Operating systems play a major role in modern computing, in the management of hardware and software resources. The major functionalities being; processor management, multitasking, file systems, memory management and parallelization. I shall approach the parallelization aspect and its unique implementation provided by Microsoft Windows and Linux by comparing their respective advantages and disadvantages and how each operating system is efficient in task execution, in regards to the increasing demands of high-performance computing.

Parallelization has revolutionized computing allowing one to perform tasks simultaneously by dividing them into smaller, concurrent operations and have better control on modern multi-core processors.

First of all, when did the need of parallelization appear ? It started when upgrading one’s processor wasn’t enough to speed up applications and enhance performances. Though in the early days of computing, when single-core processing was the main way of making processors, talk about making processors smaller for lighter products and easier insertion started to appear, at first shrinking the size of transistors was feasible with the technological advances available, microprocessors were made. However, they simply became too small thus further miniaturization became too complex and cost inefficient also decreasing the rate of progress (Mishra, 2023).

Therefore, arose the need of an more efficient alternative; parallelism. Allowing multiple tasks to be performed simultaneously, by distributing computation across multiple cores or GPUs, precisely answering the demand of better processing.

To further reinforce the application of parallelism, modern applications such as data analytics, AI and simulations need huge computing power. Data analytics must process enormous datasets involving sorting, filtering, aggregating and generating insights from vast information, it seems pretty evident that multiple tasks must be processed and if such had to be done one at time, it would heavily slow down the processing process and highly reduce the efficiency (Mishra, 2023). With parallel processing they can be split into smaller, concurrent tasks allowing fast analysis and data-driven decisions.

The same explanation can be used for AI, requiring deep learning of neural networks and scientific or engineering simulations like weather forecasts or fluid dynamics which require intricate numerical calculations.

For Microsoft, parallelization was introduced for its large user base and for developers looking for solutions to use parallelization without much effort. APIs made to exploit the concept such as Parallel Patterns Library (PPL) and Task Parallel Library (TPL) to handle threads effectively – a thread of execution is the smallest sequence of programmed instruction that can be managed individually by a schedular, which is typically part of the operating system (Wikipedia, 2024); in many cases it is a component of process (which, notably, is the case in this situation).

A small parenthesis to talk about threads; in Flynn’s Taxonomy it would be SIMT (Single Instruction Multiple Threads) a modern and optimized array processor as a variant of SIMD (Single Instruction Multiple Data: it executes the same instruction on multiple data elements concurrently, widely used in GPUs and multimedia applications). This variant would enable GPUs to simply run of multiple threads on separate data.

Linux on the other hand, offers more flexibility and better performances with pthreads and OpenMP which grant granular control on parallelization, helping for tasks like clusters of intricate calculations for example. Linux also grants deep personalization ideal for supercomputers and embedded systems for companies or users whom need to have a grasp on the entirety of their systems (TylerMSFT, alexbuckgit, v-kents, nschonni, mikeblome, Mikejo5000, ghogen, BrianPeek & Saisang, 2021).

To go further into details with the solutions provided by Windows and Linux, starting with Windows Parallel Patterns Library; it’s an API which simplifies parallel execution by using ThreadPool; it allocates threads dynamically depending on the tasks at hand, maximising the effectivity without the need of manual intervention (TylerMSFT, v-kents, nxtn, Mikejo5000, mikeblome, ghogen & Saisang, 2021). Parallel Patterns Library is also a key component to the Concurrency Runtime, thus allowing parallel processing of tasks via constructions such as parallel\_for\_each or more secure containers such as concurrent\_vector, enhancing the fluidity of interactives applications as well.

Then there is Task Parallel Library, integrated in .NET, it facilitates parallelism with abstractions like Task, Parallel.For and PLINQ (IEvangelist, tdykstra, BillWagner, gewarren, DennisLee-DennisLee, nxtn, mairaw, pkulikov, ChrisMaddock, Mikeio5000, mjhoffman65, guardrex, tompratt-AQ & yishengjin1413, 2022). These tools allow precise parallelization without directly handling threads.

Finally, Microsoft MPI (Message Passing Interface), as the name suggests, its an MPI adapted for Microsoft which allows communication between high performance computing nodes by dividing the tasks between the machines and keeping them interconnected (AnnaDaly, tfosmark, alexbuckgit, DhurataJ, jithinjosepkl & DanMSFT, 2022).

Now with Linux which focuses on flexibility being one of its best aspects, starting with POSIX Threads also known as pthreads, is an execution model independent of a programming language which allows one to create and handle threads. More or less, a threads library offering granular control over said library, however its use does requires some technique and expertise in the functionality of parallelism and SIMT.

Secondly, OpenMP is high tech parallelization API that support multi-platform shared-memory multiprocessing programming, essentially an implementation of multithreading a method of parallelization in which a primary thread stems into sub-threads and divides the tasks between said sub-threads, which then run concurrently also allowing easy communication interprocessus between the clusters as everything is more or less connected. This kind of API is ideal for scientific calculations.

And last but not least, Completely Fair Scheduler (CFS); Linux uses Completely Fair Scheduler at its core and as its name suggests, it’s a scheduling algorithm, it plans processes using a balanced and structured tree system (red-black trees) to efficiently and fairly schedules tasks to complete across multiple cores, essentially the “ideal multi-tasking CPU”. It is also highly customizable for different kinds of tasks for different users or companies (Anonymous, n.d.).

Now onto the strengths and weaknesses of both aspects on Windows and Linux:

Windows’ solutions have user-friendly interfaces, abstract of technical details and are suitable for interactive applications, though less flexible for more advanced applications and may be overloaded with several threads as they are very thread heavy programs, as most if not all programs rely on them, which could overclocking if the CPU isn’t fast enough. Despite that, it is great for most applications available to the public and interactive applications (e.g.: GUI, videos games). Integrated in most Windows environments.

Linux’s solutions are highly flexible, customizable for specific cases and are very high-performance, these great perks do entail harder and more complex handling resulting in higher margins of error when handling these programs and can also result in bad configurations. In spite of the high skill cap in handling Linux’s programs they are highly rewarding (high risk 🡪 high reward) as they are great for scientific intricate and complex calculations, high-performance servers or supercomputers which require overall control over every step for the calculations and a grasp on the entirety of a server or a computer processes. Compatible with a large variety of material and open source software.

As you can probably tell, they serve different purposes thus neither is better than the other as they suit totally different aspects because their objectives are different. Windows aims to provide solutions for a large number of users, ready for use for intermediate developers also aiming to maximise the productivity for those users. Whilst Linux, aims for more experienced users and a variety of tasks by providing great flexibility and customization at core level. Made for high-tech companies and environments needing high-level performances.

In conclusion, parallelization is an essential domain in modern IT, Windows and Linux provide solutions adapted to their respective audiences and their demands. Windows aims for a more user-friendly integration whereas Linux aims for more flexibility and optimization. These choices reflect the essential needs of each system: answering the interactive needs of Windows users and the high-level complex techniques demanded by Linux users. This contrast in needs shows how the available uses of parallelization can provide depending on the challenges it has to face and will have to face in a constantly evolving world, specifically in the IT domain a never-ending resource and one with endless uses that we cannot even fathom at this point in history.

References:

TylerMSFT, alexbuckgit, v-kents, nschonni, mikeblome, Mikejo5000, ghogen, BrianPeek & Saisang. (2021). *Parallel Programming in Visual C++.*

<https://learn.microsoft.com/en-us/cpp/parallel/parallel-programming-in-visual-cpp?view=msvc-170>

AnnaDaly, tfosmark, alexbuckgit, DhurataJ, jithinjosepkl & DanMSFT. (2022). *Microsoft MPI.*

<https://learn.microsoft.com/en-us/message-passing-interface/microsoft-mpi>

IEvangelist, tdykstra, BillWagner, gewarren, DennisLee-DennisLee, nxtn, mairaw, pkulikov, ChrisMaddock, Mikeio5000, mjhoffman65, guardrex, tompratt-AQ & yishengjin1413. (2022). *Parallel programming in .NET: A guide to documentation.*

<https://learn.microsoft.com/en-us/dotnet/standard/parallel-programming/>

TylerMSFT, v-kents, nxtn, Mikejo5000, mikeblome, ghogen & Saisang. (2021). *Parallel Patterns Library (PPL).*

<https://learn.microsoft.com/en-us/cpp/parallel/concrt/parallel-patterns-library-ppl?view=msvc-170>

Mishra, D. (2023). *The Power of Parallel Programming: Why It’s a Necessity.*

<https://medium.com/@durganshu/the-power-of-parallel-programming-why-its-a-necessity-2429e4622f5e>

Wikipedia. (2024). Thread (computing).

<https://en.wikipedia.org/wiki/Thread_(computing)#:~:text=In%20computer%20science%2C%20a%20thread,a%20component%20of%20a%20process>.

Karl-Bridge-Microsoft, OptimalFailure, tilkinsc, DennisLee-DennisLee, v-kents, DCtheGeek, drewbatgit, mijacobs & msatranjr. (2022). *Creating Threads.*

<https://learn.microsoft.com/en-us/windows/win32/procthread/creating-threads>

Anonymous. (n.d.). *The Linux Kernel.*

<https://docs.kernel.org/scheduler/sched-design-CFS.html>