If it happens that you are confused by comparisons like the ones listed below then you came to the right place so please proceed and read the entire article for a better overview:

- Multiprogramming vs multiprocessing
- Multitasking vs multiprocessing
- Multitasking vs multithreading

Introduction

In the context of computing and operating systems, one might encounter many (confusing) terms which may look similar but eventually refer to different concepts. In this post, I will try to summarize the basic differences between various operating systems types and explain why and how they are not the same. The information provided is not new and can be found all over the place on the internet and actually that may add to the confusion. I hope that having all (not exactly) the terms clarified in one place would make it easy to remember. There could be many reasons why there are different operating systems types but it is mainly because each system type is designed from the ground up to meet certain design goals. On the other hand, the underlying hardware architecture influences the way systems are designed and implemented.

In a typical computing system, probably there are (many) concurrent application processes competing for (few) resources, for example the CPU. An operating system must handle resource allocation with due care. There is a popular operating systems term for such allocation known as scheduling. Depending on operating system's type, the goals of scheduling processes can be different. Regardless of system type there should be some kind of fairness when allocating resources, stated policy must be enforced and the overall system usage should be balanced. As we will see later, the goals that need to be achieved characterize the operating system type. Let us now talk about some operating systems types and terminology…

What is multiprogramming

One time, I was at the post office standing in line waiting my turn to be served. My turn came but I was not fully prepared because my mail was not prepackaged. They gave me an empty box to do that on the side. I started packaging my mail while another customer is occupying my spot. It does not make sense to block the whole line while packaging my mail however it is a better idea to allow other customers proceed and get served in the mean time. I think this example (to some extent) is very similar in concept to multi programming model where programs are like customers and CPU is like the post office assistant. Assuming one assistant (single processor system) then only one customer can be served at a time. While a customer is being served he or she continues until he or she finishes or waits on the side. As long as the assistant is helping a customer he does not switch to serve other customers. So what is multiprogramming ?

In a multiprogramming operating system there are one or more programs (processes or customers) resident in computer's main memory ready to execute. Only one program at a time gets the CPU for execution while the others are waiting their turn. The whole idea of having a multi-programmed system is to optimize system utilization (more specifically CPU time). The currently executing program gets interrupted by the operating system between tasks (for example waiting for IO, recall the mail packaging example) and transfer control to another program in line (another customer). Running program keeps executing until it voluntarily gives the CPU back or when it blocks for IO. As you can see, the design goal is very clear: processes waiting for IO should not block other processes which in turn wastes CPU time. The idea is to keep the CPU busy as long as there are processes ready to execute.

Note that in order for such a system to function properly, the operating system must be able to load multiple programs into separate partitions of the main memory and provide the required protection

because the chance of one process being modified by another process is likely to happen. Other problems that need to be addressed when having multiple programs in memory is fragmentation as programs enter or leave (swapping) the main memory. Another issue that needs to be handled as well is that large programs may not fit at once in memory which can be solved by using virtual memory. In modern operating systems programs are split into equally sized chunks called pages but this is beyond the scope of this article.

In summary, Multiprogramming operating system allows multiple processes to reside in main memory where only one program is running. The running program keeps executing until it blocks for IO and the next program in line takes the turn for execution. The goal is to optimize CPU utilization by reducing CPU idle time. Finally, please note that the term multi-programming is an old term because in modern operating systems the whole program is not loaded completely into the main memory.

Multiprocessing

Multiprocessing sometimes refers to executing multiple processes (programs) at the same time. This is confusing because we already have multi-programming (defined earlier) and multitasking (will talk about it later) that are better to describe multiple processes running at the same time. Using the right terminology keeps less chance for confusion so what is multiprocessing then?

Multiprocessing refers actually to the CPU units rather than running processes. If the underlying hardware provides more than one processor then that is multiprocessing. There are many variations on the basic scheme for example having multiple cores on one die or multiple dies in one package or multiple packages in one system. In summary, multiprocessing refers to the underlying hardware (multiple CPUs, Cores) while multiprogramming refers to the software (multiple programs, processes). Note that a system can be both multi-programmed by having multiple programs running at the same time and multiprocessing by having more than one physical processor.

Multitasking

Multitasking has the same meaning as multiprogramming in the general sense as both refer to having multiple (programs, processes, tasks, threads) running at the same time. Multitasking is the term used in modern operating systems when multiple tasks share a common processing resource (CPU and Memory). At any point in time the CPU is executing one task only while other tasks waiting their turn. The illusion of parallelism is achieved when the CPU is reassigned to another task (context switch). There are few main differences between multitasking and multiprogramming (based on the definition provided in this article). A task in a multitasking operating system is not a whole application program (recall that programs in modern operating systems are divided into logical pages). Task can also refer to a thread of execution when one process is divided into sub tasks (will talk about multi threading later). The task does not hijack the CPU until it finishes like in the older multiprogramming model but rather have a fair share amount of the CPU time called quantum (will talk about time sharing later in this article). Just to make it easy to remember, multitasking and multiprogramming refer to a similar concept (sharing CPU time) where one is used in modern operating systems while the other is used in older operating systems.

Multi Threading

Before we proceed, let us recap for a minute. Multiprogramming refers to multiple programs resident in main memory and (apparently but not exactly) running at the same time. Multitasking refers to multiple processes running simultaneously by sharing the CPU time. Multiprocessing refers to multiple CPUs so where does multi threading fit in the picture.

Multi threading is an execution model that allows a single process to have multiple code segments (threads) run concurrently within the context of that process. You can think of threads as child processes that share the parent process resources but execute independently. Multiple threads of a single process can share the CPU in a single CPU system or (purely) run in parallel in a multiprocessing system. A multitasking system can have multi threaded processes where different processes share the CPU and at the same time each has its own threads.

The question is why we need to have multiple threads of execution within a single process context. Let me give an example where it is more convenient to have a multi threaded application. Suppose that you have a GUI application where you want to issue a command that require long time to finish for example a complex mathematical computation. Unless you run this command in a separate execution thread you will not be able to interact with the main application GUI (for example updating a progress bar) because it is going to be frozen (not responding) while the calculation is taking place.

Multi threading is a smart way to write concurrent software but it also comes with a price because the programmer has to be aware of race conditions when two or more threads try to access a shared resource and leave the system in an inconsistent state or a deadlock. Thread synchronization (for example using locks or semaphores) is used to solve this problem which is beyond the scope of this article.

Time Sharing

Recall that in a single processor system, parallel execution is an illusion. One instruction from one process at a time can be executed by the CPU even though multiple processes reside in main memory. Imagine a restaurant with only one waiter and few customers. There is no way for the waiter to serve more than one customer at a time but if it happens that the waiter is fast enough to rotate on the tables and provide food quickly then you get the feeling that all customers are being served at the same time. This is the example of time sharing when CPU time (or waiter time) is being shared between processes (customers). Multi programming and multitasking operating systems are nothing but time sharing systems. In multi programming though the CPU is shared between programs it is not the perfect example on CPU time sharing because one program keeps running until it blocks however in a multitasking (modern operating system) time sharing is best manifested because each running process takes only a fair amount of the CPU time called quantum time. Even in a multiprocessing system when we have more than one processor still each processor time is shared between running processes. As you can see all terms are somehow related in one way or another however not using the right term in the right context is what makes the confusion so keep that in mind.

Real Time System

At the beginning of this article we mentioned that a system is characterized by the goals that need to be achieved. In a typical time sharing operating system processes are scheduled so that CPU time is shared among the group. Depending on the scheduling algorithm, each process gets its share amount of CPU time but there is no guarantee that a process gets the CPU whenever it wants to. On the other hand, in a real time system a process is guaranteed to get CPU attention when a specific event happens. There should be an operational deadline from the time the event is triggered to the time the system responds back. Processes in a real time system are mission critical for example in the case of industrial robots in an assembly line where at each stage a certain operation is expected to take place.

Conclusion

Multi programming, multitasking, multi threading, time sharing and real time systems all refer to software implementation of scheduling processes for CPU execution. Each scheduling

implementation serves certain design goals that characterize a particular operating system type. Multiprocessing on the other hand refers to the number of CPU units the underlying hardware provides.

That is all for today, so what do you think of the article ? is it useful ? Please use the comments section below for questions, corrections or feedback. Thanks for reading.