6.1 Multiprocessing Systems

Multiprocessing is the use of two or more central processing units (CPUs) within a single computer system. The term also refers to the ability of a system to support more than one processor and/or the ability to allocate tasks between them. 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 (multiple cores on one die, multiple dies in one package, multiple packages in one system unit, etc.).

6.1.1 Real and Pseudo-Parallelism

- Traditionally, software has been written for serial computation:
  - To be run on a single computer having a single Central Processing Unit (CPU);
  - A problem is broken into a discrete series of instructions.
  - Instructions are executed one after another.
  - Only one instruction may execute at any moment in time.
- In the simplest sense, parallelism is the simultaneous use of multiple compute resources to solve a computational problem:
  - To be run using multiple CPUs
  - A problem is broken into discrete parts that can be solved concurrently
  - Each part is further broken down to a series of instructions
  - Instructions from each part execute simultaneously on different CPUs
- Real parallelism consists of the parallel modes of physical devices so that each can carry parallel operations to each other. Core parallelism consists of real parallelism. Multiple core processes which are physically different and performs their own operations in parallel.
- Pseudo parallelism consists of the same device carrying the parallel operation. We can logically manage the parallelism for system. Concurrent processing using parallelism is the pseudo parallelism which operates either in time division or using other types of parallel algorithms.

6.1.2 Flynn's Classification

- There are different ways to classify parallel computers. One of the more widely used classifications, in use since 1966, is called Flynn's classification.
- Flynn's classification distinguishes multi-processor computer architectures according to how they can be classified along the two independent dimensions of ***Instruction*** and ***Data***. Each of these dimensions can have only one of two possible states: ***Single*** or ***Multiple***.
- The matrix below defines the 4 possible classifications according to Flynn:

S I S D **Single Instruction, Single Data**	**S I M D** **Single Instruction, Multiple Data**
M I S D **Multiple Instruction, Single Data**	**M I M D** **Multiple Instruction, Multiple Data**

1. Single Instruction, Single Data (SISD):

- A serial (non-parallel) computer
- **Single Instruction:** Only one instruction stream is being acted on by the CPU during any one clock cycle
- **Single Data:** Only one data stream is being used as input during any one clock cycle

- Single instruction is performed on a single set of data in a sequential form.
- Deterministic execution
- This is the oldest and even today, the most common type of computer

Examples: older generation mainframes, minicomputers and workstations; most modern day PCs.

2. Single Instruction, Multiple Data (SIMD):

- A type of parallel computer
- **Single Instruction:** All processing units execute the same instruction at any given clock cycle
- **Multiple Data:** Each processing unit can operate on a different data element
- Best suited for specialized problems characterized by a high degree of regularity, such as graphics/image processing.
- Single Instruction is performed on multiple data. A good example is the 'For' loop statement. Over here instruction is the same but the data stream is different. Examples:
    - Processor Arrays: Connection Machine CM-2, MasPar MP-1 & MP-2, ILLIAC IV
    - Vector Pipelines: IBM 9000, Cray X-MP, Y-MP & C90, Fujitsu VP, NEC SX-2, Hitachi S820, ETA10

Most modern computers, particularly those with graphics processor units (GPUs) employ SIMD instructions and execution units.

3. Multiple Instruction, Single Data (MISD):

- A type of parallel computer
- **Multiple Instruction:** Each processing unit operates on the data independently via separate instruction streams.
- **Single Data:** A single data stream is fed into multiple processing units.
- N numbers of processors are working on different set of instruction on the same set of data.
- Few actual examples of this class of parallel computer have ever existed. One is the experimental Carnegie-Mellon C.mmp computer (1971).
- Some conceivable uses might be:
    - multiple frequency filters operating on a single signal stream

Multiple cryptography algorithms attempting to crack a single coded message.

4. Multiple Instruction, Multiple Data (MIMD):

- A type of parallel computer
- **Multiple Instruction:** Every processor may be executing a different instruction stream
- **Multiple Data:** Every processor may be working with a different data stream
- There is an interaction of N numbers of processors on a same data stream shared by all processors.
- Execution can be synchronous or asynchronous, deterministic or non-deterministic
- Currently, the most common type of parallel computer - most modern supercomputers fall into this category.
- Examples: most current supercomputers, networked parallel computer clusters and "grids", multi-processor SMP computers, multi-core PCs.

Note: many MIMD architectures also include SIMD execution sub-components

6.1.3 Instruction Level, Thread Level and Process Level Parallelism

Instruction Level Parallelism

Instruction-level parallelism (**ILP**) is a measure of how many of the operations in a computer program can be performed simultaneously. Consider the following program:

1. e = a + b
2. f = c + d
3. g = e * f

Operation 3 depends on the results of operations 1 and 2, so it cannot be calculated until both of them are completed. However, operations 1 and 2 do not depend on any other operation, so they can be calculated simultaneously. If we assume that each operation can be completed in one unit of time then these three instructions can be completed in a total of two units of time, giving an ILP of 3/2.

A goal of compiler and processor designers is to identify and take advantage of as much ILP as possible. Ordinary programs are typically written under a sequential execution model where instructions execute one after the other and in the order specified by the programmer. ILP allows the compiler and the processor to overlap the execution of multiple instructions or even to change the order in which instructions are executed.

Micro-architectural techniques that are used to exploit ILP include:

- **Instruction pipelining** where the execution of multiple instructions can be partially overlapped.
- **Superscalar execution**, VLIW, and the closely related Explicitly Parallel Instruction Computing concepts, in which multiple execution units are used to execute multiple instructions in parallel.

Thread Level Parallelism

Thread parallelism (also known as **Task Parallelism**, **function parallelism** and **control parallelism**) is a form of parallelization of computer code across multiple processors in parallel computing environments. Thread parallelism focuses on distributing execution processes (threads) across different parallel computing nodes. It contrasts to data parallelism as another form of parallelism.

It was later recognized that finer-grain parallelism existed with a single program. A single program might have several threads (or functions) that could be executed separately or in parallel. Some of the earliest examples of this technology implemented input/output processing such as direct memory access as a separate thread from the computation thread. A more general approach to this technology was introduced in the 1970s when systems were designed to run multiple computation threads in parallel. This technology is known as multi-threading (MT).

As a simple example, if we are running code on a 2-processor system (CPUs "a" & "b") in a parallel environment and we wish to do tasks "A" and "B" , it is possible to tell CPU "a" to do task "A" and CPU "b" to do task 'B" simultaneously, thereby reducing the run time of the execution.

Thread parallelism emphasizes the distributed (parallelized) nature of the processing (i.e. threads), as opposed to the data (data parallelism). Most real programs fall somewhere on a continuum between Thread parallelism and Data parallelism.

The pseudocode below illustrates task parallelism:

```
program:
...
if CPU="a" then
  do task "A"
else if CPU="b" then
  do task "B"
end if
...
end program
```

The goal of the program is to do some net total task ("A+B").

Code executed by CPU "a":

```
program:
...
do task "A"
...
end program
```

Code executed by CPU "b":

```
program:
...
do task "B"
...
end program
```

This concept can now be generalized to any number of processors.

Data parallelism

Data parallelism is parallelism inherent in program loops, which focuses on distributing the data across different computing nodes to be processed in parallel. "Parallelizing loops often leads to similar (not necessarily identical) operation sequences or functions being performed on elements of a large data structure." Many scientific and engineering applications exhibit data parallelism.

A loop-carried dependency is the dependence of a loop iteration on the output of one or more previous iterations. Loop-carried dependencies prevent the parallelization of loops. For example, consider the following pseudocode that computes the first few Fibonacci numbers:

```
  PREV1 := 0
  PREV2 := 1
  do:
    CUR := PREV1 + PREV2
    PREV1 := PREV2
    PREV2 := CUR
  while (CUR < 10)
```

This loop cannot be parallelized because CUR depends on itself (PREV2) and PREV1, which are computed in each loop iteration. Since each iteration depends on the result of the previous one, they cannot be performed in parallel. As the size of a problem gets bigger, the amount of data-parallelism available usually does as well.

Process Level Parallelism

Process-level parallelism is the use of one or more central processing units (CPUs) within a single computer system. The term also refers to the ability of a system to support more than one processor and/or the ability to allocate tasks between them. There are many variations on this basic theme, and the definition of process level parallelism can vary with context, mostly as a function of how CPUs are defined (multiple cores on one die, multiple dies in one package, multiple packages in one system unit, etc.).

This varies, depending upon who you talk to. In the past, a CPU (Central Processing Unit) was a singular execution component for a computer. Then, multiple CPUs were incorporated into a node. Then, individual CPUs were subdivided into multiple "cores", each being a unique execution unit. CPUs with multiple cores are sometimes called "sockets". The result is a node with multiple CPUs, each containing multiple cores.

- During the past 20+ years, the trends indicated by ever faster networks, distributed systems, and multi-processor computer architectures (even at the desktop level) clearly show that **parallelism is the future of computing**.
- In this same time period, there has been a greater than 1000x increase in supercomputer performance, with no end currently in sight.

6.1.4 Inter-process Communication, Resource Allocation and Deadlock

Inter-process communication

In computing, **Inter-process communication** (**IPC**) is a set of methods for the exchange of data among multiple threads in one or more processes. Processes may be running on one or more computers connected by a network. IPC methods are divided into methods for message passing, synchronization, shared memory, and remote procedure calls (RPC). The method of IPC used may vary based on the bandwidth and latency of communication between the threads, and the type of data being communicated.
There are several reasons for providing an environment that allows process cooperation:

- Information sharing
- Speedup
- Modularity
- Convenience
- Privilege separation

IPC may also be referred to as inter-thread communication and inter-application communication.
The combination of IPC with the address space concept is the foundation for address space independence/isolation.

Resource Allocation

In computing, **resource allocation** is necessary for any application to be run on the system.
When the user opens any program this will be counted as a process, and therefore requires the computer to allocate certain resources for it to be able to run. Such resources could be access to a section of the computer's memory, data in a device interface buffer, one or more files, or the required amount of processing power.

A computer with a single processor can only perform one process at a time, regardless of the amount of programs loaded by the user (or initiated on start-up). Computers using single processors appear to be running multiple programs at once because the processor quickly alternates between programs, processing what is needed in very small amounts of time. This process is known as multitasking or *time slicing*. The time allocation is automatic, however higher or lower priority may be given to certain processes, essentially giving high priority programs more/bigger **slices** of the processor's time.

On a computer with multiple processors different processes can be allocated to different processors so that the computer can truly multitask. Some programs, such as Adobe Photoshop, which can require intense processing power, have been coded so that they are able to run on more than one processor at once, thus running more quickly and efficiently.

Deadlock

A process requests resources; if the resources are not available at that time, the process enters a wait state. Waiting processes may never again change state, because the resources they have requested are held by other waiting processes. This situation is called a deadlock.
Processes need access to resources in reasonable order. Suppose a process holds resource A and requests resource B, at same time another process holds B and requests A; both are blocked and remain in deadlock.

A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause. Usually the event is release of a currently held resource. None of the processes can run, release resources and then be awakened.

6.1.5 Features of Typical Operating System

Operating system is a system s/w. which control and manage the hardware through bios (basic i/o system). Bios is the part of operating system which is built in the system and it run application s/w on a hardware. It is an interface b/w user and system/s is a complex s/w. In a simple word operating system is that which is active on a ram before switch of the system. The operating systems key elements are

- A technical layer of software for driving software components.
- A file system for organizing and accessing files logically.
- Simple command language enabling users to run their own programs and manipulating files.

OS Features

1. **Process Management**

Operating system manages the process on hardware level and user level. To create, block, terminate, request for memory, Forking, releasing of memory etc. in multi tasking operating system the multiple processes can be handle and many processes can be create ,block and terminate ,run etc. it allow multiple process to exist at any time and where only one process can execute at a time. The other process may be performing I/O and waiting. The process manager implements the process abstraction and creating the model to use the CPU. It is the major part of operating system and its major goal is scheduling, process synchronization mechanism and deadlock strategy

2. File Management

Is a computer program and provide interface with file system. it manage the file like creation, deletion, copy, rename etc files typically display in hierarchical and some file manager provide network connectivity like ftp, nfs, smb or webdav.

3. Memory Management

Is a way to control the computer memory on the logic of data structure? It provides the way to program to allocate in main memory at their request. The main purpose of this

manager is; It allocates the process to main memory; minimize the accessing time and process address allocate in a location of primary memory. The feature of memory manager on multi tasking is following.

- Relocation
- Protection
- Sharing
- Logical organization
- Physical organization

4. Device Management

Allow the user to view its capability and control the device through operating system. Which device may be enabling or disable or install or ignore the functionality of the device. In Microsoft windows operating system the control panel applet is the device manager .it also built on web application server model. Device Manager provides three graphical user interfaces (GUIs). Device manager manage the following:

- Device configuration
- Inventory collection
- S/W distribution
- Initial provisioning

5. Resource management

Is a way to create, manage and allocate the resources. Operating system is a responsible to all activities which is done in computer. Resource manager is the major part of operating system. The main concept of operating system is managing and allocating the resources. The resources of the computer are storage devices, communication and I/O devices etc. these all resources manage and allocate or de allocate by resource manager.