

[Courses](#)[Login](#)[Suggest an Article](#)

Operating System | Multiple-Processor Scheduling

In multiple-processor scheduling **multiple CPU's** are available and hence **Load Sharing** becomes possible. However multiple processor scheduling is more **complex** as compared to single processor scheduling. In multiple processor scheduling there are cases when the processors are identical i.e. HOMOGENEOUS, in terms of their functionality, we can use any processor available to run any process in the queue.

Approaches to Multiple-Processor Scheduling –

One approach is when all the scheduling decisions and I/O processing are handled by a single processor which is called the **Master Server** and the other processors executes only the **user code**. This is simple and reduces the need of data sharing. This entire scenario is called **Asymmetric Multiprocessing**.

A second approach uses **Symmetric Multiprocessing** where each processor is **self scheduling**. All processes may be in a common ready queue or each processor may have its own private queue for ready processes. The scheduling proceeds further by having the scheduler for each processor examine the ready queue and select a process to execute.

Processor Affinity –

Processor Affinity means a processes has an **affinity** for the processor on which it is currently running.

When a process runs on a specific processor there are certain effects on the cache memory. The data most recently accessed by the process populate the cache for the processor and as a result successive memory access by the process are often satisfied in the cache memory. Now if the process migrates to another processor, the contents of the cache memory must be invalidated for the first processor and the cache for the second processor must be repopulated. Because of the high cost of invalidating and repopulating caches, most of the SMP(symmetric multiprocessing) systems try to avoid migration of processes from one processor to another and try to keep a process running on the same processor. This is known as **PROCESSOR AFFINITY**.

There are two types of processor affinity:

1. **Soft Affinity** – When an operating system has a policy of attempting to keep a process running on the same processor but not guaranteeing it will do so, this situation is called soft affinity.
2. **Hard Affinity** – Some systems such as Linux also provide some system calls that support Hard Affinity which allows a process to migrate between processors.

Load Balancing –

Load Balancing is the **phenomena** which keeps the **workload** evenly **distributed** across all processors in an SMP system. Load balancing is necessary only on systems where each processor has its own private queue of process which are eligible to execute. Load balancing is unnecessary because once a processor becomes idle it immediately extracts a runnable process from the common run queue. On SMP(symmetric multiprocessing), it is important to keep the workload balanced among all processors to fully utilize the benefits of having more than one processor else one or more processor will sit idle while other processors have high workloads along with lists of processors awaiting the CPU.

There are two general approaches to load balancing :

1. **Push Migration** – In push migration a task routinely checks the load on each processor and if it finds an imbalance then it evenly distributes load on each processors by moving the processes from overloaded to idle or less busy processors.
2. **Pull Migration** – Pull Migration occurs when an idle processor pulls a waiting task from a busy processor for its execution.

Multicore Processors –

In multicore processors **multiple processor** cores are places on the same physical chip. Each core has a register set to maintain its architectural state and thus appears to the operating system as a separate physical processor. **SMP systems** that use multicore processors are faster and consume **less power** than systems in which each processor has its own physical chip.

However multicore processors may **complicate** the scheduling problems. When processor accesses memory then it spends a significant amount of time waiting for the data to become available. This situation is called **MEMORY STALL**. It occurs for various reasons such as cache miss, which is accessing the data that is not in the cache memory. In such cases the processor can spend upto fifty percent of its time waiting for data to become available from the memory. To solve this problem recent hardware designs have implemented multithreaded processor cores in which

two or more hardware threads are assigned to each core. Therefore if one thread stalls while waiting for the memory, core can switch to another thread.

There are two ways to multithread a processor :

1. **Coarse-Grained Multithreading** – In coarse grained multithreading a thread executes on a processor until a long latency event such as a memory stall occurs, because of the delay caused by the long latency event, the processor must switch to another thread to begin execution. The cost of switching between threads is high as the instruction pipeline must be terminated before the other thread can begin execution on the processor core. Once this new thread begins execution it begins filling the pipeline with its instructions.
2. **Fine-Grained Multithreading** – This multithreading switches between threads at a much finer level mainly at the boundary of an instruction cycle. The architectural design of fine grained systems include logic for thread switching and as a result the cost of switching between threads is small.

Virtualization and Threading –

In this type of **multiple-processor** scheduling even a single CPU system acts like a multiple-processor system. In a system with Virtualization, the virtualization presents one or more virtual CPU's to each of virtual machines running on the system and then schedules the use of physical CPU'S among the virtual machines. Most virtualized environments have one host operating system and many guest operating systems. The host operating system creates and manages the virtual machines and each virtual machine has a guest operating system installed and applications running within that guest. Each guest operating system may be assigned for specific use cases, applications, and users, including time sharing or even real-time operation. Any guest operating-system scheduling algorithm that assumes a certain amount of progress in a given amount of time will be negatively impacted by the virtualization. In a time sharing operating system that tries to allot 100 milliseconds to each time slice to give users a reasonable response time. A given 100 millisecond time slice may take much more than 100 milliseconds of virtual CPU time. Depending on how busy the system is, the time slice may take a second or more which results in a very poor response time for users logged into that virtual machine. The net effect of such scheduling layering is that individual virtualized operating systems receive only a portion of the available CPU cycles, even though they believe they are receiving all cycles and that they are scheduling all of those cycles. Commonly, the time-of-day clocks in virtual machines are incorrect because timers take no longer to trigger than they would on dedicated CPU's.

Virtualizations can thus undo the good scheduling-algorithm efforts of the operating systems within virtual machines.

Reference –

Operating System Principles – Galvin

Recommended Posts:

Operating Systems | Set 1

Operating Systems | Set 2

Operating Systems | Set 3

Operating Systems | Set 4

Operating System | Critical Section

Operating Systems | Set 5

Operating Systems | Set 6

Operating Systems | Set 10

Operating Systems | Set 7

Operating Systems | Set 8

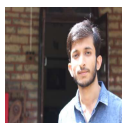
Operating Systems | Set 9

Operating Systems | Set 11

Operating Systems | Set 12

Operating Systems | Set 13

Operating Systems | Set 14



Ayush_Pandey_22

Check out this Author's [contributed articles](#).

If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please Improve this article if you find anything incorrect by clicking on the "Improve Article" button below.

Article Tags : [Operating Systems](#)

Practice Tags : [Operating Systems](#)

Be the First to upvote.

☐ To-do ☐ Done**3**Based on **8** vote(s)[Feedback/ Suggest Improvement](#)[Add Notes](#)[Improve Article](#)Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.[Load Comments](#)[Share this post!](#)

A computer science portal for geeks

710-B, Advant Navis Business Park,
Sector-142, Noida, Uttar Pradesh - 201305
feedback@geeksforgeeks.org

COMPANY

[About Us](#)
[Careers](#)
[Privacy Policy](#)
[Contact Us](#)

PRACTICE

[Company-wise](#)
[Topic-wise](#)
[Contests](#)
[Subjective Questions](#)

LEARN

[Algorithms](#)
[Data Structures](#)
[Languages](#)
[CS Subjects](#)
[Video Tutorials](#)

CONTRIBUTE

[Write an Article](#)
[Write Interview Experience](#)
[Internships](#)
[Videos](#)

