation model described below:

Idle Workstations

Every user has a workstation on their desk for doing their work. These workstations are powerful and are thus valuable resources. But, really people don't use their workstations all of the time. There's napping, lunches, reading, meetings, etc. - lots of times when workstations are **idle**. In addition, when people are using their workstation, they often don't need all of its power (e.g., reading mail doesn't require a 200-Mhz CPU and 64-Mbytes of RAM). The goal then is to move processes from active workstations to idle workstations to balance the load and thus make better use of network resources. But, people "own" (or at least feel like they own) their workstations. So a key issue for using idle workstations is to avoid impacting workstation users (i.e., we can't slow them down). So what happens when I come back from lunch and find your programs running on my machine? I'll want your processes moved elsewhere NOW. The ability to move an active process from one machine to another is called **process migration**. Process migration is necessary to deal with the user-returning-from-lunch issue and is useful for rebalancing network load as the load characterises a network change over time.

2.6.5 Consistency Maintenance

The final issue is consistency. There are four key aspects of consistency: atomicity, coherence, failure consistency, and clock consistency.

Atomicity

- The goal is to provide the "all-or-nothing" property for sequences of operations.
- Atomicity is important and difficult for distributed systems because operations can be messages to one or more servers. To guarantee atomicity thus requires the coordination of several network nodes.

Coherence

Coherence is the problem of maintaining the consistency of replicated data.

We have said that replication is a useful technique for (1) increasing availability and (2) improving performance.

When there are multiple copies of an object and one of them is modified, we must ensure that the other copies are updated (or invalidated) so that no one can erroneously read an out-of-date version of the object.

A key issue for providing coherence is to deal with the event-ordering problem.

Failure Consistency

- We talked last time about the importance of the goal of failure tolerance. To build systems that can handle failures, we need a model that clearly defines what a failure is. There are two key types of failures: *Fail-stop and Byzantine*.
- ***Fail-stop*** is a simplified failure model in which we assume that the only way a component will fail is by stopping. In particular, a will never fail by giving a wrong answer.

- ***Byzantine*** failure is a more encompassing model that permits failures in which a failed node might not stop but might just start giving the wrong answers. Techniques for dealing with Byzantine failure typically involve performing a computation redundantly (and often in different ways with different software). The system then compares all of the answers generated for a given computation and thus detects when one component starts giving the wrong answer. The ability to detect Byzantine failures is important for many safety-critical systems such as aircraft avionics.
- In this class we will assume failures are Fail-Stop; this is a common assumption for distributed systems and greatly simplifies failure tolerance.

Clock Consistency

- Maintaining a consistent view of time needed to order network events is critical and challenging. This becomes further difficult when the nodes of the distributed system are geographically apart. Let us study the scheme provided by Lamport on Ordering of events in a distributed environment in the next section.

2.6.6 Lamport's Scheme of Ordering of Events

Lamport proposed a scheme to provide ordering of events in a distributed environment using logical clocks. Because it is impossible to have perfectly synchronized clocks and global time in a distributed system, it is often necessary to use logical clocks instead.

Definitions:

Happened Before Relation (->): This relation captures causal dependencies between events, that is, whether or not events have a cause and effect relation. This relation (->) is defined as follows:

- a -> b, if *a* and *b* are in the same process and *a* Occurred before *b*.
- a -> b, if *a* is the event of sending a message and *b* is the receipt of that message by another process.

If a -> b and b -> c, then a -> c, that is, the relation has the property of transitivity.

Causally Related Events: If *event a -> event b*, then *a* casually affects *b*.

Concurrent Events: Two distinct events *a* and *b* are concurrent (*a* || *b*) if (not) *a* -> *b* and (not) b -> a. That is, the events have no causal relationship. This is equivalent to b || a.

For any two events *a and b* in a system, only one of the following is true: a -> b, b -> a, or a || b.

Lamport introduced a system of logical clocks in order to make the -> relation possible. It works like this: Each process P_i in the system has its own clock C_i. C_i can be looked at as a function that assigns *a* number, $C_i(a)$ to an event *a*. This is the timestamp of the event *a* in process P_i. These numbers are not in any way related to physical time — that is why they are called logical clocks. These are generally implemented using counters, which increase each time an event occurs. Generally, an event's timestamp is the value of the clock at the time it occurs.

Conditions Satisfied by the Logical Clock system:

For any events *a* and *b*, if a -> b, then C(a) < C(b). This is true if two conditions are met:

- If a occurs before b, then $C_i(a) < C_i(b)$.
- If a is a message sent from P_i and b is the receipt of that same message in P_j, then $C_i(a) < C_j(b)$.

Implementation Rules Required:

Clock C_i is incremented for each event: $C_i := C_i + d \quad (d > 0)$
if *a* is the event of sending a message from one process to another, then the receiver sets its clock to the max of its current clock and the sender's clock - that is, $C_j := \max(C_j, t_m + d) \quad (d > 0)$.

2.7 DISTRIBUTED SYSTEM STRUCTURE

There are three main alternative ways to structure a distributed application. Each of them defines a different **application model** for writing distributed programs.

The three distributed system application models are:

- Client / Server (and RPC)
- Distributed objects
- Distributed shared memory.

Each model requires a different set of "system" software to support applications written in that style. This system software consists of a combination of operating system and runtime-library code and support from languages and compilers. What all of this code does is translate the distributed features of the particular application model into the sending and receiving of network messages. Because, remember that the only way that nodes (workstations/PCs) can communicate in a distributed system is by sending and receiving messages.

2.7.1 Application Model 1: Client Server

- A designated node exports a "service"; this node is called the "server".
- Nodes that import and use the service are called "clients".
- Clients communicate with servers using a "request-response" protocol.

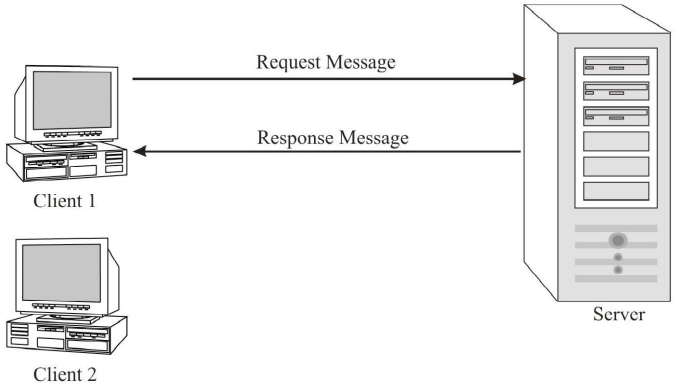

Figure 1: Client Server Model

- Request-response differs from other protocols.
    - Peer-to-peer protocols don't have a designated server.
    - Streaming protocols such as TCP/IP work differently as we will see a couple of classes from now.
- We should really use the terms client and server to refer to the pieces of the software that implement the server and not to the nodes themselves. Why? Because a node can actually be both a client and server. Consider NFS for example. One workstation might export a local filesystem to other nodes (i.e., it is a server for the file system) and import other filesystems exported by remote nodes (i.e., it is a client for these filesystems).
- Application interface can be either send/receive or RPC
    - **Send/receive :** The Unix socket interface is an example. Clients communicate with server by building a message (a sequence of characters) that it then explictly sends to the server. Clients receive messages by issuing a "receive" call.
      We'll talk about this on Friday.
    - **RPC :** A remote procedure call interface allows a client to request a server operation by making what looks like a procedure call. The system software translates this RPC into a message send to the server and a message receives to wait for the reply. The advantages of RPC over send/ receive is that the system hides some of the details of message passing from programs: it's easier to call a procedure than format a message and then explicitly send it and wait for the reply. (Study more on RPC given in section 2.9 of this unit).

2.7.2 Application Model 2: Distributed Objects

- Similar to RPC-based client-server.
- Language-level objects (e.g., C++, Java, Smalltalk) are used to encapsulate data and functions of a distributed service.
- Objects can reside in either the memory of a client or a server.
- Clients communicate with servers through objects that they "share" with the server. These shared objects are located in a client's memory and look to the client just like other "local" objects. We call these objects "remote" objects because they represent a remote service (i.e., something at the server). When a client invokes a method of a remote object, the system might do something locally or it might turn the method invocation into an RPC to the server. (Recall that in the object-oriented world, procedures are called methods and procedure callls are called method invocations).
- There are two main advantages of Distributed Objects over RPC: (1) objects hide even more of the details of distribution than RPC (more next week) and (2) objects allow clients and servers to communicate using in two different ways: function shipping (like RPC) and data shipping.

- ***function shipping*** means that a client calls the server and asks it to perform a function; this is basically like RPC.
- ***data shipping*** means that the server sends some data to the client (stores it as part of an object) and the client then performs subsequent functions locally instead of having to call the server every time it wants to do something.

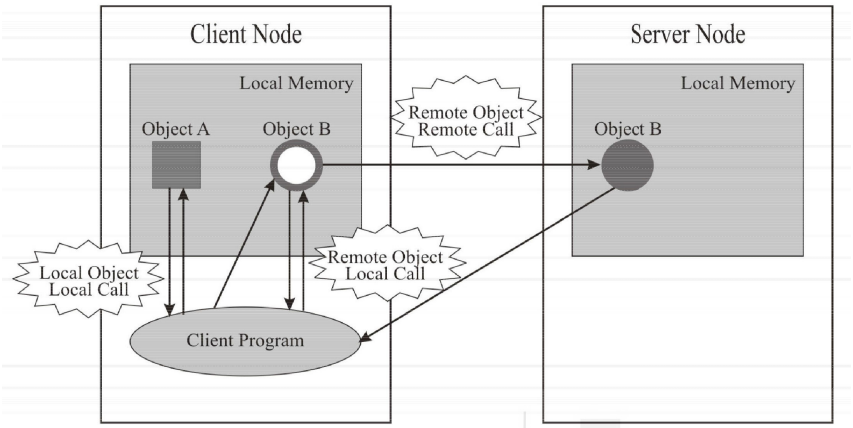

Figure 2: Distributed objects models

- For example, in the picture above, a client program running can access local object A and remote object B (controlled by the server node) in exactly the same way; the arrows show flow of a procedure call into the object and a return from it. The server can design B so that calls are turned into remote calls just like RPC (red) or it can ship some data to the client with the client's copy of the object and thus allow some calls to run locally on the client (orange).
- More on distributed objects later in the course.

2.7.3 Application Model 3: Distributed Shared Memory

- Usually used for peer-to-peer communication instead of client-server.
- Clients communicate with each other through shared variables.

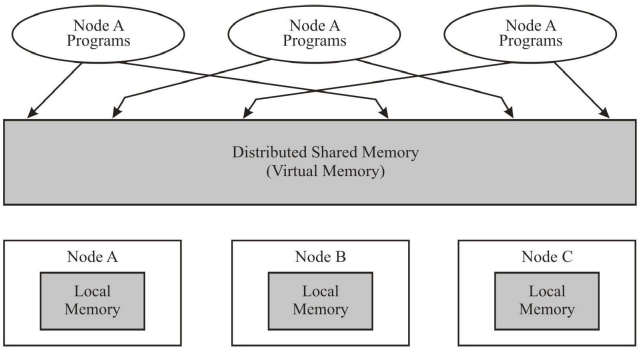

Figure 3: Distributed shared memory model

- The system implements the illusion of shared memory and translates accesses to shared data into the appropriate messages.
- To a program, the distributed system looks just like a shared-memory multiprocessor.

- The advantage is that it is really easy to program: hides all of the details of distribution.
- The disadvantage is that it often hides too much. As far as a program knows, everything in its memory is local. But really, some parts of its memory are stored on or shared with a remote node. Access to this remote data is very SLOW compared with accessing local data. For good performance, a program usually needs to know what data is local and what data is remote.
- Another disadvantage is that it is very complicated to implement a distributed-shared memory system that works correctly and performs well.
- We'll talk more about distributed shared memory towards the end of the term.

2.8 MUTUAL EXCLUSION IN DISTRIBUTED SYSTEMS

When all processes sharing a resource are on the same machine, mutual exclusion is easily assured by the use of a semaphore, spin-lock or other similar shared abstraction. When the processes involved are on different machines, however, mutual exclusion becomes more difficult.

Consider the following example: A number of machines in a network are competing for access to a printer. The printer is so constructed that every line of text sent to the printer must be sent in a separate message, and thus if a process wants to print an entire file, it must obtain exclusive use of the printer, somehow, send all the lines of the file it wishes to print, and then release the printer for use by others on the network.

A trivial solution to this problem is to install a print spooler process somewhere on the net. That print spooler would gather lines of files provided by various applications processes, maintain a queue of completed files that are ready to print, and print one such file at a time. This works, but it introduces some problems: First, the spooler must have buffer capacity to hold the aggregate of all the files that have not yet been printed. Second, the spooler may become a bottleneck, limiting the performance of the entire system, and third, if the processor supporting the spooler is unreliable, the spooler may limit the reliability of the system.

2.8.1 Mutual Exclusion Servers

In the printer example being used here, the problem of storage space in the spooler typically becomes acute with graphics printers. In such a context, it is desirable to block an applications process until the printer is ready to accept data from that applications process, and then let that process directly deliver data to the printer.

For example, an applications process may be coded as follows:

Send request for permission to the spooler
Await reply giving permission to print

Loop

```
    send data directly to printer
  End Loop

  Send notice to spooler that printing is done
```

If all users of the spooler use this protocol, the spooler is no longer serving as a spooler, it is merely serving as a mutual exclusion mechanism! In fact, it implements exactly the same semantics as a binary semaphore, but it implements it using a client server model.

The process implementing a semaphore using message passing might have the following basic structure:

```
Loop
  Await message from client
  Case message type of
   P:
     If count > 0
     Send immediate reply
     count = count - 1
     Else
       Enqueue identity of client
     End if
   V:
     If queue is empty,
     count = count + 1
     Else
       Dequeue one blocked client
       Send a delayed reply
     End if
  End case
End Loop
```

This requires a count and a queue of return addresses for each semaphore. Note that, by presenting this, we have proven that blocking message passing can be used to implement semaphores. Since we already know that semaphores plus shared memory are sufficient to implement blocking message passing, we have proven the equivalence, from a computation theory viewpoint, of these two models of interprocess communication.

The disadvantage of implementing semaphores using a server process is that server becomes a potential source of reliability problems. If we can build a mutual exclusion algorithm that avoids use of a dedicated server, for example, by having the processes that are competing for entry to a critical section negotiate directly with each other, we can potentially eliminate the reliability problem.

2.8.2 Token Based Mutual Exclusion

One alternative to the mutual-exclusion server given above is to arrange the competing processes in a ring and let them exchange a token. If a process receives

the token and does not need exclusive use of the resource, it must pass the token on to the next process in the ring. If a process needs exclusive use of the resource, it waits for the token and then holds it until it is done with the resource, at which point it puts the token back in circulation.

This is the exact software analog of a token ring network. In a token ring network, only one process at a time may transmit, and the circulating token is used to assure this, exactly as described. The token ring network protocol was developed for the hardware level or the link-level of the protocol hierarchy. Here, we are proposing building a virtual ring at or above the transport layer and using essentially the same token passing protocol.

This solution is not problem free. What if the token is lost? What if a process in the ring ceases transmission? Nonetheless, it is at the root of a number of interesting and useful distributed mutual exclusion algorithms. The advantage of such distributed algorithms is that they do not rest on a central authority, and thus, they are ideal candidates for use in fault tolerant applications.

An important detail in the token-based mutual exclusion algorithm is that, on receiving a token, a process *must* immediately forward the token if it is not waiting for entry into the critical section. This may be done in a number of ways:

- Each process could periodically check to see if the token has arrived. This requires some kind of non-blocking read service to allow the process to poll the incoming network connection on the token ring. The UNIX FNDELAY flag allows non-blocking read, and the Unix select() kernel call allows testing an I/O descriptor to see if a read from that descriptor would block; either of these is sufficient to support this polling implementation of the token passing protocol. The fact that UNIX offers two such mechanisms is good evidence that these are afterthoughts added to Unix after the original implementation was complete.

- The receipt of an incoming token could cause an interrupt. Under UNIX, for example, the SIGIO signal can be attached to a socket or communications line (see the FASYNC flag set by fcntl). To await the token, the process could disable the SIGIO signal and do a blocking read on the incoming token socket. To exit the critical section, the process could first enable SIGIO and then send the token. The SIGIO handler would read the incoming token and forward it before returning.

- A thread or process could be dedicated to the job of token management. We'll refer to such a thread or process as the mutual exclusion agent of the application process. Typically, the application would communicate with its agent using shared memory, semaphores, and other uni-processor tools, while the agent speaks to other agents over the net. When the user wants entry to the critical section, it sets a variable to "let me in" and then does a wait on the entry semaphore it shares with the agent. When the user is done, the user sets the shared variable to "done" and then signals the go-on semaphore it shares with the agent. The agent always checks the shared variable when it receives the token, and only forwards it when the variable is equal to "done".

2.8.3 Lamport's Bakery Algorithm

One decentralised algorithm in common use, for example, in bakeries, is to issue numbers to each customer. When the customers want to access the scarce resource (the clerk behind the counter), they compare the numbers on their slips and the user with the lowest numbered slip wins.

The problem with this is that there must be some way to distribute numbers, but this has been solved. In bakeries, we use a very small server to distribute numbers, in the form of a roll of tickets where conflicts between two customers are solved by the fact that human hands naturally exclude each other from the critical volume of space that must be occupied to take a ticket. We cannot use this approach for solving the problem on a computer.

Before going on to more interesting implementations for distributing numbers, note that clients of such a protocol may make extensive use of their numbers! For example, if the bakery contains multiple clerks, the clients could use their number to select a clerk (number modulo number of clerks). Similarly, in a FIFO queue implemented with a bounded buffer, the number modulo the queue size could indicate the slot in the buffer to be used, allowing multiple processes to simultaneously place values in the queue.

Lamport's Bakery Algorithm provides a decentralised implementation of the "take a number" idea. As originally formulated, this requires that each competing process share access to an array, but later distributed algorithms have eliminated this shared data structure. Here is the original formulation:

For each process, *i*, there are two values, C[i] and N[i], giving the status of process I and the number it has picked. In more detail:

N[i] = 0 —> Process i is not in the bakery.
N[i] > 0 —> Process i has picked a number and is in the bakery.
C[i] = 0 —> Process i is not trying to pick a number.
C[i] = 1 —> Process i is trying to pick a number.

when

N[i] = min(for all j, N[j] where N[j] > 0)
Process i is allowed into the critical section.

Here is the basic algorithm used to pick a number:

```
C[i] := 1;
N[i] := max( for all j, N[j] ) + 1;
C[i] := 0;
```

In effect, the customer walks into the bakery, checks the numbers of all the waiting customers, and then picks a number one larger than the number of any waiting customer.

If two customers each walk in at the same time, they are each likely to pick the same number. Lamport's solution allows this but then makes sure that customers notice that this has happened and break the tie in a sensible way.

To help the customers detect ties, each customer who is currently in the process of picking a number holds his hand up (by setting C[i] to 1. s/he pulls down his hand when s/he is done selecting a number — note that selecting a number may take time, since it involves inspecting the numbers of everyone else in the waiting room.

A process does the following to wait for the baker:

```
Step 1:
while (for some j, C(j) = 1) do nothing;
```

First, wait until any process which might have tied with you has finished selecting their numbers. Since we require customers to raise their hands while they pick numbers, each customer waits until all hands are down after picking a number in order to guarantee that all ties will be cleanly recognised in the next step.

```
Step 2:

 repeat
   W := (the set of j such that N[j] > 0)
    (where W is the set of indeces of waiting processes)
   M := (the set of j in W
         such that N[j] <= N[k]
         for all k in W)
   (where M is the set of process indices with minimum numbers)
   j := min(M)
    (where  is in M and the tie is broken)
 until i = j;
```

Second, wait until your ticket number is the minimum of all tickets in the room. There may be others with this minimum number, but in inspecting all the tickets in the room, you found them! If you find a tie, see if your customer ID number is less than the ID numbers of those with whom you've tied, and only then enter the critical section and meet with the baker.

This is inefficient, because you might wait a bit too long while some other process picks a number after the number you picked, but for now, we'll accept this cost.

If you are not the person holding the smallest number, you start checking again. If you hold the smallest number, it is also possible that someone else holds the smallest number. Therefore, what you've got to do is agree with everyone else on how to break ties.

The solution shown above is simple. Instead of computing the value of the smallest number, compute the minimum process ID among the processes that hold the smallest value. In fact, we need not seek the minimum process ID, all we need to do is use any deterministic algorithm that all participants can agree on for breaking the tie. As long as all participants apply the same deterministic algorithms to the same information, they will arrive at the same conclusion.

To return its ticket, and exit the critical section, processes execute the following trivial bit of code:

```
N[i] := 0;
```

When you return your ticket, if any other processes are waiting, then on their next scan of the set of processes, one of them will find that it is holding the winning ticket.

Moving to a Distributed Context

In the context of distributed systems, Lamport's bakery algorithm has the useful property that process *i* only modifies its own *N[i]* and *C[i]*, while it must read the entries for all others. In effect, therefore, we can implement this in a context where each process has read-only access to the data of all other processes, and read-write access only to its own data.

A distributed implementation of this algorithm can be produced directly by storing *N[i]* and *C[i]* locally with process *i*, and using message passing when any process wants to examine the values of *N* and *C* for any process other than itself. In this case, each process must be prepared to act as a server for messages from the others requesting the values of its variables; we have the same options for implementing this service as we had for the token passing approach to mutual exclusion. The service could be offered by an agent process, by an interrupt service routine, or by periodic polling of the appropriate incoming message queues.

Note that we can easily make this into a fault tolerant model by using a fault-tolerant client-server protocol for the requests. If there is no reply to a request for the values of process i after some interval and a few retries, we can simply assume that process *i* has failed.

This demonstrates that fault tolerant mutual exclusion can be done without any central authority! This direct port of Lamport's bakery algorithm is not particularly efficient, though. Each process must read the variables of all other processes a minimum of 3 times−once to select a ticket number, once to see if anyone else is in the process of selecting a number, and once to see if it holds the minimum ticket.

For each process contending for entry to the critical section, there are about *6N* messages exchanged, which is clearly not very good. Much better algorithms have been devised, but even this algorithm can be improved by taking advantage of knowledge of the network structure. On an Ethernet or on a tree-structured network, a broadcast can be done in parallel, sending one message to *N* recipients in only a few time units. On a tree-structured network, the reply messages can be merged on the way to the root (the originator of the request) so that sorting and searching for the maximum N or the minimum nonzero N can be distributed efficiently.

2.8.4 Ricart and Agrawala's Mutual Exclusion Algorithm

Another alternative is for anyone wishing to enter a critical section to broadcast their request; as each process agrees that it is OK to enter the section, they reply to the broadcaster saying that it is OK to continue; the broadcaster only continues when all replies are in.

If a process is in a critical section when it receives a request for entry, it defers its reply until it has exited the critical section, and only then does it reply. If a process is not in the critical section, it replies immediately.

This sounds like a remarkably naive algorithm, but with point-to-point communications between N processes, it takes only 2(N−1) messages for a process to enter the critical section, N−1 messages to broadcast the request and N−1 replies.

There are some subtle issues that make the result far from naive. For example, what happens if two processes each ask at the same time? What should be done with requests received while a process is waiting to enter the critical section?

Ricart and Agrawala's mutual exclusion algorithm solves these problems. In this solution, each process has 3 significant states, and its behaviour in response to messages from others depends on its state:

Outside the critical section

The process replies immediately to every entry request.

After requesting entry, awaiting permission to enter.

The process replies immediately to higher priority requests and defers all other replies until exit from the critical section.

Inside critical section.

The process defers all replies until exit from the critical section.

As with Lamport's bakery algorithm, this algorithm has no central authority. Nonetheless, the interactions between a process requesting entry to a critical section and each other process have a character similar to client-server interactions. That is, the interactions take the form of a request followed (possibly some time later) by a reply.

As such, this algorithm can be made fault tolerant by applying the same kinds of tricks as are applied in other client server applications. On receiving a request, a processor can be required to immediately send out either a reply or a negative acknowledgement. The latter says "I got your request and I can't reply yet!"

With such a requirement, the requesting process can wait for either a reply or a negative acknowledgement from every other process. If it gets neither, it can retry the request to that process. If it retries some limited number of times and still gets no answer, it can assume that the distant process has failed and give up on it.

If a process receives two consecutive requests from the same process because acknowledgements have been lost, it must resend the acknowledgement. If a process waits a long time and doesn't get an acknowledgement, it can send out a message saying "are you still there", to which the distant process would reply "I got your request but I can't reply yet". If it gets no reply, it can retry some number of times and then give up on the server as being gone.

If a process dies in its critical section, the above code solves the problem and lets one of the surviving processes in. If a process dies outside its critical section, this code also works.

Breaking Ties in Ricart and Agrawala's algorithm

There are many ways to break ties between processes that make simultaneous requests; all of these are based on including the priority of each requesting process

in the request message. It is worth noting that the same alternatives apply to Lamport's bakery algorithm!

A unique process ID can be used as the priority, as was done in Lamport's bakery algorithm. This is a static priority assignment and is almost always needed to break ties in any of the more complex cases. Typically, a process will append its statically assigned process ID to any more interesting information it uses for tiebreaking, thus guaranteeing that if two processes happen to generate the same interesting information, the tie will still be broken.

The number of times the process has previously entered the same critical section can be used; if processes that have entered the critical section more frequently are given lower priority, then the system will be fair, giving the highest priority to the least frequent user of the resource.

The time since last access to the critical section offers a similar opportunity to enforce fairness if the process that used the critical section least recently is given the highest priority.

If dynamic priority assignments are used, what matters is that the priority used on any entry to the critical section is frozen prior to broadcasting the request for entry, and that it remains the same until after the process is done with that round of mutual exclusion. It is also important that each process has a unique priority, but this can be assured by appending the process ID as the least significant bits of the dynamically chosen priority.

2.9 REMOTE PROCEDURE CALLS

Remote Procedure Call (RPC) is a powerful technique for constructing distributed, client-server based applications. It is based on extending the notion of conventional or local procedure calling, so that the called procedure need not exist in the same address space as the calling procedure. The two processes may be on the same system, or they may be on different systems with a network connecting them. By using RPC, programmers of distributed applications avoid the details of the interface with the network. The transport independence of RPC isolates the application from the physical and logical elements of the data communications mechanism and allows the application to use a variety of transports.

RPC is a client/server infrastructure that increases the interoperability, portability and flexibility of an application by allowing the application to be distributed over multiple heterogeneous platforms. It reduces the complexity of developing applications that span multiple operating systems and network protocols by insulating the application developer from the details of the various operating system and network interfaces−function calls are the programmer's interface when using RPC. RPC makes the client/server model of computing more powerful and easier to program.

2.9.1 How RPC Works?

An RPC is analogous to a function call. Like a function call, when an RPC is made, the calling arguments are passed to the remote procedure and the caller waits for a response to be returned from the remote procedure. The *Figure 4*

shows the flow of activity that takes place during an RPC call between two networked systems. The client makes a procedure call that sends a request to the server and waits. The thread is blocked from processing until either a reply is received, or it times out. When the request arrives, the server calls a dispatch routine that performs the requested service, and sends the reply to the client. After the RPC call is completed, the client program continues. RPC specifically supports network applications.

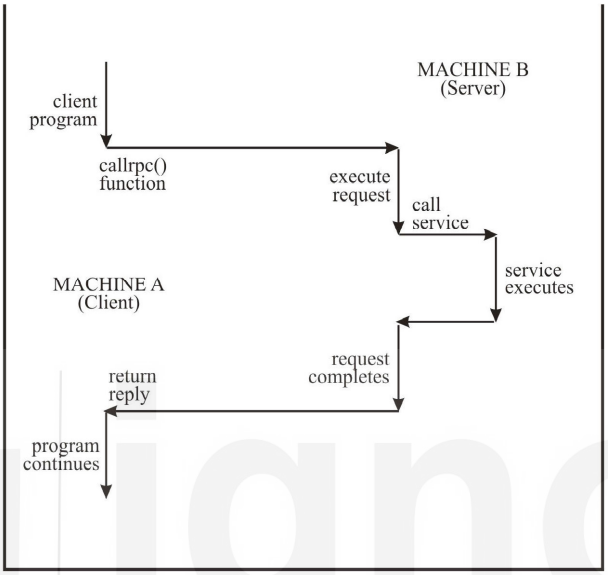

Figure 4: Flow of activity that Tables place during a RPC call between two networked system

2.9.2 Remote Procedure Calling Mechanism

A remote procedure is uniquely identified by the triple: (program number, version number, procedure number). The program number identifies a group of related remote procedures, each of which has a unique procedure number. A program may consist of one or more versions. Each version consists of a collection of procedures which are available to be called remotely. Version numbers enable multiple versions of a RPC protocol to be available simultaneously. Each version contains a number of procedures that can be called remotely. Each procedure has a procedure number.

2.9.3 Implementation of RPC

RPC is typically implemented in one of two ways:

1) Within a broader, more encompassing propriety product.

2) By a programmer using a proprietary tool to create client/server RPC stubs.

2.9.4 Considerations for Usage

RPC is appropriate for client/server applications in which the client can issue a request and wait for the server's response before continuing its own processing. Because most RPC implementations do not support peer-to-peer, or asynchronous, client/server interaction, RPC is not well-suited for applications involving distributed objects or object-oriented programming.

Asynchronous and synchronous mechanisms each have strengths and weaknesses that should be considered when designing any specific application. In contrast to asynchronous mechanisms employed by Message-Oriented Middleware, the use of a synchronous request-reply mechanism in RPC requires that the client and server are always available and functioning (i.e., the client or server is not blocked). In order to allow a client/server application to recover from a blocked condition, an implementation of a RPC is required to provide mechanisms such as error messages, request timers, retransmissions, or redirection to an alternate server. The complexity of the application using a RPC is dependent on the sophistication of the specific RPC implementation (i.e., the more sophisticated the recovery mechanisms supported by RPC, the less complex the application utilising the RPC is required to be). RPC's that implement asynchronous mechanisms are very few and are difficult (complex) to implement.

When utilising RPC over a distributed network, the performance (or load) of the network should be considered. One of the strengths of RPC is that the synchronous, blocking mechanism of RPC guards against overloading a network, unlike the asynchronous mechanism of Message Oriented Middleware (MOM). However, when recovery mechanisms, such as retransmissions, are employed by an RPC application, the resulting load on a network may increase, making the application inappropriate for a congested network. Also, because RPC uses static routing tables established at compile-time, the ability to perform load balancing across a network is difficult and should be considered when designing a RPC-based application.

Tools are available for a programmer to use in developing RPC applications over a wide variety of platforms, including Windows (3.1, NT, 95), Macintosh, 26 variants of UNIX, OS/2, NetWare, and VMS. RPC infrastructures are implemented within the Distributed Computing Environment (DCE), and within Open Network Computing (ONC), developed by Sunsoft, Inc.

2.9.5 Limitations

RPC implementations are nominally incompatible with other RPC implementations, although some are compatible. Using a single implementation of a RPC in a system will most likely result in a dependence on the RPC vendor for maintenance support and future enhancements. This could have a highly negative impact on a system's flexibility, maintainability, portability, and interoperability.

Because there is no single standard for implementing a RPC, different features may be offered by individual RPC implementations. Features that may affect the design and cost of a RPC-based application include the following:

- support of synchronous and/or asynchronous processing
- support of different networking protocols
- support for different file systems
- whether the RPC mechanism can be obtained individually, or only bundled with a server operating system.

Because of the complexity of the synchronous mechanism of RPC and the

proprietary and unique nature of RPC implementations, training is essential even for the experienced programmer.

2.10 OTHER MIDDLEWARE TECHNOLOGIES

Other middleware technologies that allow the distribution of processing across multiple processors and platforms are:

- Object Request Broker(ORB)
- Distributed Computing Environment (DCE)
- Message-Oriented Middleware (MOM)
- COM/DCOM
- Transaction Processing Monitor Technology
- 3-Tier S/W Architecture.

☞ **Check Your Progress 1**

1) How is a distributed OS different from Network OS? Explain.

.....

.....

.....

2) What is a Distributed File System?

.....

.....

.....

3) Define Load Balancing.

.....

.....

.....

4) What is the significance of Time Consistency?

.....

.....

.....

5) What is a Remote Procedure Call and mention its use.

.....

.....

.....

2.11 SUMMARY

A distributed operating system takes the abstraction to a higher level, and allows hides from the application where things are. The application can use things on any of many computers just as if it were one big computer. A distributed operating system will also provide for some sort of security across these multiple computers, as well as control the network communication paths between them. A distributed operating system can be created by merging these functions into the traditional operating system, or as another abstraction layer on top of the traditional operating system and network operating system.

Any operating system, including distributed operating systems, provides a number of services. First, they control what application gets to use the CPU and handle switching control between multiple applications. They also manage use of RAM and disk storage. Controlling who has access to which resources of the computer (or computers) is another issue that the operating system handles. In the case of distributed systems, all of these items need to be coordinated for multiple machines. As systems grow larger handling them can be complicated by the fact that not one person controls all of the machines so the security policies on one machine may not be the same as on another.

Some problems can be broken down into very tiny pieces of work that can be done in parallel. Other problems are such that you need the results of step one to do step two and the results of step two to do step three and so on. These problems cannot be broken down into as small of work units. Those things that can be broken down into very small chunks of work are called fine-grained and those that require larger chunks are called coarse-grain. When distributing the work to be done on many CPUs there is a balancing act to be followed. You don't want the chunk of work to be done to be so small that it takes too long to send the work to another CPU because then it is quicker to just have a single CPU do the work, You also don't want the chunk of work to be done to be too big of a chunk because then you can't spread it out over enough machines to make the thing run quickly.

In this unit we have studied the features of the distributed operating system, architecture, algorithms relating to the distributed processing, shared memory concept and remote procedure calls.

2.12 SOLUTIONS / ANSWERS

Check Your Progress 1

1) A distributed operating system differs from a network of machines each supporting a network operating system in only one way: The machines supporting a distributed operating system are all running under a single operating system that spans the network. Thus, the print spooler might, at some instant, be running on one machine, while the file system is running on others, while other machines are running other parts of the system, and under some distributed operating systems, these parts may at times migrate from machine to machine.

With network operating systems, each machine runs an entire operating system. In contrast, with distributed operating systems, the entire system is itself distributed across the network. As a result, distributed operating systems typically make little distinction between remote execution of a command and local execution of that same command. In theory, all commands may be executed anywhere; it is up to the system to execute commands where it is convenient.

2) Distributed File System (DFS) allows administrators to group shared folders located on different servers and present them to users as a virtual tree of folders known as a namespace. A namespace provides numerous benefits, including increased availability of data, load sharing, and simplified data migration.

3) Given a group of identical machines, it is wasteful to have some machines overloaded while others are almost idle. If each machine broadcasts its load average every few minutes, one can arrange for new processes to use whichever machine was least loaded at the time. However, one machine must not be given too many processes at once. If the system supports migration, and the load difference between two machines is great enough, a process should be migrated from one to the other.

4) Machines on a local area network have their own clocks. If these are not synchronized, strange things can happen: e.g., the modification date of a file can be in the future. All machines on a LAN should synchronize their clocks periodically, setting their own time to the "network time". However, adjusting time by sudden jumps also causes problems: it may lead to time going backward on some machines. A better way is to adjust the speed of the clock temporarily. This is done by protocols such as NTP.

5) Many distributed systems use Remote Procedure Calls (RPCs) as their main communication mechanism. It is a powerful technique for constructing distributed, client server based applications. It is based on extending the notion of conventional or local procedure calling, so that the called procedure need not exist in the same address space as the calling procedure. The two processes may be on the same system, or they may be on different systems with a network connecting them. By using RPC, programmers of distributed applications avoid the details of the interface with the network. The transport independence of RPC isolates the application from the physical and the logical elements of the data communications mechanism and allows the application to use a variety of transports.

2.13 FURTHER READINGS

1) Abraham Silberschatz, Peter Baer Galvin and Greg Gagne, *Applied Operating System Concepts*, 1/e, John Wiley & Sons, Inc, New Delhi.

2) Maarten van Steen and Henk Sips, *Computer and Network Organization: An Introduction*, Prentice Hall, New Delhi.

3) Andrew S. Tanenbaum and Albert S.Woodhull, *Operating Systems Design and Implementation*, 2/e, Prentice Hall, New Delhi.

4) Andrew S. Tanenbaum, *Modern Operating Systems*, 2/e, Prentice Hall, New Delhi.

5) Maarten van Steen and Andrew S. Tanenbaum, *Distributed Systems* 1/e, Prentice Hall, New Delhi.

6) Andrew S. Tanenbaum, *Distributed Operating Systems*, 1/e, Prentice Hall, 1995), New Delhi.

7) Jean Bacon, *Concurrent Systems: An Integrated Approach to Operating Systems*, *Database, and Distributed Systems*, 2/e, Addison Wesley, New Delhi.

8) Jean bacon and Tim Harris, *Operating Systems: Concurrent and distributed software design*, Addison Wesley, 2003, New Delhi.

9) Coulouris, Dolimore and Kindberg, *Distributed Systems: Concepts and Design* 3/e, Addison-Wesley, New Delhi.

10) D.M. Dhamdhere, *Operating Systems – A Concept Based Approach*, Second Edition, TMGH, 2006, New Delhi.

UNIT 3 MOBILE OPERATING SYSTEMS

Structure

3.1 INTRODUCTION

In the earlier units we have studied about the functions of traditional and desktop operating systems and in this unit we will study about the Mobile devices and Mobile Operating Systems. Earlier mobile communication technologies were dominated by vertically integrated service provision which are highly bounded mainly to voice and short message services that are organized in a monopolistic competition between few mobile virtual network operators, service providers and enhanced service providers.

In the recent years, however, radical change driven by advancements in technology, witnessed the introduction and further development of smart phones where the user can get access to new applications and services by connecting to the device manufacture's application stores and the like. These smart phones have added many features of a full-fledged computer: high speed processors, large storage space, multitasking, high-resolution screens and cameras, multipurpose communication hardware, and so on. However, these devices market is dominated by a number of different technological platforms, including different operating systems (OS) and application development platforms, resulting in a variety of different competing solutions on the market driven by different actor. This unit is focused on mobile devices, their characteristics and mobile operating systems.

3.2 OBJECTIVES

After going through this unit, you should be able to:

- define and identify various mobile devices;
- understand the characteristics of mobile devices along g with their constraints;
- define the mobile OS and identify its functionalities, and
- describe various design issues of mobile OS.

3.3 MOBILE DEVICES – AN INTRODUCTION

A wide variety of mobile devices are available to address a broad range of applications and users. They range from very inexpensive web-enabled devices to high-end customized tablets, with laptops, a variety of PDAs, and smart phones in between. Along with size differences come variations in the features and performance that these devices provide. No matter which type of mobile application you are looking to deploy, a device is available that will meet your needs. Let us study about some of the mobile devices:

Smartphones

Smartphones have taken our society by storm. If you don't already have one, you want one. Examples include the iPhone and Android phones, including the Google Pixel phone. Smartphones are advanced versions of traditional cell phones in that they have the same features as cell phones — such as the ability to make and receive phone calls, text messages and voicemail — but they can also be used to browse the internet, send and receive email, participate in social media and shop online. They also can download apps from the internet using a cellular or Wi-Fi connection to expand the smartphone capabilities in a vast number of ways.

Tablets

Tablets are portable, like laptops, but they provide a different experience. Instead of running a traditional laptop and desktop computer applications, they run apps designed specifically for tablets. The experience is similar, but not the same as using a laptop computer. Tablets come in all sizes, from slightly larger than a smartphone to the size of a small laptop.

Although you can buy a separate keyboard accessory, tablets come with virtual onscreen keyboards for typing and inputting information. They use touch-screen interfaces, and the familiar mouse is replaced with a tap from a finger. There are many tablet manufacturers of tablets, but among the best-reviewed are Google Pixel, Samsung Galaxy Tab, Nexus, and Apple iPad.

E-Readers

E-Readers are specialized tablets that are designed for reading digital books. Those digital books can be purchased or downloaded free from online sources. Well-known e-reader lines include Barnes & Noble Nook, Amazon Kindle and Kobo, all of which are available in several models. You can also read digital books on tablets that have an ebook app installed. For example, Apple's iPad ships with

iBooks and supports downloadable apps to read Nook, Kindle and Kobo digital books.

Other Mobile Devices

Some portable music players have access to the internet and can download apps to enhance their value to their owners. Apple's iPod touch is an iPhone without the phone. In all other respects, it offers the same experience. Sony's high-end Walkman is a luxurious audio player with Android streaming apps.

PDAs, the business person's best friend for years, fell out of favor with the introduction of smartphones, but some are being re-imagined with Wi-Fi access and with rugged designs that make them useful to the military and people who work outdoors.

Wearables, like smartwatches and fitness trackers, are among the newest additions to the mobile device landscape. Many are powered by the same mobile operating systems as phones and tablets, and they're capable of running their own apps. Most wearable devices are made to pair with another mobile device, like a smartphone to share data and create an altogether more convenient experience.

3.2.1 Characteristics of Mobile Devices

Mobile devices come in all sizes with a wide range of features. Choosing the correct device involves evaluating a variety of criteria other than just cost. Each of these is described in detail later in this section. Some of the characteristics of the Mobile devices are:

Limited Processing Power: The processors used in these devices will be low processing frequency normally in the range of 50MHz to 100MHz.

Small Screen Size: These devices will have a very limited screen size due to limited space available on the device.

Limited Batter Power: These devices have limited battery power and cannot be increased beyond a limit. Power management is a big challenge for the mobile devices. In case of any power failure, the data safety also needs to be provided.

Limited Memory: There are 2 types of memories – ROM and RAM. ROM is for the Mobile Operating System and pre-installed programs and RAM is for users processing of data. Instead of a hard disk, mobile devices use flash memory as non-volatile memory.

There are three major factors those need to be considered – (i) Data input mechanism, (ii) wireless connectivity options and (iii) the mobile operating system that the device is using.

3.2.2 Data Input Mechanisms for Mobile Devices

Along with device size and power, the data input mechanism is one of the most important aspects of selecting a mobile device. This does depend on the application, and the levels of user interaction required, but for most m-business applications where a substantial amount of data is entered into the application, data input options have to be given strong consideration. When it comes to methods of data input, several options are available.

a. Keypad Input (Earlier Model)

Mobile phones typically implement a 12-digit keypad. This is very intuitive and effective for numerical input but cumbersome and awkward for entering text. This is because each key on a keypad represents three characters. Users have to press a key multiple times to enter a single character. Since mobile phones are primarily designed for voice usage, it is unlikely that the keypad will change anytime soon. That said, there are technologies available that make entering text easier. One of these is called T9, which stands for Text on 9 Keys. This is a predictive text input technology that allows a user to input words by pressing only one button per letter. T9 translates the button press sequences into words. It does not work in all cases, but when it does, it is about twice as fast as using traditional multi-tap systems. America Online, the owner of T9, estimates that it is accurate 95 percent of the time.

Device manufacturers are constantly looking for ways to make keypad text entry more effective. Toggle sticks, rollers, and touch screens will help to alleviate this problem in some areas, but do not expect fast data entry to be a reality on a keypad-driven device anytime soon.

b. Physical Keyboard Input

Even with the advances being made in other forms of data input, the keyboard remains the most efficient and easy to use. Mobile devices have many ways to take advantage of keyboard data entry. Laptops and handheld PCs come equipped with physical keyboards, while the smaller palm-sized PDAs often support keyboards as a peripheral. These keyboards are usually attached as a clip-on unit or via a cable, but wireless connections are also an option. In addition, most PDAs have PC companion software, allowing the user to enter data such as contact information on the PC. Some handheld devices have taken the step of incorporating a keyboard into the unit itself. This is accomplished with small thumb-based keyboards at the bottom of the unit. Research in Motion (RIM) was the first to incorporate this concept into its BlackBerry devices. With its revolutionary design, RIM has attracted a wide variety of users to its devices, prompting other companies, including Handspring and Sharp, to adopt the thumb-sized keyboard as the means for data input.

c. Pen-Based Input

One of the breakthroughs for handheld devices was the introduction of touch screens for pen-based input. This allows a user to enter data using a stylus, without requiring any form of physical keyboard or keypad. Several pen-based input mechanisms are available. Deciding which ones to implement will depend on the mobile operating system and device being used. The most common types of pen-based input are:

- **Soft keyboards.** The most approachable method of entering text on mobile devices is by using a soft keyboard, a keyboard displayed on the screen. Users enter data by pressing “keys”, just as they would on a physical keyboard. This method is very easy to learn and use, but it is also takes up screen space and can be limiting in terms of input speed. It is often used as a backup mechanism when other forms of data input are not effective.

- **Character recognition.** Character recognition works by interpreting which letters a user is entering on the screen. There is usually a specific area on the screen for users to construct the characters. The Windows CE operating system from Microsoft provides this capability. Software runs locally on the device to do the interpretation. The letters have to be constructed fairly accurately for them to be understood. This is usually not a problem for small amounts of data, but can be a challenge for users who need to enter data quickly. They often find the characters difficult to write, leading to mistakes.
- **Graffiti.** Palm OS introduced a new form of character recognition based up on simple character-set called Graffiti. The Graffiti characters allow for quicker data input since the letters are easy to input and recognize. There is a learning curve involved, but once Graffiti is mastered, data can be input quickly with few errors.
- **Handwriting recognition.** Some forms of handwriting recognition work by taking screen captures of the inputted text, while others actually interpret written characters. The interpretation solutions take character recognition to a new level. Instead of being able to interpret only a defined set of characters with predetermined strokes, handwriting recognition attempts to interpret a user's personal style of text entry. This requires sophisticated software and powerful hardware to do the analysis. Microsoft is implementing forms of handwriting recognition on its Tablet PC platform.

d. Voice Input

Voice input is both effective and easy to use. Simple voice commands, such as initiating a call or looking up a contact, can be executed directly on the mobile device. This works by prerecording voice commands into the device. When a voice command is entered, it is compared to the recorded message and executes the corresponding action. Many mobile phones come equipped with this feature.

3.2.3 Wireless Communication

Three basic connection options are available for obtaining wireless communication from a wireless device:

- Two-unit configuration
- Detachable, and
- Integrated

(i) Two-Unit Configuration

Two-unit connections require two pieces of equipment to work together, such as a PDA and a cell phone. One unit provides the wireless connectivity for the other unit to use. These devices can communicate with each other in a variety of ways:

- **Cable connection:** This involves having a physical cable connecting the cell phone and mobile device. The phone must have wireless data support as well as an interface cable to the device of choice.

- **Infrared connection:** This involves lining up the Infrared ports on the cell phone and mobile device for communication. A direct line of sight is required for this solution to work. In addition, only selected phones have infrared support available.
- **Bluetooth connection:** Bluetooth may be the best option for two-unit connectivity. It allows a mobile phone to provide connectivity through a personal area network (PAN) up to a range of about 10 meters. A direct line of sight is not required, so it is possible to have the cell phone in a different location from the mobile device. This will become an increasingly popular option as more Bluetooth-enabled devices are released.
- **Near Field Connection**: It is incorporated into may payment cards, ticketing and the like to enable swift and very easy transactions to be made.NFC technology is also used in many other areas where short range secure communications need to be made, and can even be incorporated into mobile phones and other devices. NFC is standards-based technologies used to provide short range wireless connectivity technology that carry secure two-way interactions between electronic devices. Communications are established in a simple way, not requiring set-up by users as in the case of many other wireless communications. As such NFC enables users to perform contactless transactions, access digital content and connect electronic devices by touching devices together.NFC near field communication provides contactless communication up to distances of about 4 or 5 cms.

The advantage of the two-unit configuration is that you can choose each device based on its own functionality, and you do not have to sacrifice features for wireless capabilities. The configuration also provides a great degree of flexibility. If one device becomes outdated or malfunctions, it can be replaced without having to replace the entire system. This is especially appealing when the data device is a laptop or PDA that may not need to be replaced as frequently. In addition, the cell phone can potentially provide the wireless communication for several other devices.

The downside to this configuration is its complexity. Both the cell phone and the mobile device have to be configured properly for this to work. When Bluetooth is not available, the cable or Infrared connections can become cumbersome.

This setup is recommended only where occasional wireless connectivity is required. When frequent access is necessary, it is worthwhile to investigate some of the other options available.

(ii) Detachable Configuration

A detachable configuration involves using a plug-in module or clip-on attachment to a mobile device to provide wireless connectivity. These additional modules can provide connectivity to a variety of wireless networks with little configuration. For the plug-in modules, either CompactFlash or PCMCIA cards can be used. This obviously requires a device that has a slot for this type of peripheral. For devices that do not have integrated CompactFlash or PCMCIA slots, external jackets are often available. These jackets (often called sleds) provide the support for CompactFlash or PCMCIA cards.

The benefit of using these interfaces is that multiple wireless modems are available, providing connectivity to several wireless networks including WLANs, WANs, and PANs. You can then switch interface cards as you move into different wireless environments. Cards are also available that support connecting to multiple network protocols in one unit.

In some cases, the only available option is a clip-on modem. This is similar to the external jacket, except that the modem is incorporated directly to the unit. These modules are often purchased in conjunction with a wireless service plan since they only provide access to a single wireless protocol, most commonly a wireless WAN.

The majority of the PDAs available now use a detachable configuration for wireless connectivity. It allows users to select the PDA and wireless component separately, but at the same time have them integrate well together. The disadvantage is that the wireless modem uses the open expansion slot so other peripherals cannot be used at the same time.

(iii) Integrated Configuration

An emerging trend is to enclose wireless connectivity within the mobile device. This has always been the case for voice-oriented devices such as cell phones and smart phones, but is a new concept for many handheld devices. This configuration has many advantages, as it alleviates complexity and provides tight integration between the mobile OS and wireless modem. Applications for these devices can be designed to take advantage of the wireless modem, knowing that it is always going to be present. Troubleshooting is also simplified, since there is only one manufacturer involved in the solution.

On the downside, some flexibility is lost. The user is now limited to the wireless net-work type that has been integrated into the device. In addition, it may become more difficult for developers to calculate the amount of power an application requires, as they will have a hard time differentiating between the power that the application requires and that of the wireless modem.

3.2.4 Mobile Operating System

A mobile operating system (mobile OS) is a system software that allows smartphones and other handheld / mobile devices to run applications as well as manage the device's precious resources like processing power, battery and memory.

A mobile OS typically starts up when a device powers on, presenting a screen with icons or tiles with information and through which applications can be accessed. Apart from user applications, mobile operating systems also manage network connectivity.

Android, Apple iOS and Blackberry OS and Widows Mobile OS are some of the famous mobile operating systems which provides the combined features of personal computer operating systems and the features including touch screen, cellular, Bluetooth, Wi-Fi, GPS navigation system, camera, speech recognition, voice recorder, music player etc.

More details regarding Mobile OS are given in section 3.3 and onwards.

3.2.5 Types of Processors Used by Mobile Devices

Let us study on the processors those are used the mobile devices. In the recent days most of our devices are come with the Octa-core or Quad-core processors. In the processors the main element is the core that reads and executes instructions. Devices began with a single-core processor, but processor companies have created more powerful devices by including more cores in one device that led to dual-core devices. Soon there were quad-core processors (as in four cores), and now there are hexa- (six) and octa-core (eight) smartphones and tablets.

Benefits of Multiple Cores

The more cores, faster they can divide the work you're requesting the phone to do. So the benefits of a multi-core are it enhances your device performance and your apps loads quickly on your device. You can easily capture high-quality photos and videos then browse your collection without pausing the system.

Some Popular Processors for Mobile Devices

i. **Qualcomm Snapdragon** : It is a US based company, Qualcomm was the first brand that introduce CDMA technology in the market and actively involved in technology related to semiconductors designing for Mobile devices, tracking devices, satellite phones, Virtual Reality, wireless charging etc. It is widely known for its Snapdragon brand which is responsible for releasing mobile processors and LTE modems. It first claim to fame in processor market when it released the first 1 Ghz processor that time average speed of most smartphones was only 512 Mhz. The processor comes under 4 series (220, 400, 600 and 800). Qualcomm Snapdragon processors are known for its performance. The processor handles multitasking very well and can handle heavy and intensive processing which is especially good for gaming. The presence of inbuilt Adreno graphics matches the performance of processor. Snapdragon processors also produce less heat compared to other processors. Generally Snapdragon processors are costlier than other processors. Qualcomm also make processors are also used for wearable platforms, automotive platforms, embedded platforms, vision intelligence platforms and extended reality platforms.

ii. **Apple Mobile Processors**: Apple does not manufacture any microprocessors. Instead, they make contracts with processor manufacturing companies mainly Samsung and TSMC for making custom built processors that suits their design and performance expectations. The processor comes under

   a. A- Series: The A series is a family of "System on Chips" (SoC) used in iPhones, iPad, iPod touch & Apple TV. They are designed by Apple and manufactured by Samsung & TSMC. They integrate CPU, GPU, Cache memory and other electronics necessary to provide mobile computing functions within a single physical package.

   b. S- Series: The S series is a family of "System in Package" (SiP) used in Apple watch. They are designed by Apple and manufactured by Samsung.

   c. T- Series: The T-Series is designed to use in TouchID sensors in MacBook Pro. The only version released till now is Apple T1.

d. W-Series: Apple W series is used in headphones for wireless audio connectivity. The series is currently in Apple W1 which is used in wireless headphones and AirPods.

e. H-Series: The Apple H1 chip was first used in the 2019 version of AirPods, and was later used in the Powerbeats Pro, the Beats Solo Pro and the AirPods Pro. Specifically designed for headphones ("H" in a model number stands for *headphones*), it has Bluetooth 5.0, supports hands-free "Hey Siri" commands, and offers 30 percent lower latency than the W1 chip used in earlier AirPods.

f. M-Series: The M1 chip, Apple's first processor designed for use in Macs, is manufactured using TSMC's 5 nm process. It was announced on November 10, 2020, and is used in the M1 MacBook Air, Mac mini, and MacBook Pro (2020).

iii. **Intel Atom and Core M Processors:** Intel is an American multinational company synonymous with PC and microprocessors. Atom is the brand name given for the low power consuming and low cost 32 and 64 bit chips manufactured for using in smartphones and tablet devices.Intel processors are based on X86 architecture which is powerful than ARM, but consume more power compared to ARM architecture. Intel Atom processors are currently in Atom X5 and X7 series. These chips are 64 bit Quad core processors in 14 nanometer size with speeds up to 1.6 GHz that can be scaled up to 2.4 Ghz.Intel also released Intel Core M microprocessors which are ultra low-voltage designed for ultra-thin notebooks, mobile devices and 2 in 1 convertibles. The processor consumes less 4.5 watts or less power making it ideal for long battery life. These are dual core processors with speeds around 1.5Ghz which can be scaled to 3.2 GHz speeds. Intel Core M processors offer 40% boost in CPU and graphics performance than previous versions.

iv. **Nvidia Tegra Processors:**Nvidia Corporation is an American based company technology company which specializes in making processing units for graphics, gaming units and mobile devices. Nvidia develops chips for smartphones, tablets and mobile devices under the brand Tegra.Tegra processors are built on 64 bit ARM architecture. Tegra till now have gone through Tegra 1, 3, 4, 4i, K1, X1 series. The process is Quad Core with 256 GPU cores capable of 4K video capabilities. The chips are built on 20 nm technology. The processor is currently used in Nvidia SHIELD Android TV. Tegra processors used in Andorid TVs, smartphone and tablets.

v. **Samsung Exynos:** Exynos is a brand of Samsung Electronics which makes processors based on ARM architecture. Exynos processors are manufactured in the series of **ARMv7** and **ARMv8** SoCs. Exynos 8 Octa 8895 is the latest from Exynos. The processor is equipped with Octa-Core on 64-bit ARM architecture with Mali GPU. The processor is capable of running at 2.3 GHz speed with support for 3D gaming, 4K UHD resolution support. The chips are 10 nm technologies. The processor is used in Galaxy S8 and S8+ phones.

3.2.6 Limitations of Mobile Devices

There are some general limitations for mobile computing devices which are given below:

- **Insufficient bandwidth**: Mobile Internet access is generally slower than direct cable connections, using technologies such as GPRS and EDGE, and more recently HSDPA and HSUPA 3G/4G networks. These networks are usually available within range of commercial cell phone towers. Higher speed wireless LANs are inexpensive but have very limited range.
- **Security standards**: When working on mobile, one is dependent on public networks, requiring careful use of VPN. Security is a major concern while concerning the mobile computing standards on the fleet. One can easily attack the VPN through a huge number of networks interconnected through the line.
- **Power consumption**: When a power outlet or portable generator is not available, mobile computers must rely entirely on battery power. Combined with the compact size of many mobile devices, this often means unusually expensive batteries must be used to obtain the necessary battery life.
- **Transmission interferences**: Weather, terrain, and the range from the nearest signal point can all interfere with signal reception. Reception in tunnels, some buildings, and rural areas is often poor.
- **Potential health hazards**: Cell phones may interfere with sensitive medical devices. Questions concerning mobile phone radiation and health have been raised.
- **Human interface with device**: Screens and keyboards tend to be small, which may make them hard to use. Alternate input methods such as speech or handwriting recognition require training.

3.3 MOBILE OPERATING SYSTEM

A mobile operating system (Mobile OS) is a software platform on top of which other programs called application programs, can run on mobile devices such as personal digital assistant (PDA), tablets, cellular phones, smart phone etc.. Over the years, Mobile OS design has experienced a three-phase evolution: from the PC-based operating system to an embedded operating system to the current smart phone-oriented operating system in the past decade. The mobile OS is designed keeping in view the physical and other limitations of the mobile devices. Depending the capabilities of the Mobile devices and the way they support, mobile OS may also differ with the devices. Mobile operating System concept becomes popular in the decade of 1990s. In 1994 IBM introduced first Smartphone called IBM Simson. It was designed with a file system from Datalight ROM-DOS. In 2000, the first modern mobile OS was introduced by Symbian. In 2007, Apple introduced the popular iOS with a lot of internet related capabilities. In 2008, Google introduced a popular Android mobile OS. Later 2010, various Mobile OS start competing for each other to bring most user friendly features.

Many mobile devices are available in the current market. These devices are running with a mobile OS. Few of devices manufactures are using their own operating systems but few others are using the available Mobiles OS from the market with or without modifications. Below are the list operating systems using in the major mobile devices.

Throughout the process, Mobile OS architecture has gone from complex to simple to something in-between. The evolution process is naturally driven by the technology advancements in hardware, software, and the Internet:

i. *Hardware:* The industry has been reducing the factor size of microprocessors and peripherals to design actual mobile devices. Before the form factor size was reduced enough, the mobile device could not achieve both small size and processing capability at the same time. We had either a PC-sized laptop computer or a much weaker personal data assistant (PDA) in phone size. Mobile operating systems for PDAs usually did not have full multitasking or 3D graphics support. Features like sensors, such as accelerometers, and capacitor-based touch screens were not available in the past mobile operating systems.

ii. *Software:* With a laptop computer, the software is mainly focused on the user's productivity, where support for keyboard and mouse that have precise inputs are essential. The software for a personal data assistant, as its name implies, helps the user to manage personal data such as contacts information, e-mail, and so on. The mobile operating systems were not designed for good responsiveness or smoothness with a rich user interface (UI) including both touch screen and other sensors.

iii. *Internet:* Along with Internet development, especially after Web 2.0, there is abundant information in the network waiting to be searched, organized, mined, and brought to users. People are increasingly living with the Internet instead of just browsing the Web. More and more people are involved in the development, including information contribution, application development, and social interactions. The mobile operating systems cannot be self-contained, but have to be open systems.

The aforementioned technological advancements have resulted in a variety of different competing mobile operating system solutions on the market driven by different companies. Some of these includes Google's Android, Apples' iOS, Nokia's Symbian, RIM's BlackBerry OS, Samsung's Bada, Microsoft's Windows Phone, Hewlett-Packard's webOS, and embedded Linux distributions such as Maemo and MeeGo.

3.4 EVOLUTION OF MOBILE OS

The ever growing, innovating, and awesome world of smartphones has a history as everything else. Back from the days of Palm OS to Android Honeycomb where more than half a dozen Smartphone Operating Systems are available in the global market.

Presenting below in the Table 1 is a timeline view of how the evolution has taken place from 1996 – 2021:

Table 1: Evolution of Mobile OS

Name of the Mobile O/S	Year of launch	New Features
Palm OS 1.0	1996	RIM applications Address, Datebook, Memo pad, To-do list

Palm OS 2.0	1997	Mail and Expense are added
Palm OS 3.0	1998	HotSync Support, Web Clipping Support, native 8-bit color support
Pocket PC 2000	2000	
Pocket PC 2002	2001	MSN Messenger, Media Player 8 Enhanced UI
Palm OS 4.0	2001	External File Systems, 16-bit color screens
Palm OS 5.0	2002	PACE Emulator, Bluetooth Emulator
Windows Mobile 2003	2003	Bluetooth Integration, Pocket Internet Explorer, Windows Media Player 9.0
Windows Mobile 2003 SE	2004	SE Potrait and Landscape switching for Pocket PCs and WPA
Palm OS Cobalt	2004	Telecommunication, Wi-Fi and Bluetooth Connectivity
Windows Mobile 5	2005	Windows Media Player 10 Mobile, Global Positioning System (GPS) Management Interface, Introduction to Office Mobile
Blackberry OS 4.1	2005	
iPhone OS 1.0	2007	Core iOS UI, Web Clips on Home Screen, Multitouch Gestures, Mobile Safari, Visual Voicemail, Maps, iTunes Sync, iTunes Wi-Fi Music Store, Multitouch Keyboard
Windows Mobile 6	2007	.NET Compact Framework v2 SP2 Microsoft SQL Server 2005 Compact Edition Windows Live
Blackberry OS 4.2	2007 folders	Voice Notes Option,Email and SMS in separate
iPhone OS 2.0	2008 contacts	Third Party Application support, Sync Google
Blackberry OS 4.5	2008	HTML emails, faster performance and improved multi-tasking, Microsoft Office Documents ToGO
Windows Mobile 6.1	2008	Threaded SMS full page, zooming in Internet Explorer and Domain Enroll
Android 1.0 Alpha	2008	Google Apps like Gmail, Maps, Calendar and Youtube
Symbian OS	2008	Desktop Interactive Widgets, FaceBook IM chat
Android 1.1 Base	2009	Support for saving attachments from MMS, Marque in layouts, API changes
Blackberry OS 5	2009	Wireless Sync, Blackberry Enterprise Server 5, almost revamped web browser

Windows 6.5	2009	Internet Explorer Mobile 6 and Multi touch support
Samsung Bada 1.0	2009	
iPhone OS 3.0	2009	Push Notifications, cut, copy and paste, Turn-by-Turn Navigation, Voice memos
HP Web OS	2009	Synergy app, multi-touch gestures and multi-tasking
Android 1.5 (Cup cake)	2009	Bluetooth A2DP and AVRCP support, Uploading videos to YouTube and pictures to Picasa
Android 1.6 (Donut)	2009	WVGA screen resolution support Google free turn-by turn navigation
Android 2.0/2.1 (Éclair)	2009	HTML5 support, Microsoft Exchange Server, Bluetooth 2.1
iPhone iOS 4.0	2010	Multitasking, Folders
Blackberry OS 6	2010	New media interface, stronger social media integration, Multiple Contact Lists, track pad support for swipe gestures
Windows Phone 7	2010	Tiled UI, Loud-based services support, Multitasking
Android 2.2 (Froyo)	2010	USB tethering and Wi-Fi hotspot functionality, Adobe Flash 10.1support
Android 2.3 (GingerBread)	2010	Multi-touch software keyboard, Support for extra large screen sizes and resolutions
Symbian^2 OS	2010	Royalty-free version
Symbian^3 OS	2010	Native Webkit browser, 2D and 3D graphics architecture, UI improvements and support for external displays through HDMI
Android 3.0 (HoneyComb)	2011	Optimized tablet support, with a new UI, 3D desktop, Video chat with Gtalk support
iOS 5	2011	Siri, Notification center, PC-Free, iTunes Wi-Fi Sync, iMessage, iCloud
Android 4.0 (Sandwich)	2012	It combined many of the options of the tablet-only Honeycomb version with the smartphone-oriented Gingerbread.
Android 4.1 (jelly Bean)	2012	Some of the new additions in these software updates included new notification features that displayed more content or action buttons, along with full support for the Android version of Google's Chrome web browser, which was included in Android 4.2.

iOS 6	2012	iCloud tabs, Siri enhancements, mail enhancements, FaceTime over Cellular, Facebook integration, Passbook, Homegrown Maps and Tun-by-Turn navigation
Windows Phone 8	2012	Replaced Windows CE-based architecture
iOS 7	2013	Visual overhaul, control center, AirDrop, iTunes Radios, FaceTime Audio, Refreshed Core Apps
Android 4.4 KitKat	2013	It was optimized to run on smartphones that had as little as 512 MB of RAM.
Windows Phone 8.1 Start Screen.	2014	Notification center, support for Internet Explorer 11 web browser, tab syncing among Windows 8.1 devices, separate volume controls and the option to skin and add a third column of live tiles to the
iOS 8	2014	Continuity, QuickType, Widgets, iCloud Drive, Extensibility, Healthkit, Homekit, Family Sharing
Android 5.0 Lollipop	2014	It was the first version of the OS that used Google's new Material Design language. It made liberal use of lighting and shadow effects, among other things, to simulate a paper-like look for the Android user interface.
Windows 10 mobile	2015	Mobile OS for smartphones and tablets running on ARM architecture. Its primary focus is unification with Windows 10, its PC counterpart.
iOS 9	2015	Night Shift, Lower Power Mode, Public Beta Program
Android 6.0 Marshmallow	2015	It included features such as a new vertically scrolling app drawer, along with Google Now on Tap, native support for fingerprint biometric unlocking, USB Type-C support, the introduction of Android Pay (now Google Pay), and much more.
Android 7.0 Nougat	2016	Nougat's many new features included better multi-tasking functions for the growing number of smartphones with bigger displays, such as split-screen mode, along with quick switching between Apps. Google made a number of big changes behind the scenes too. It switched to a new JIT compiler to speed up apps, supported the Vulkan API for faster 3D rendering, and enabled OEMs to support its now-defunct Daydream VR platform
iOS 10	2016	iMessage Apps, Delete built-in Apps

Android 8.0 Oreo	2017	As far as features go, Android Oreo packed in lots of visual changes to the Settings menu, along with native support for picture-in-picture mode, notification channels, new autofill APIs for better management of passwords and fill data, and much more. Android Oreo first came installed on Googles' own Pixel 2 phones.
iOS 11	2017	Augmented Reality, Major Enhancements on iPad, AirPlay 2
Android 9.0 Pie	2018	Android 9.0 Pie also included some new features designed to help extend your smartphone's battery life. That was achieved with the use of on-device machine learning which predicts which apps you will use now, and which apps you won't use until later. Pie also has Shush, a feature that automatically puts your phone in Do Not Disturb mode when you flip your phone screen down on a flat surface.
iOS 12	2018	Grouped Notifications, ARKit 2, Siri Improvements, Screen Time, Memoji
Android 10	2019	Android Q is officially known just as Android 10. The features included support for the rush of then-upcoming foldable phones. Android 10 also introduced a system-wide dark mode, along with new gesture-navigation controls, a more efficient sharing menu, smart reply features for all messaging apps, and more control over app-based permissions.
iOS 13	2019	System-wide Dark Mode, New Improved Siri Voice, Overhauled Stock Apps like Reminders and Notes, New Privacy and Security Options, New Potrait Lighting Option, Sign In with Apple User Account System
Android 11	2020	That includes a new Conversations notification category where all of your chats from various apps are collected in one place. You also have the option to save every notification that has appeared on your phone in the past 24 hours. A brand new feature lets you record your phone's screen, complete with audio, without needing a third-party app. There's also a new section of Android 11 dedicated to controlling smart home devices.

In the following section let us study the need for the mobile OS.

3.5 NEED FOR THE MOBILE OS

The Mobile OS powers mobile devices like mobile phones, Smartphones, PDAs, and Tablet PCs (a wireless PC equipped with a touch screen and a digital pen). It

manages the mobile device hardware, memory, and software resources. It controls the functioning, and the capabilities of a mobile device. The feature set, security, reliability, and the ease of use are not the only criteria that make a mobile platform good or bad, but also the flexibility of the mobile platform to integrate with diverse set of devices and software systems.

A mobile operating system controls everything from handling the input obtained from touch screen, keyboard, or some external device to controlling the memory and the overall functioning of the device. It also manages the communication and the interplay between the mobile device and other compatible hardware such as, computers, televisions, or printers.

Operating system manages and controls all the features and functionalities of the mobile device. Therefore, a Smartphone is a combination of the hardware and the operating system, which determines what capabilities it can or cannot support.

As the operating system manages the hardware and software resources of Smartphones, it is responsible for determining the functions and features available on the device. Smartphone is a combination of mobile technology that is the mobile phone and PDA, which are based on the computer applications

All Smartphones are embedded with an operating system which enables the operation of software applications. In addition to the principle features like phone calls and messaging, you can send e-mails, manage your personal and office documents, and visit websites for searching information, play online games, and read news. It also allows sharing and downloading of documents and applications.

If a Smartphone is to allow multiple applications to run simultaneously, it must have an operating system that facilitates the sharing of processing and memory resources among multiple applications. Additionally, the operating system must allow users to switch between the active applications.

3.6 CHARACTERISTICS OF SMARTPHONE OS

Even though the functionality of an operating system used in computers and Smartphones are same, certain characteristics of a Smartphone operating system are different from the ones used in computers. The characteristics that a Smartphone operating system should comprise are as follows:

(i) ***Resource-limited hardware:*** Smartphones should be able to support various applications. It should also provide facility to access Internet. But to meet these requirements, Smartphones have limited memory and processing power when compared to the desktop PCs and Laptops. Thus, the operating system must be careful in using hardware resources especially memory. It should not only utilize less memory but also consist of architecture that provides support for applications to limit their use of memory. It should also have the capability to handle low-memory situations gracefully.

(ii) ***Robustness***: A user expects a mobile operating system to be robust. This means it should be strong and unlikely to fail or crash. The device must not only be designed to avoid crash, but must also provide support functions and policies. These support functions and policies allow the device to handle

application errors and out-of-memory situations, without hampering the functionalities of the Smartphone.

(iii) ***User interface for limited user hardware:*** The operating system should implement a user interface environment that is efficient and intuitive to use, despite the smaller screen and limited user input capabilities of the Smartphone. Furthermore, the screen sizes and input capabilities vary between different models of Smartphones, so the User Interface architecture should be flexible, such that it can be customized for the various user interface objects.

(iv) ***Library support:*** Smartphone operating systems should contain middleware libraries and frameworks with APIs that implement and abstract the functionality of the features of the Smartphone. The purpose is to provide functional consistency and to ease the software development. Middleware library and framework is a software layer that acts as a mediatory between the application and the system's operating system. The middleware framework consists of a set of components that connects the application with the underlying OS. Examples of Smartphone middleware include libraries and frameworks for email, SMS, MMS, Bluetooth, cryptography, multimedia, User Interface features, and GSM or GPRS, which provide more support for Smartphone features.

3.7 DESIGN ISSUES IN MOBILE OS

Since, the mobile devices are different as compared to conventional desktop systems; the OS for them would differ. Keeping in view of the constraints of mobile devices, the design issues of Mobile OS will vary when compared with the conventional desktop versions. Let us discuss some of the design issues for a mobile OS:

Power Management

In the power management system power management, device power management and processor power management need to be considered.

Battery Management

Mobile OS collects the information about the state of the batteries of the device and inform the user. In case of abnormal consumption or battery discharge due to some application/game, OS must perform an emergency shutdown keeping the data loss at minimum level.

Thermal Management

As the mobile devices operate with the batteries, heat dissipation is also a challenge. It is one of the functions' of the Mobile OS to manage and reduce the power consumption of the devices.

Memory Management

Mobile OS should consume less space in RAM. Instead it should have a small piece of code existing in the RAM. However, to avoid RAM space there should be a provision that the kernel, applications and libraries should be directly executed in ROM. The microkernel structure of mobile OS also may help in reducing the

memory size. Also memory leakages need to be taken care by the Mobile OS. Efficient memory management techniques need to be adopted while designing the mobile OS.

Scheduling

The mobile OS may adopt the pre-emptive scheduling mechanism. The interrupt latency and dispatch latency must be minimized in mobile OS.

File System

Flash file systems are used in mobile devices. They offer a fast-read access. The file system should adopt the memory management techniques that minimize the fragmentation problem, caching techniques and intelligent reclaim techniques.

Security

One of the major issues to be considered is Security. Unlike desktops, these mobile devices also may get infected with worms, Trojan horses or viruses wherein compromising the user's security and privacy. As all the mobile devices are prone to attacks any time, it is therefore the major function of the mobile OS to provide security mechanisms to address all the security issues.

In Block-4 along with case studies, the handling of various functions by the individual Mobile OS is discussed.

3.8 POPULAR MOBILE OS

Some of the popular mobile operating systems are:

- Android
- iOS
- Symbian
- Windows Mobile
- Linux

Android

Android is an operating system for mobile devices that is developed by Google. Android operating systems are based on the Linux kernel and the GNU software. Android has a large community of developers writing applications that has helped to extend the functionality of the devices. The developers write managed code in Java, controlling the device via Google-developed Java libraries. Android is a mobile operating system that has an open-source framework and is based on Linux which helps us to develop advanced and user-friendly applications. It follows a Monolithic kernel approach

Now, we will start with Android Architecture, it comprises of five levels, which are the Linux kernel, Libraries, Application framework, Android runtime, and System applications.

The Android operating systems offer a virtual machine that is optimized for the mobile devices. It provides a structured data storage by the use of SQLite. It

facilitates with technologies such as, Bluetooth, 3G and WiFi that are hardware dependent. It enables reuse and replacements of components through an application framework. For example Sony Ericsson XPERIA X10 is developed using Android operating system.

A virtual machine can be referred to an environment or a program that does not physically exists but is created inside another environment.

You can study in detail on Android in Block-4 where Case Studies are discussed.

iOS

iOS is an operating machine based on the UNIX environment constructed for Apple's iPhone, iPod touch and iPad of different cell devices. The iOS architecture is layered. It follows a hybrid kernel approach. It contains an intermediate layer between the applications and the hardware so they do not communicate directly. The lower layers in iOS provide the basic services and the higher layers provide the user interface and sophisticated graphics. iOS is also used to manage the hardware of a gadget and for supplying applied sciences required to enhance each and every application and internet applications also. iOS used to be first introduced and launched as the operating system gadget of the iPhone introduce on the twenty-ninth of June, in the year 2007. The iOS has also consisted of apps like phone, messaging, core services of CF network, security services as keychain and certificate and trust services and core os.

You can study in detail on iOS in Block-4 where Case Studies are discussed.

Symbian

Symbian operating system is the most popular operating system used in most Smartphones and mobile phones today. The heritage of Symbian OS begins with some of the first handheld devices. This operating system began its existence in 1988 as SIBO (an acronym for '16-bit organizer'). SIBO ran on computers developed by Psion Computers, which developed operating system to run on small footprint devices. It was designed with specific criteria that can be characterized by event-driven communications, using client-server relationships and stack-based configurations. Symbian OS model follows layered approach and contains UI frame work layer, Application services layer, Base service layer and kernel services and hardware interface layer. Symbian OS has a micro kernel architecture that provides robustness, availability and responsive.

Client-server describes the relation between two computer programs. The client program sends a service request to the server program. The server program fulfils the request. The Symbian operating system is developed using C++ programming language.

The Symbian operating system is a mobile operating system that was specially built to run on a Smartphone platform. It fits in the memory of a mobile phone because of its compatibility. It is considered as a full-fledged operating system.

Symbian operating system supports multitasking and multithreading. Many processes can run concurrently, they can communicate with each other and utilize multiple threads that run internal to each process. It facilitates good support for

graphics and data management. This operating system has a file system that is compatible with Microsoft Windows operating system. It even supports other file system implementations through a plug-in interface. It uses TCP/IP networking as well as several other communication interfaces, such as serial, infrared and Bluetooth. For example, Nokia's bestseller Smartphone 6600 was developed using the Symbian operating system.

Windows Mobile

Windows Mobile is an operating system used in various mobile phones and Smartphones. It encompasses the entire software stack from the kernel to the application interface. This operating system is compatible with the Microsoft Office suite of programs. The current version is called "Windows Mobile 6.5". It is based on the Windows CE 5.2 kernel. Additionally, third-party software development is available for Windows Mobile, and the software can be purchased via the Windows Marketplace for Mobile. Originally appearing as the Pocket PC 2000 operating system, most Windows Mobile devices come with a stylus pen, which is used to enter commands by tapping it on the screen.

Windows CE is a modular operating system that serves as the foundation for several classes of devices. Windows Mobile is best described as a subset of platforms based on a Windows CE. Currently, Pocket PC (now called Windows Mobile Classic), Smartphone (Windows Mobile Standard), and PocketPC Phone Edition (Windows Mobile Professional) are the three main platforms under the Windows Mobile umbrella. Windows Mobile is a Microsoft-defined custom platform for general use in Smartphones and PDAs. It consists of a Microsoft-defined set of minimum profiles (Professional Edition, Premium Edition) of software and hardware that is supported. It provides feature rich OS and interface for cellular phone handsets. It offers productivity features to business users, such as email, as well as multimedia capabilities for consumers.

Linux

Linux is unique among the other operating systems. This is because its development is driven by a community of developers rather than by a single company such as Symbian operating system in Nokia phones and blackberry OS in blackberry phones. According to ARCchart, the Linux operating system supports more processors than other operating system, though the most popular phone models still use the Symbian operating system. However, Linux is an open source operating system, where in, the developers constantly change and update it even at the kernel level platforms.

Smartphones based on the open source Linux operating system have emerged in the market. There are many advantages to using an open-source operating system like Linux. No cost and the opportunity to tap into the Linux open source community are appealing. This has made Linux grow, not only for the server and PC market space, but also in the embedded device area including handheld computers. For example, Motorola that is a notable supporter of Linux has released the A760 Smartphone based on this operating system.

The comparison of Symbian, iOS and Android are given below in Table 2.

Table 2: Comparative Chart of Symbian, iOS and Android

PARAMETERS	SYMBIAN	iOS	ANDROID
OS family	RTOS	Darwin	Linux
Vendor	Symbian Ltd. And Symbian foundation	Apple	OpenHandset Alliance, Google
Architecture	OS with applications, UI frame works and kernel on ARM Processor	This OS has kernel which interact with Driver run time, Kernel, UI	Divided into components (App Layer, libraries, runtime and Linux Kernel)
User Interface	User friendly	It has to load application and view it	UI is highly configurable
Developed in (Programming language)	C,C++,ME, Python, Ruby, Flash Lite	C, C++, Objective –C, Swift	C, C++, Java
App Store	Nokia Ovi Store	App Store	Google Play
License	Proprietary	Proprietary	Open source
Battery Demand	Less	Less	Highest
Security	Hard to crack	Hard to crack	Softest to crack
Voice Assistant	Vlingo 3.2	Siri	Google now
Sideloading	Available	Done by installing Xcode7	Available
Environment	QT, Carbide, Vistamax, Eclipse	XCode(Apple), Appcode	Eclipse(Google)
Processor used	ARM, x86	ARM 64,ARM.	ARM, x86, MIPS
Memory Utilization	Memory Management Unit and cache all resides on a System on Chip	Automatic reference counting. No garbage collection	Paging, Memory Map, No Swapping
Power Management	When power goes below threshold then switches to low resolution display and refresh it	Not optimized power management compared to other OS	Power Management is not optimal

☞ **Check Your Progress 1**

1) Explore Sailfish OS.

.....

.....

.....

.....

2) Mention the features of (a) HarmonyOS (b) KaiOS

.....

.....

.....

.....

3.9 SUMMARY

In this unit, mobile devices and mobile operating systems were discussed.

Mobile devices are the devices that are easily transported and typically are handheld and powered by a battery. They would have unique challenges in their design to enable usability, ruggedness, and reliability. Characterized by relatively lightweight and small screen size, they span a wide range of device types and applications, including mobile phones and tablets. Mobile devices also include custom devices designed for specific use cases, such as military, medical, or industrial applications where particular peripherals, security, certification, or regulatory requirements must be fulfilled.

Users of mobile devices require a number of characteristics: portability, ease of use, advanced and diverse capabilities, and durability. The application may require the integration of specific sensors, connectors, communications ports, and more, compliance with a number of industry or government mandated standards, and perhaps long term availability. A successfully designed mobile device is a simultaneous solution of a broad array of competing requirements along with engineering and business constraints.

Mobile operating systems are a relatively new development in the computing world for the mobile devices. Mostly they are based on some existing older OS, typically written for stationary computers and laptops. For example, Android is based on the Linux kernel, iOS on the Mac OS X kernel, and Windows Phone on the Windows NT kernel. The mobile OS borrows many features from the desktop world; however, it needs to meet the demands of the mobile environment for communication needs, positioning of the device, power challenges caused by mobile operation with rechargeable batteries, and smaller size for the convenience and comfort of users. The OS is responsible for overseeing and managing the hardware and software components on the smartphone and it is responsible for the system-level power management.

In the following units apart from the case studies pertaining to desktops, ANDROID and iOS mobile OS will be discussed.

3.10 SOLUTIONS/ANSWERS

Check Your Progress 1

1) **Sailfsh OS** is the logical evolution of MeeGo, an operating system developed by Nokia, based on Linux and in order to attack the market of smart phones and tablets. When Meego was dropped, some of their workers left to form Nokia Jolla, one covered by the Finnish company, which has renamed the system as Sailfsh OS. The operating system has its own programming language (Qt / QML), although accepted through their respective sdks developing apps in HTML5 (Cordoba Qt) and Java (Android apps).

   The architecture is similar to what we have seen in most mobile operating systems: Linux kernel, a layer of components (based on Mer), with the interface middleware and applications. Sailfish is a modern and secure operating system combining the Linux Kernel for a particular platform use. It is a general purpose linux based operating system.

   It is built on the heritage of MeeGo. Sailfish is developed by Jolla in permanent cooperation with the Sailfish community. The Sailfish community makes development requests and decides development priorities by voting. The Mer project receives contribution from Jolla community and Mer contributes middleware for jolla thereby assuring compatibility of both projects. It's like a classic Linux distribution with the great addition of the Jolla crafted Sailfish UI. There's no buttons on Sailfish devices, everything is handled with simple gestures, supported by an appealing design style. It is really a very interesting platform for both users and developer. Sailfish used in theJjolla smart phone and other licensees. Sailfish OS can run Android applications through a proprietary compatibility layer.

2) **(a) HarmonyOS**

   Huawei unrevealed HarmonyOS 2.0 in 2020. Huawei has planned to relase it as an open-source platform worldwide. It will open up and share its core capabilities in areas like connectivity, cameras and AI. It will work closely with ecosystem partners to deliver apps and services that provide consumers good experience.

   HarmonyOS is completely different from Android and iOS. It is a microkernel-based, distributed OS that delivers a smooth experience across all scenarios. It has a trustworthy and secure architecture, and it supports seamless collaboration across devices. Following are some of the features of HarmonyOS:

   - *Seamless experience*
   - *Smooth performance*
   - *More Secure*
   - *All-in-one*

(b) KaiOS

KaiOS is a web-based mobile operating system that enables a new category of smart feature phones. It is forked from B2G (Boot to Gecko), a successor of the discontinued Firefox OS. KaiOS brings support of 4G/LTE, GPS, and Wi-Fi, as well as HTML5-based apps and longer battery life, to non-touch devices. It has an optimized user interface for smart feature phones, needs little memory, and consumes less energy than other operating systems. It also comes with the KaiStore, which enables users to download applications in categories like social media, games, navigation, and streaming entertainment.

KaiOS apps are based on web technologies like HTML, CSS and Javascript and are run by Gecko runtime. If you have written a web page you already know the basics.

Rendering is done by parsing HTML/CSS and painted using graphic APIs. JavaScript are executed by the JS engine SpiderMonkey and connected to C++ components by XPConnect and WebIDL bindings. Applications and the core process communicate only through IPC protocols defined by IPDL.

Additionally a set of core components including file and memory management, threads and data structures etc. are provided through the XPCOM component object model.

KaiOS Product is decoupled into 2 components: Core and App Profile

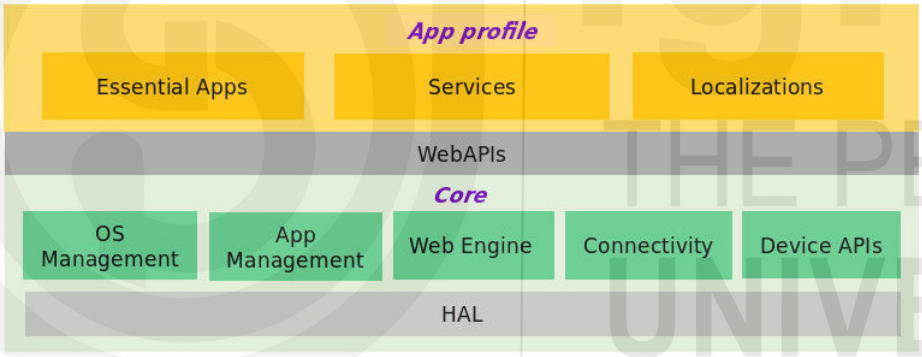

Fig 1: KaiOS Architecture

Core: The underlying platform that consists of the Web application runtime, i.e. Gecko, plus the hardware adaptation layer and a few other supporting module

App Profile: A collection of built-in web apps, which represents the user interfacing functions of KaiOS for a given device form factor.

3.11 FURTHER READINGS

1) Milan Milenkovic, *Operating Systems, Concepts and Design*, TMGH, 2000.

2) D.M. Dhamdhere, *Operating Systems, A concept-based approach*, TMGH, 2002.

3) Andrew S Tanenbaum, Herbert Bos, Modern Operating Systems, 4th Edition, Pearson , 2015.

4) Naresh Chauhan, Principles of Operating Systems, Oxford Press, 2014.

ignou
THE PEOPLE'S
UNIVERSITY