Lecture 10: Concurrent Processing

Concurrent processing is very important in operating system design, particularly for multiprocessor systems, where several CPUs are in operation. In this lecture, we will examine some important concepts, principles and concerns relating to the management of concurrent processes. The lecture proceeds under the following subheadings:

- Introduction
- Hardware Classifications
- MIMD Classifications
- Process Synchronization
- Types of Distributed Operating Systems
- Clusters
- Summary and Concluding Remarks

10.1 Introduction

Parallel processing (also called or concurrent processing or multiprocessing) is the situation in which two or more processors operate in unison. In this situation, a special component of the operating system called the *processor manager* has the daunting task of managing the activities of each processor in the computer system.

Two scenarios for parallel processing are possible:
- One job, several processors
- Several jobs, several processors

The rationale for parallel processing lies in two main advantages:
- Increased computing power
- Enhanced performance and throughput

By splitting a task into concurrent processes that run on different processors, we can obtain levels of performance not physically possible using single processor systems.

Since the 1980s, multiprocessor systems have been developed for high-end computers of IBM mainframes, minicomputers and supercomputers, as well as VAX computers. For instance, early versions of the IBM i systems facilitated a standard of four (4) processors (controllers) with the possibility of additional ones. Today, multiprocessor systems are available at all levels of computers — from microcomputers to supercomputers.

Two main challenges to multiprocessor systems used to be:
- How to connect the processors
- How to synchronize the parallel operation

10.2 Hardware Classifications

Using Flynn's classifications [Flynn, 1997] and Tanenbaum's additional clarifications [Tanenbaum, 1995], we can identify various classifications of multiple CPU computer systems, as follows:

Single Instruction Stream, Single Data Stream (SISD): All traditional single-processor computers fall in this category.

Single Instruction Stream, Multiple Data Stream (SIMD): This class refers to array processors where a single instruction unit fetches an instruction and then commands many data units to carry it out in parallel. Distributed array processors fall in this category.

Multiple Instruction Stream, Single Data Stream (MISD): No known computer system fits this model, due to its impracticality.

Multiple Instruction Stream, Multiple Data Stream (MIMD): This describes a wide group of parallel and distributed computer systems which can be broken down into sub-categories:
- **Tightly Coupled Multiprocessors:** These systems have multiple processors, but share most or all of the resources such as primary storage, secondary storage and operating system. They may be further divided into *bus systems* (typically up to 64 CPUs recommended for reliable performance) and *switched systems* (for more than 64 CPUs).
- **Loosely Coupled Multi-computers:** This describes a typical LAN, MAN, or WAN. Several topologies apply, but they can generally be broken down into bus systems and switched systems.

10.3 MIMD Configurations

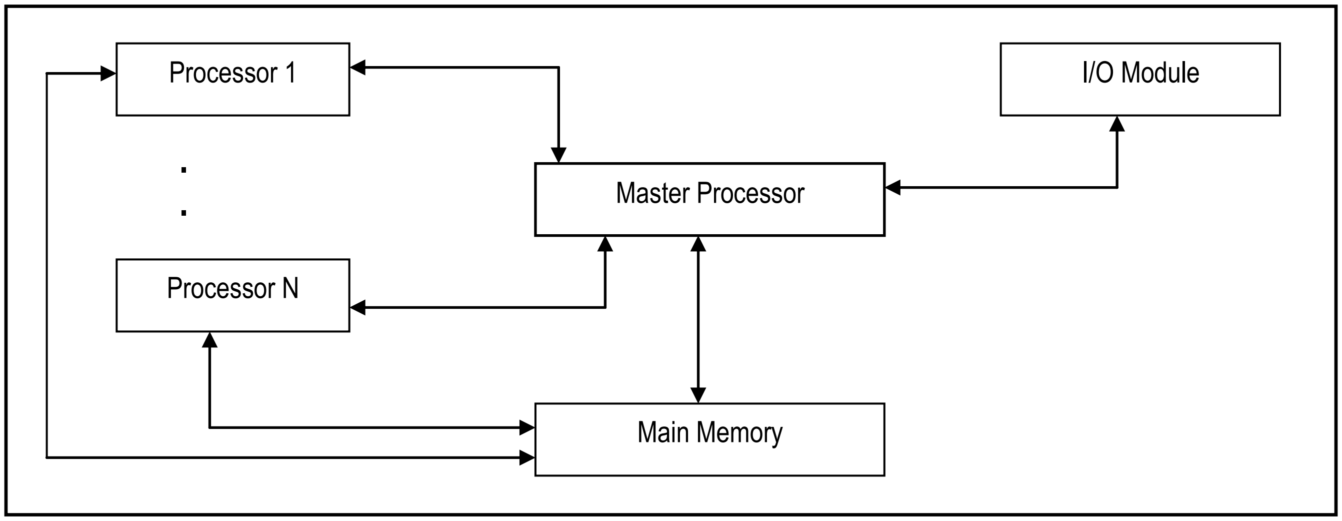
MIMD configurations for multi-computers are well know and are typically covered in a course in computer networks or electronic communication systems. Configurations for multiprocessors are often neglected in such courses; we will therefore briefly look at two such configurations here, along with a third more familiar one. The three typical configurations to be considered are:
- Master-slave configuration
- Loosely coupled configuration
- Symmetric configuration

10.3.1 Master-Slave Configuration

In this configuration, there is a single processor with additional "slave" processors. The "slave processors are managed by the "master" processor, as illustrated in figure 10.1.

The master processor manages issues such as:
- File management
- I/O device management
- Memory management
- Work schedule for the slave processors

Figure 10.1 Typical Master-Slave Configuration



This configuration represents the more traditional implementation of MIMD classification. The system maybe tightly coupled (involving the sharing of memory and/or operating system) or loosely coupled.

The main advantage of the master-slave configuration are:
- Simplicity of design
- Ease of maintainability and control

The main disadvantages are:
- Total reliance on the master processor compromises the reliability of the system
- A slave processor could be idle while waiting to be scheduled by the master processor
- Slave processors must interrupt the master processor each time they need intervention such as I/O requests. This can create bottlenecks at the master processor, if there are several slave processors and many interrupts

10.3.2 Loosely Coupled Configuration

In a loosely coupled configuration, there are several complete computer systems, each with its own CPU, memory, I/O modules and operating systems (figure 10.2 illustrates).

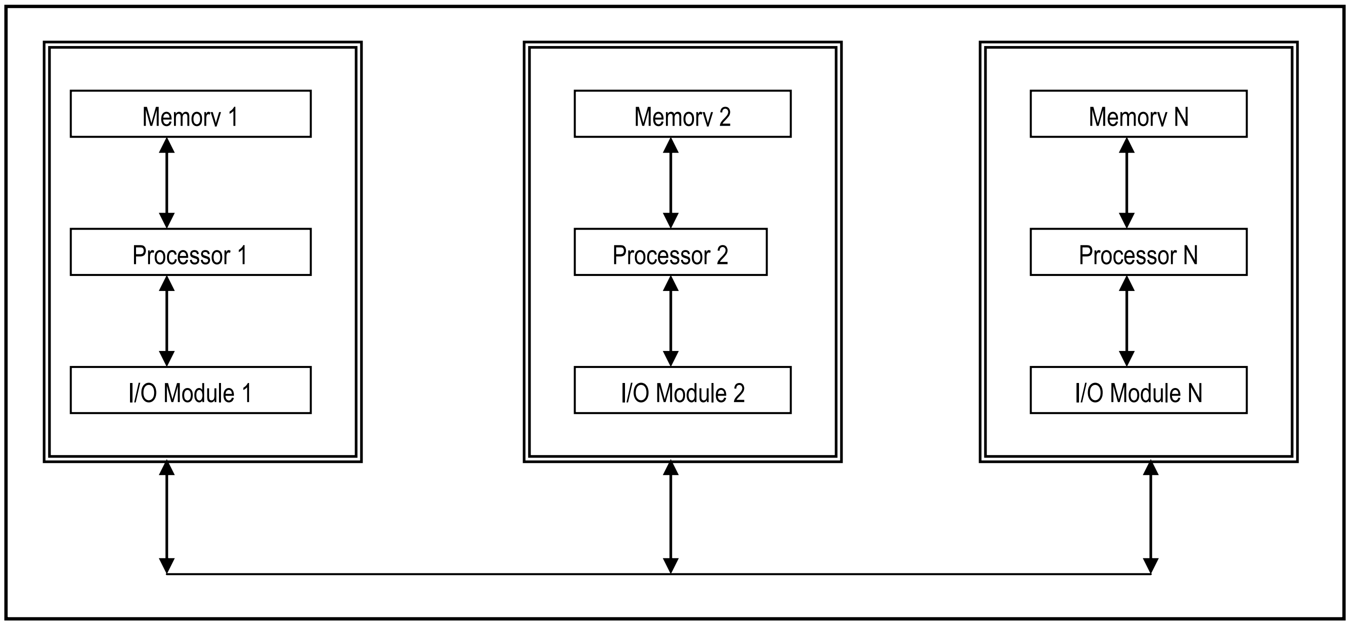
The processors manage their own resources, but each processor can communicate and cooperate with all others in the network. A loosely coupled system is popularly referred to as a distributed communication system.

When a job is created, it is assigned to a specific processor until completed. However, that host processor may solicit the input of other processors, in running the job.

Each processor must maintain a global table of all jobs and their respective allocations in the system.

The processors will share access medium(s) based on the (network) topology implemented (fully connected, partially connected, ring, star, bus or tree — review your notes on *electronic communication systems*).

Figure 10.2 Loosely Coupled Multiprocessor System (Bus Topology)



This configuration is a case of Flynn's MIMD classification. Most LANs, MANs and WANs fall in this category.

The main advantages of this configuration are:
- Robustness and reliability: It is not easy to partition the system or bring it to a halt
- Efficient use of all the resources (processors) of the system
- Lower level of inter-process interrupts, hence improved performance

10.3.2 Loosely Coupled Configuration (continued)

The main disadvantages are:

- Increased complexity: Management of concurrent processes becomes more challenging.
- Redundancy: In the interest of reliability, it is often necessary to introduce certain redundancies in the system, for instance, replication of data, as well as global tables for each processor.
- In a distributed database environment, redundancy introduces a third problem — that of data integrity (see [Foster, 2016]).

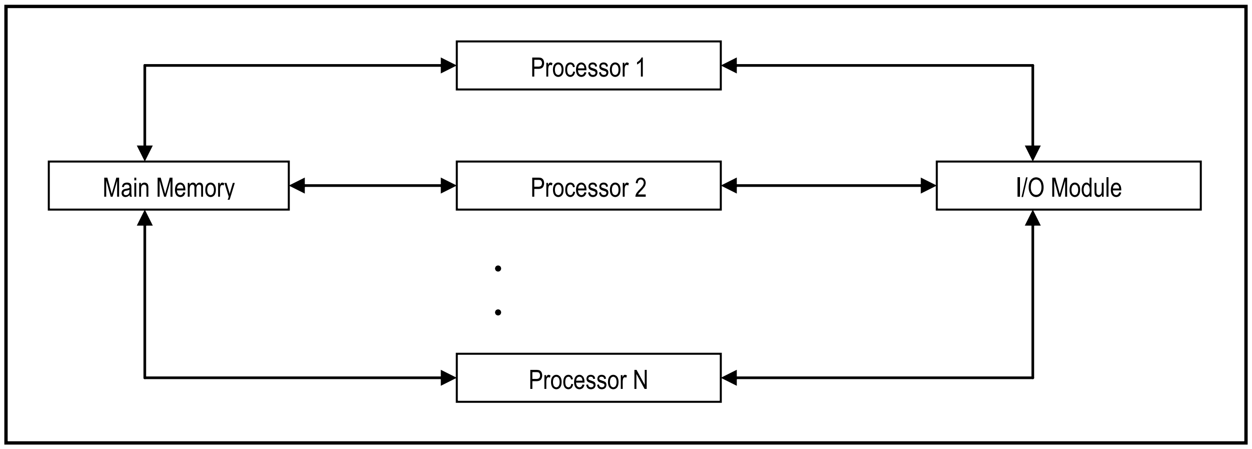
10.3.3 Symmetric Configuration

In the symmetric configuration, memory as well as the I/O modules is shared by multiple processors, but not in a master-slave relationship (figure 10.3 illustrates).

Processor scheduling is decentralized. A single copy of the operating system and a global table, listing the status of each process, are stored in a common area of memory. Each processor has the same scheduling algorithm to determine process selection.

The processors are typically of the same type (and run the same operating system).

Figure 10.3 Symmetric Multiprocessing With Homogeneous Processors



When a process is interrupted, its processor simply updates the corresponding entry in the process table, then finds another process to run. Thus, the processors are kept constantly busy.

10.3.3 Symmetric Configuration (continued)

A process may therefore be serviced by several processors before its completion. This improves system performance, but also increases the capacity for processor conflicts.

This configuration is another case of Flynn's MIMD classification. It is also referred to as a *tightly coupled multiprocessing* system, where most or all of the primary and (optionally) secondary storage facilities are shared.

The significant advantages of this configuration are:

- Robustness and reliability: It is not easy to partition the system or bring it to a halt
- Efficient use of all the resources (processors) of the system
- Good balancing of processor loads
- Graceful degradation in the event of failure of any processor
- Avoidance of unnecessary redundancies typical of the loosely coupled configuration

The significant disadvantages are:

- Synchronization of the processors is a major challenge
- The processors must be homogeneous

10.4 Process Synchronization

The operating system must be able to synchronize parallel activities with a high degree of precision. In so doing, it must be able to make a resource (about which deadlock is possible) unavailable to other processes, while being used by one of them. Such resources include I/O devices (not printers), memory location (such as representing records in a file), a common system bus (such as in an Ethernet network). Upon release of this resource, a waiting process may access it.

Discuss: As an example, discuss what is likely to happen when a process reads a database record for update, while there are other processes attempting to read the same record for update as well.

The **lock-and-key** arrangement is a commonly used synchronism strategy: *Critical regions* of memory/processes are assigned keys. Before a process allocates a critical region, it must first:

a) see if the key is available;
b) obtain the key.

Once it has the key, all other processes are "locked out" until the key is released. Both of the above actions must be performed in a single machine cycle, to avoid deadlock. Three common locking mechanisms are *Test-and-Set; Wait-and-Signal; Semaphores.*

10.4.1 Test-And-Set

The *test-and-set* operation is an indivisible machine instruction (known as "TS"), introduced by IBM for its system 360/370 computers. In a single machine cycle, it tests the availability of the key; if the key is available, the critical region is allocated to the process and the key is set to "unavailable". The key is implemented as a single bit (0 for available; 1 for unavailable).

The main advantage of the TS operation is its simplicity. The main disadvantages are:

- Process starvation could occur, since processes gain access to the critical region in an arbitrary manner.
- Waiting processes remain unproductive resource-consuming loops — a phenomenon known as *"busy waiting".*

10.4.2 Wait-And-Signal

Wait-And-Signal is an enhancement to the test-and-set strategy. Two new operations are introduced as part of the process scheduler's set of operations: WAIT and SIGNAL.

WAIT is activated when the process encounters a busy key. This operation sets the process's PCB to a blocked state and links it to a queue of processes waiting to enter a critical region. The process is now subject to the process scheduler (see lecture 4).

SIGNAL is activated when a process releases a critical region and the corresponding key is set to "available". The operating system checks the queue of processes waiting to enter the critical region and selects one of them (based on the queuing discipline), setting it to a READY state. The process scheduler will next choose this process for running.

The effect of WAIT and SIGNAL operations is to eliminate the "*busy waiting"* condition on processes and return control to the operating system.

10.4.3 Semaphores

A semaphore is a non-negative integer variable, used as a flag that controls access of a critical region. The idea of semaphores was introduced by Dijkstra in 1965.

Dijkstra defined two operations to manipulate the semaphore: The P-operation (from the Dutch word "proberan", which means, to test) allows a test; the V-operation (from the Dutch word "verhogen" which means, to increment) increments the semaphore. The semaphore is subject to three operations:

- It may be initialized to a nonnegative integer value
- It may be decremented by the **wait** operation.
- It may be incremented by the **signal** operation.

10.4.3 Semaphores (continued)

If **s** is a semaphore variable, then the V-operation [increment] on **s** is given by
V(s): s:= s+1;
This is the **signal** operation. This necessitates a fetch, an increment, and a *store* sequence. Like the TS, this **signal** operation is indivisible — it must be performed in a single machine cycle to avoid deadlock.

The P-operation [test] on *s* is to test the value of *s*, and if it is not zero, to decrement it by 1. This is the **wait** operation. Thus
P(s): If (s<= k) then s := s-1; wait;
This necessitates a test, fetch, decrement, and store sequence. Again, the sequence must be indivisible in a machine cycle. Here, *k* is used to denote the initial semaphore value. Many texts do not provide this stipulation. However, it is presented here to add clarity to the discussion.

A process can only access a critical region if its semaphore is non-zero (i.e. s>0). When accessed, the V-operation [increment] takes effect for that critical region. When the process concludes use of a critical region, it issues a V-operation on the semaphore, thus s := s+1.

Semaphores can be used in the following ways:

- **Counting Semaphore:** A *counting semaphore* is a semaphore that is allowed to range over an unrestricted domain. In this scenario, if there are *k* instances of a resource, its semaphore s is initialized to *k*. If multiple processes need an instance of the resource, they each can, provided that s >= 0. Each process desiring to use the resource issues a wait, thus decrementing s. Each process increments s after usage. When s = 0, no resource is available.
- **Binary Semaphore:** The semaphore value varies between 0 and 1. If a process accesses the critical region and its semaphore is 1, it is locked to that process, and the semaphore is decremented to 0; other attempting processes will further decrement the semaphore. When the process holding the critical region is finished, it increments the semaphore. The semaphore acts as a mutual exclusion, — no two processes can access a critical region at the same time; hence the use of the term *mutex* to describe the semaphore variable.

Mutual exclusion is automatic for sequential processes, but for parallel processes, it must be explicitly stated and maintained, hence semaphores.

10.5 Types of Distributed Operating Systems

There are two broad categories of network-oriented operating systems:

- Network Operating Systems
- Distributed Operating Systems

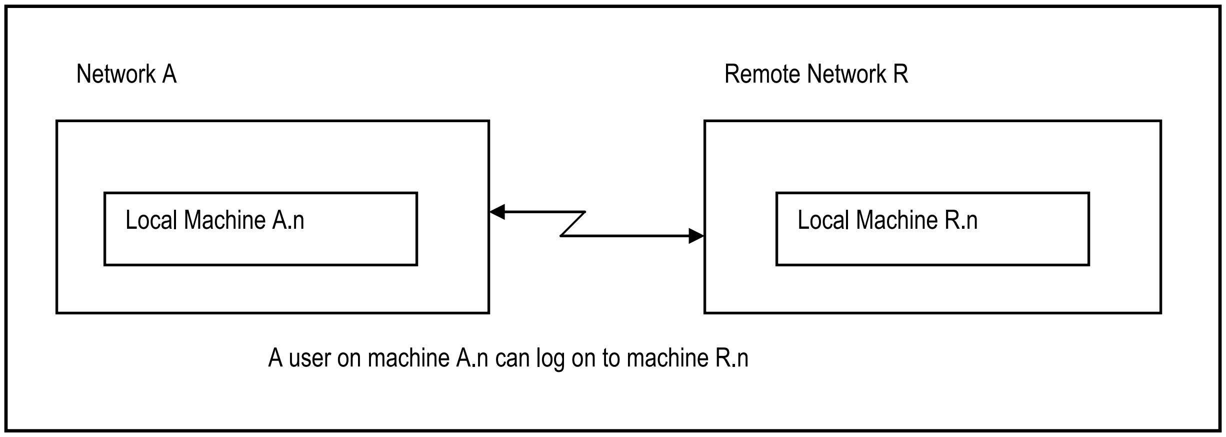
10.5.1 Network Operating Systems

A network operating system (NOS) is an OS that provides the environment in which users can access remote resources by either logging into a remote machine, or transporting data from a remote machine to their own machines.

Remote Access

Remote access is typically facilitated by telnet — a protocol which allows a user to log onto a remote machine belonging to another network. The telnet protocol creates a transparent bi-directional link such that instructions entered by the user are transmitted to the remote machine, and all output from the remote machine are sent back to the user. In order to telnet to a remote machine, the user must first have an account on that machine. Figure 10.4 provides a graphic illustration of the service that telnet provides. One excellent modern implementation of this is Microsoft's Terminal Service Client — a service that allows you to remotely login to a Microsoft server over a private or public network.

Figure 10.4: Illustrating Telnet



Remote File Transfer

Remote file transfer is typically facilitated by the **file transfer protocol** (ftp). This program allows the user of a computer in network A (see figure 10.4) to log onto a computer in network R and copy a file from the remote computer to the user's local computer or vice versa. The ftp protocol supports three commands only: **get, put** and **ls**.

The command

```
ftp <Remote Domain Name>
```

allows remote log on; it initiates the protocol.

The command

```
get <Remote File Name>
```

allows the actual file transfer; it copies remote files to local machine.

The command

```
put <Remote File Name>
```

Also allows the actual file transfer; it copies a local file to the remote machine. The command **ls** lists files in current directory of the remote machine.

An FTP connection is not transparent in the sense that a telnet connection is. Additionally, most contemporary FTP software solutions provide a graphical user interface (GUI). A user who does not have an account for the remote machine can still use ftp, via the anonymous account (which requires no password).

Note: The remote user can only access directories and files to which he/she has been authorized. In the case of anonymous, only objects placed in the directory tree of user anonymous (with appropriate access rights to public) are accessible.

10.5.2 Distributed Operating Systems

In a distributed operating system, the users access remote resources in the same manner as they do local resources. The operating system manages distributed processing in one of three ways:

- Data migration
- Computation migration
- Process migration

Data Migration

When a user at site A desires to access an object (e.g. a data file) at site B, the file is replicated at site A. Two approaches are possible:

a. Replicate the entire file. If changes were made at site A, then the entire file is retransmitted to site B, subsequent to its use at site A. Obviously, this could be very expensive and inefficient. The Andrews filing system uses this approach.
b. Replicate only the fragment(s) of the file that are required at site A (similar to demand paging). If changes are made, retransmit the changed fragments after usage at site A. The Sun Microsystems' Network Filing System (NFS) protocol uses this approach.

Computation Migration

In some circumstances, it may be more prudent to transfer computation, instead of data, across a network, hence the term computational migration. Generally, if the time required to transfer data is greater than the time to execute a remote command, then the latter should be used. This situation typically occurs where large files are involved.

Computation migration may be effected in any of the following ways:

a. Process P at site A issues a *remote procedure call* (RPC) to execute a routine on a remote system at site B. The procedure at site B executes and returns the result to site A.
b. Process P at site A sends a message to site B. The OS at site B creates a new process, Q, to carry out the designated task. When process Q completes, the OS at site B sends the result back to site A (process P) via another message.

The second approach is ideal for supporting multiple concurrent processes (on different systems) for a given job. Both Linux and Unix are known for supporting RPCs.

10.5.2 Distributed Operating Systems (continued)

Process Migration

Process migration is a logical extension of computation migration: A process may be dissected into multiple sub-processes (threads) which can be executed (possibly concurrently) at different sites. This approach may be employed for several reasons:

a. **Load balancing:** Processes may be distributed across a network to share (balance) the work load among different processors.
b. **Computational speed up:** If a single process can be split up into multiple (preferably) concurrent sub-processes, the total process turn around will be significantly reduced.
c. **Hardware preference:** A process may have characteristics that qualify it to be more suited for a particular processor, as opposed to some other processor in a network. Example: matrix inversion is more efficient on an array processor then a microprocessor.
d. **Software preference:** A process may require software that is available at specific site(s), and it is less expensive and more convenient to move the process, rather than the software.
e. **Data access:** If the process must access huge files at specific site(s), it may be more prudent to migrate the process rather than the data files.

There are two possible approaches to process migration:

- **Transparent Process Migration:** This method is usually employed to achieve load balancing among homogeneous systems. No user (programmer) intervention is required by the cooperating systems.
- **User-controlled Migration:** The user (programmer) explicitly specifies how the process migration should occur. This approach is applicable to situations where there are hardware and/or software preferences to be met.

The World Wide Web (WWW) provides excellent examples of process migration, for example:

- A web client triggering a database operation on a web server.
- Java applets being sent from server sites to client sites where they are executed.
- Multi-agent systems which operate on heterogeneous OS platforms.

10.6 Concurrent Programming

Most traditional programming languages are by nature, allowing instructions to be executed sequentially. Examples include COBOL, Pascal, C, Fortran, etc. More sophisticated high level languages allow for concurrent programming. Examples are object oriented languages such as Java, Ada, C++, etc.

In early systems, programmers had to specify *explicit parallelism*. This was tedious and error prone; it also made program maintenance difficult. In modern systems, compilers are designed to optimize simple sequential code into parallel code, where possible. This is referred to as *implicit parallelism*.

Explicit parallelism is still important, for instance in code such as writing communication protocols, operating system threads and certain applications for distributed systems

10.7 Clusters

Like parallel systems, clustered systems incorporate multiple CPU's to accomplish computational processing. The definition of the term "clustered systems" is somewhat nebulous, but it is generally agreed that in such a system two or more individual systems are coupled together, sharing storage, and are closely linked via a LAN, MAN or WAN.

In a clustered system, each node can monitor one or more of the other nodes over the network. If a monitored machine fails the monitoring machine can take ownership of its storage and restart application(s) that were running on the failed machine.

In asymmetric clustering, one machine operates in *hot standby mode*, while the other machine runs the application. In the event that the server fails, the machine in hot standby mode trips in and takes over as the server.

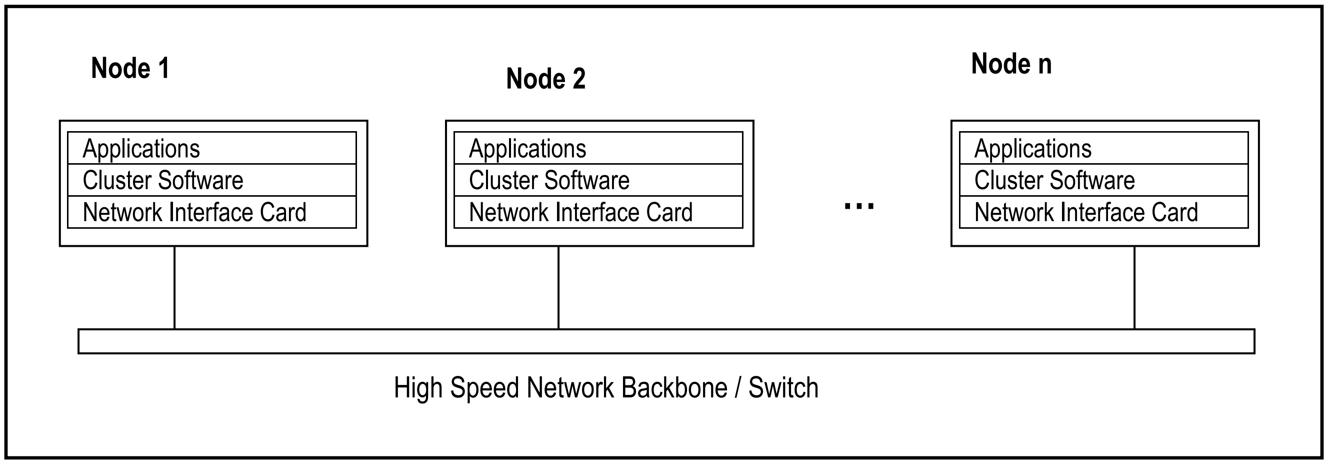
In symmetric clustering, two or more nodes run the applications, and they monitor each other, adding more reliability and efficiency to the system.

10.7.1 Cluster Architecture

The basic architecture of a cluster system is illustrated in figure 10.5. Note:

- The computers are connected via a high speed LAN or switch.
- A middleware layer of software is installed on each participating machine (node). This software provides a unified system image to the user, referred to as a *single system image*. The middleware software is also responsible for load balancing and responding to failure in any node.
- Applications running on the respective node may be sequential applications as well as parallel applications.

Figure 10.5: Cluster Architecture



10.7.1 Cluster Architecture (continued)

The following are some desirable features of the cluster software:

- **Single Entry point:** A user logs on to the cluster rather than a single node of the system.
- **Single File Hierarchy:** The user sees a single hierarchy of directories and objects.
- **Single Control Point:** A default node is used for management of the cluster.
- **Single Virtual Networking:** A node can access any other node or point in the cluster, irrespective of the multiple computers and networks that may be involved in the cluster.
- **Single Memory Space:** Memory of the component nodes is aggregated producing a single virtual memory space. Programs can therefore share variables across different nodes.
- **Single Job Management:** A user can submit a job without specifying which machine (host) the job will execute on.
- **Single User Interface:** A common (graphical) user interface supports all users irrespective of their host workstations.
- **Single Process Space:** A uniform process identification scheme is in place, such that a process on any given node can create, or communicate with any other process on another (remote) node.
- **Process Migration:** A process can migrate from one node to another in the cluster (typically to enable load balancing)
- **Check-pointing:** Process states are periodically saved to facilitate fallback recovering after any possible failure.

Three prominent cluster implementations are:

- Windows 2000 Cluster server
- Sun Cluster
- Beowulf Cluster

A brief summary of each one follows:

10.7.2 Windows 2000 Cluster Server

The Windows 2000 Cluster Server is a *shared-nothing cluster* in which the resources of the cluster are owned by a single node at a time.

Each node in a windows 2000 cluster has a number of important components:

- Node Manager
- Database Manager
- Event Processor
- Communications Manager
- Global Update Manager
- Resource Manager

10.7.2 Windows 2000 Cluster Server (continued)

The **node manager** is responsible for maintaining the node's membership in the cluster. Whenever a node looses contact (i.e. does not receive messages) from another node or set of nodes, the node manager broadcasts a message to the entire network. All members receiving this broadcast message must respond by sending messages to verify their respective current cluster membership. If a node manager fails to respond, it is removed from the cluster.

The **database manager** of each node maintains a cluster configuration database so that communication with other nodes can be facilitated.

The **global update manager** works closely with the node manager and the database manager to unsure that all global updates take place, thus enabling successful cluster operation.

The **event processor** of each node handles issues such as common operations and cluster service initialization.

The node's **communications manager** handles exchanges with other nodes of the cluster.

The **resource manager** makes all decisions about a node's participation in the cluster, including issues such as startup, reset, and resumption after a node failure.

10.7.3 Sun Cluster

The Sun Cluster is a distributed OS, built as a set of extensions of Solaris Unix. It provides a single-system image quite effectively.

Each node in a sun cluster has the following (software) components:
- Object and Communication Support Module
- Process Management Module
- Networking Management Module
- Global Distributed File System

The **Object and Communication Support Module** employs an object-oriented approach to the management of cluster resources. The implementation is CORBA (Common Object Request Broker Architecture) compliant; interface specifications are done in CORBA's interface definition language (IDL).

The **Process Management Module** manages process operations in such a manner as to ensure that a process's location is transparent to the end user. A global view of all processes is maintained, and each process has a unique identification. Thus, each node in the cluster can access all processes in the cluster. Process migration is also facilitated.

10.7.3 Sun Cluster (continued)

The **Networking Management Module** takes care of node-to-node communication within the cluster, as well as communication of the cluster with non-cluster machines in the "outside world". Within the cluster, a packet filter (on each node) routes packets to the proper node. Externally, the cluster appears as a single server with one IP address. Incoming client requests are load-balanced among available nodes in the cluster.

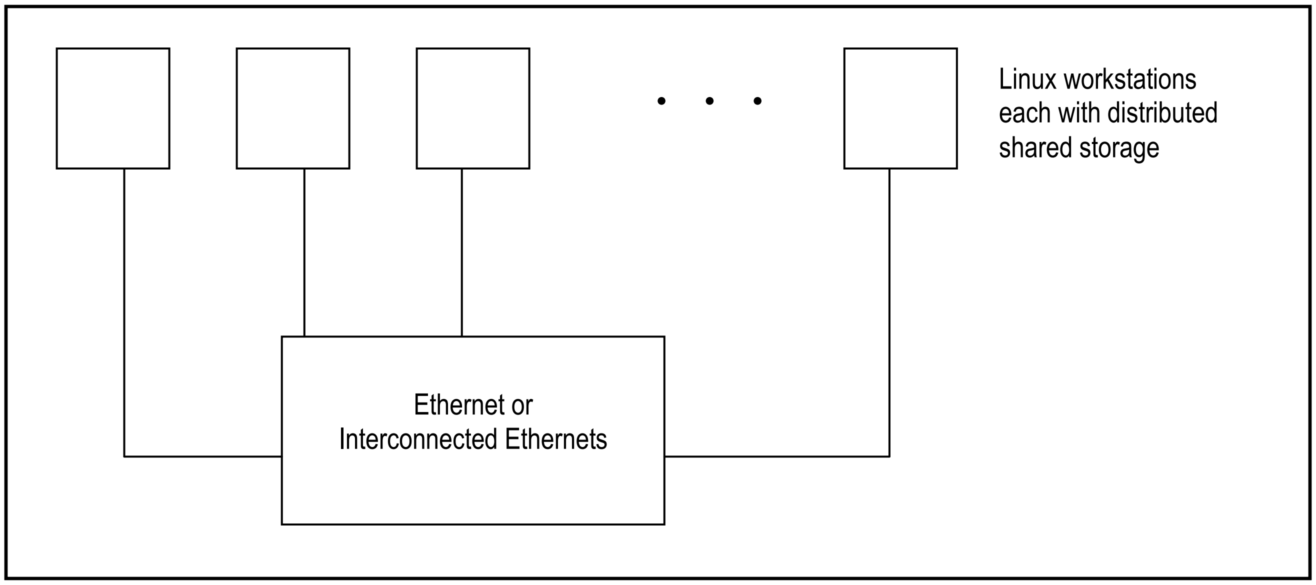
The **Global Distributed File System** provides a uniform interface to files distributed over the cluster. A process can therefore open a file, located anywhere in the cluster. Furthermore, processes on all nodes use the same (virtual) pathname to files in the cluster.

10.7.4 Beowulf Cluster

The Beowulf project was initiated in 1994 under the sponsorship of the NASA High Performance and Computing Communications (HPCC) project, with the objective of investigating the potential of clustered PC's for enhanced performance (beyond the physical capabilities of a single workstation) at minimum cost. Today, the Beowulf approach remains one of the most important cluster technologies available.

The Beowulf software was written for the Linux OS platform, but it has been successfully implemented on other platforms. The basic configuration is illustrated in figure 10.6

Figure 10.6: Basic Beowulf Configuration


Linux workstations each with distributed shared storage
Ethernet or Interconnected Ethernets

In a Beowulf cluster, each node runs the Linux OS. Secondary storage at each node may be made available for distributed access. The Ethernet support may be in the form of a single Ethernet switch or an interconnected set of switches. Standard transmission rates (10 Mbps, 100 Mbps, 1Gbps) are used.

10.7.4 Beowulf Cluster (continued)

The significant features of a Beowulf cluster are as follows:

- Mass market commodity components (and no custom components)
- Dedicated processors
- A dedicated network (LAN,MAN, WAN, or interlinked combination)
- Easy replication from multiple vendors
- Freely available software base
- Scalable I/O
- Use of readily (freely) available distribution computing tools with minimal changes

The Beowulf software, which is implemented as an add-on to the Linux OS, is readily available at website www.beowulf.org. Additionally, other organizations also offer free Beowulf tools and utilities.

Each node in a Beowulf cluster runs its own copy of the Linux kernel, and can function as an autonomous, independent Linux system, or as part of the cluster.

Important Beowulf software components include the following:

- **Beowulf Distributed Process Space (BPROC):** This component is responsible for providing the single-system image.
- **Beowulf Ethernet Channel Bonding:** This component is responsible for actual communication among nodes via the Ethernet channel
- **Pvmsync:** This component is a programming environment that provides synchronization and shared data for processes in the cluster.

10.7.5 Spinoffs from Cluster Technology

Cluster technology has inspired research leading to various contemporary developments in computer science. Two prominent spinoffs from cluster technology are *grid computing* and *cloud computing*.

Grid computing is the term used to describe a federation of computer resources from multiple domains to for a coherent and integrated super system. The *grid* qualifies as a distributed system. Grid computing differs from conventional high performance computing systems such as cluster computing in that grids tend to be more loosely coupled, heterogeneous, and geographically dispersed. Grid computing can therefore be construed as an upgrade of cluster computing.

Cloud computing is a metaphor used to describe the delivery of various computing requirements as a service to a mixed community of end-recipients. A *cloud* represents a complex abstraction that integrates various technologies and resources into a virtual information infrastructure. Cloud computing provides (typically centralized) services with a user's data, software, and computation on an application programming interface (API) over a network (typically the internet). End users access cloud services through a web browser or a light weight desktop or mobile application.

10.8 Summary and Concluding Remarks

Here is a summary of what has been covered in this lecture:

- Parallel processing is the situation in which two or more processors operate in unison. The main benefits are improved computing power, and enhanced performance.
- Possible hardware configurations include single instruction stream, single data stream (SISD), single instruction stream, multiple data stream (SIMD), multiple instruction stream, single data stream (MISD) and multiple instruction stream, multiple data stream (MIMD). MISD is not practical, and MIMD is the most widely used.
- MIMD configurations include master-slave configuration, loosely coupled configuration, and symmetric configuration.
- Common strategies for process synchronization include test-and-set, wait-and-signal, and semaphores.
- Network operating systems traditionally support remote access and remote file transfers. Distributed operating systems traditionally support data migration, computation migration, or process migration. In modern operating systems, these distinctions have become somewhat blurred.
- Like parallel systems, clustered systems incorporate multiple CPU's to accomplish computational processing. It is generally agreed that in a clustered system, two or more individual systems are coupled together, sharing storage, and are closely linked via a LAN, MAN or WAN. Prominent clusters are the Windows cluster, the Sun cluster, and the Beowulf cluster.

The field of distributed computing remains one of the most fascinating areas of computer science. For this and other reasons, it continues to be a prominent area of graduate and post graduate research. Much of this epic age of information explosion is owed to distributed systems.

Below is a checklist of areas that have attracted researchers over the years, some of which may very well appeal to you.

- Distributed database management
- Distributed operating systems development
- Distributed communication protocols
- Security mechanisms for distributed systems
- Encryption algorithms for distributed systems
- Multi-agent applications for distributed systems

10.9 References and/or Recommended Readings

1. Flynn, Michael J. 1972, September. "Some Computer Organizations and Their Effectiveness", *IEEE Transactions on Computers*, vol. C-21. pp. 948-960.

2. Flynn, Ida M. and Ann McIver McHoes. 1997. *Understanding Operating Systems*, 2nd ed. Boston, Massachusetts: PWS Publishing, 1997. pp. 125-137.

3. Foster, Elvis C & Shripad V. Godbole. 2016. *Database Systems: A Pragmatic Approach*, 2nd ed. New York: Apress Publishing See Chapter 17.

4. Nutt, Gary. 2004. *Operating Systems: A Modern Perspective* 3rd ed. Boston: Addison-Wesley. See Chapter 17.

5. Silberschatz, Abraham, Peter B. Galvin, & Greg Gagne. 2012. *Operating Systems Concepts*, 8th Edition Update. New York: John Wiley & Sons. See Chapters 16 – 18.

6. Stallings, William. 2005. *Operating Systems* 5th ed. Upper Saddle River, New Jersey: Prentice Hall. See Chapters 13 & 14.