**How branch predictor algorithm works?**

It is a digital circuit which tries to predict in which way a branch would go before the proper command is to be executed. Its purpose is to improve the flow in the instruction pipeline. Branch predictors play a critical role in achieving highly effective performance in modern pipelined microprocessors such as x86. It is based on speculative algorithm. Branch predictor predicts what the next line of code would be, and speculative algorithm predicts what the output of that code would be, before it is actually being executed.

**Implementation of Branch Prediction.**

**Static Branch Prediction-** Static prediction is the simplest branch prediction technique because it does not rely on information about the dynamic history of code executing. Instead, it predicts the outcome of a branch based solely on the branch instruction. Evaluates branches in decode stage and have a single cycle instruction fetch.

**Dynamic Branch Prediction-** Uses information about taken or not taken branches gathered at run time to predict the outcome of a branch.

**Random Branch Prediction –** Random Branch Predictor uses algorithm which predicts what the next branch would come to the process cycle to be executed at run time. It has a prediction rate of around 50% **.**

**Next line prediction –** fetches each line of instruction with a pointer to the next line. The next line predictor points to aligned pointers and predicts its outcome so that the execution time is less.

**What is Cache Coherency?**

Cache coherence is the uniformity of shared resource data that ends up stored in multiplelocal caches. When clients in a system maintain caches of a common memory resource, problems may arise with incoherent data, which is particularly the case with CPUs in a multiprocessing system.

When more than 1 cahce are connected with each other, there is a ambiguity of data, and this leads to inconsistency of data. This is called cache coherency.

**Process vs Threads**

A process usually represents an independent execution unit with its own memory area, system resources and scheduling slot.

A thread is typically a "division" within the process which usually share the same memory and operating system resources, and share the time allocated to that process.

Process operations are controlled by PCB which is a kernel data structure. PCB uses the three kinds of functions which are scheduling, dispatching and context save.

For thread,the kernel allocates a stack and a thread control block (TCB) to each thread. Threads are implemented in three different ways:kernel-level threads, user-level threads, hybrid threads. Threads can have three states running, ready and blocked.

A thread can't have individual existence whereas process can exit individually.

A process is heavy weighted, but a thread is light weighted.

**Limitations of Amdahl’s Law**

Limitations of Amdahl's Law

1. Shared resources have to be used serially

2. No task is perfectly parallelizable.

3.There is Load imbalance

4.Task interdependencies must be accounted

5. Coordinating communications among the various processes will require additional code. This adds to the overall execution time.

6. The amount of speedup depends on the size of the problem.

**Loosely Coupled vs Tightly coupled**

**Cache Memory and Levels of Cache Memory**

A Cache is used by the CPU to access data from the main memory in short time. It is a small and very fast temporary storage memory. It is designed to speed up the transfer of data or instructions. CPU Cache is located inside or near to the CPU chip. The data/instructions which are most recently or frequently used by the CPU are stored in CPU. A copy of data/instructions is stored as a cache when the CPU uses them for the first time which retrieved from RAM. The next time when CPU needs the data/instruction, it looks in the cache. If the required data/instruction is found there, then it is retrieved from the cache memory instead of main memory.

**Types/Levels of cache memory**

A computer has several different levels of cache memory. All levels of cache memory are faster than the RAM. The cache which is closer to the CPU is always faster than the other levels but it costs more and stores less data than other levels. As multiple processors operate in parallel, and independently multiple caches may possess different copies of the same memory block, this creates cache coherence problem. Cache coherence schemes help to avoid this problem by maintaining a uniform state for each cached block of data.

**Types of Cache Memory in a CPU**

**Level 1 or L1 Cache Memory**

The L1 cache memory is built on processor chip and it is very fast because it runs on the speed of the processor. It is also called primary or internal cache. It has less memory compared to other levels of cache and can store up to the 64kb cache memory. This cache is made of SRAM (Static RAM). Each time the processor requests information from memory, the cache controller on the chip uses special circuitry to first check if the memory data is already in the cache. If it is present, then the system is spared from the time-consuming access to the main memory. L1 cache is also usually split two ways, into the instruction cache and the data cache. The instruction cache deals with the information about the operation that the CPU has to perform, while the data cache holds the data on which the operation is to be performed.

Examples of L1 cache are accumulator, Program counter and address register, etc

**Level 2 or L2 Cache Memory**

The L2 cache memory is larger but slower than L1 cache. It is used to see recent accesses that are not picked by the L1 cache and it usually stores 64kb to the 2MB cache memory. An L2 cache is also found on the CPU. If L1 and L2 cache are used together, then the missing information that is not present in the L1 cache can be retrieved quickly from the L2 cache. Like L1 caches, L2 caches are composed of SRAM but they are larger. L2 is usually a separate static RAM (SRAM) chip and it is located between the CPU and DRAM (Main memory).

**Level 3 or L3 Cache Memory**

The L3 Cache memory is an enhanced form of memory present on the motherboard of the computer. It is an extra cache built into the motherboard between the processor and main memory to speed up the processing operations. It reduces the time gap between request and retrieving of the data and instructions much more quickly than the main memory. L3 cache is being used with processors nowadays, having more than 3MB of storage in it.

**What are sockets?**

Sockets allow communication between two different processes on the same or different machines. To be more precise, it's a way to talk to other computers using standard Unix file descriptors

To a programmer, a socket looks and behaves much like a low-level file descriptor. This is because commands such as read() and write() work with sockets in the same way they do with files and pipes.

There are four types of sockets available to the users. The first two are most commonly used and the last two are rarely used.

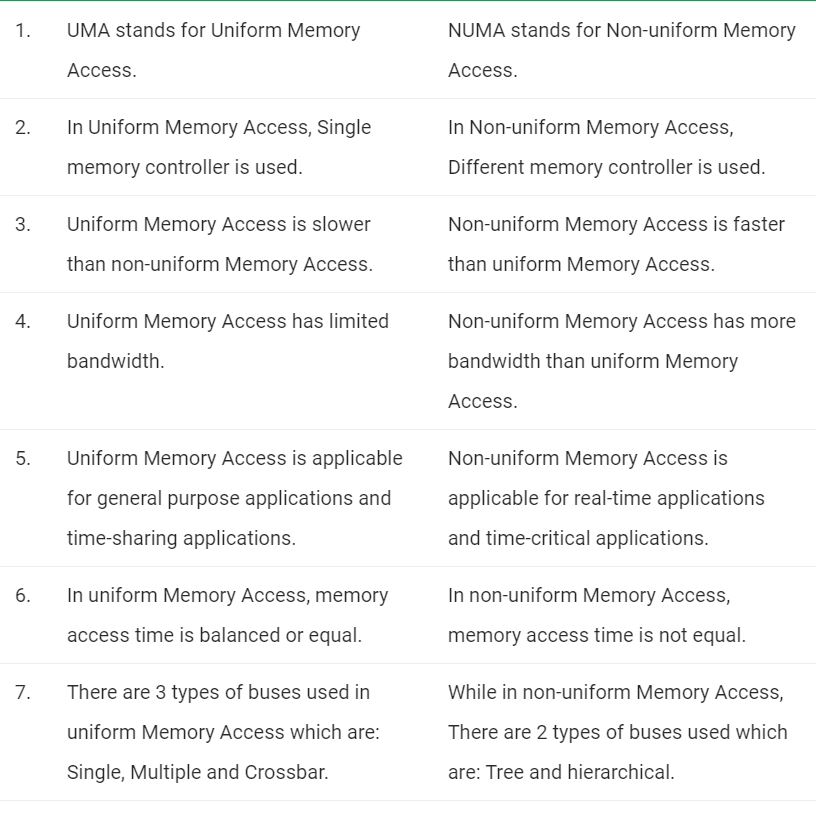
**Stream Sockets** − Delivery in a networked environment is guaranteed. If you send through the stream socket three items "A, B, C", they will arrive in the same order − "A, B, C". These sockets use TCP (Transmission Control Protocol) for data transmission. If delivery is impossible, the sender receives an error indicator. Data records do not have any boundaries.

**Datagram Sockets** − Delivery in a networked environment is not guaranteed. They're connectionless because you don't need to have an open connection as in Stream Sockets − you build a packet with the destination information and send it out. They use UDP (User Datagram Protocol).

**Raw Sockets** − These provide users access to the underlying communication protocols, which support socket abstractions. These sockets are normally datagram oriented, though their exact characteristics are dependent on the interface provided by the protocol. Raw sockets are not intended for the general user; they have been provided mainly for those interested in developing new communication protocols, or for gaining access to some of the more cryptic facilities of an existing protocol.

**Sequenced Packet Sockets** − They are similar to a stream socket, with the exception that record boundaries are preserved. This interface is provided only as a part of the Network Systems (NS) socket abstraction, and is very important in most serious NS applications. Sequenced-packet sockets allow the user to manipulate the Sequence Packet Protocol (SPP) or Internet Datagram Protocol (IDP) headers on a packet or a group of packets, either by writing a prototype header along with whatever data is to be sent, or by specifying a default header to be used with all outgoing data, and allows the user to receive the headers on incoming packets.

**Uniform Memory Access vs Non Uniform Memory Access**

****

**What is context Switching?**

Context Switching involves storing the context or state of a process so that it can be reloaded when required and execution can be resumed from the same point as earlier. This is a feature of a multitasking operating system and allows a single CPU to be shared by multiple processes.

**Clustering and how Clusters handle Load Balancing**

A cluster is a group of resources that are trying to achieve a common objective, and are aware of one another. Clustering usually involves setting up the resources (servers usually) to exchange details on a particular channel (port) and keep exchanging their states, so a resource’s state is replicated at other places as well. It usually also includes load balancing, wherein, the request is routed to one of the resources in the cluster as per the load balancing policy.

Load balancing can also happen without clustering when we have multiple independent servers that have same setup, but other than that, are unaware of each other. Then, we can use a load balancer to forward requests to either one server or other, but one server does not use the other server’s resources. Also, one resource does not share its state with other resources. Each load balancer basically does following tasks: Continuously check which servers are up. When a new request is received, send it to one of the servers as per the load balancing policy. When a request is received for a user who already has a session, send the user to the same server.

**What is Multithreading?**

Multithreading is the capability of a processor or a single core in a multicore processor to be able to produce thread of execution concurrently.

In a multithreaded application, the threads share the resources of a single or multiple cores, which include the computing units, the CPU caches, and other essential resources.

**Simultaneous Multithreading**- Simultaneous multithreading (SMT) is a technique for improving the overall efficiency of CPUs with hardware multithreading. SMT permits multiple independent threads of execution to better utilize the resources provided by modern processor architectures.

The name multithreading is ambiguous, because not only can multiple threads be executed simultaneously on one CPU core, but also multiple tasks/processes (with different page tables, different task state segments, different protection rings, different I/O permissions, etc).

Two concurrent hardware threads per CPU core are the most common, but some processors support up to eight concurrent threads per core.

**Hyperthreading**- Hyper-threading is Intel’s proprietary simultaneous multithreading implementation used to improve parallelization of computations performed on x86 microprocessors.

**What is Interconnect and its types used in HPCs?**

Interconnect is the way by which various computers communicate with each other. The biggest advantage HPCs have over ordinary consumer level computers is the Interconnect technology used by them, which allows them to significantly boost efficiency, and allows them to utilise the resources of other computers. High performance system interconnect technology can be divided into three categories: Ethernet, InfiniBand, and vendor specific interconnects, which includes custom interconnects the recently introduced Intel Omni-Path technology.

**Ethernet as an Interconnect**- Ethernet is established as the dominant low level interconnect standard for mainstream commercial computing requirements. Above the physical level, the software layers to coordinate communication resulted in TCP/IP becoming widely adopted as the primary commercial networking protocol. Ethernet is established as the dominant low level interconnect standard for mainstream commercial computing requirements. Above the physical level, the software layers to coordinate communication resulted in TCP/IP becoming widely adopted as the primary commercial networking protocol.

**Infiniband as an Interconnect**- InfiniBand is designed for scalability, using a switched fabric network topology together with remote direct memory access (RDMA) to reduce CPU overhead. The InfiniBand protocol stack is considered less burdensome than the TCP protocol required for Ethernet. This enables InfiniBand to maintain a performance and latency edge in comparison to Ethernet in many high performance workloads, and is generally used in Cluster Computers