The OS I would design would act as a central platform for a medium-sized data analytics and cloud services company. Its core job would consist of both backend computational tasks as well as frontend interactions with end-users. The OS would have a graphical user interface (GUI) as an interface meant for everyday office users and a command line interface (CLI) meant as an interface intended for power users as well as programmers. The GUI would have simplified access to apps as well as configurations of systems, whereas CLI would allow scripting, batch jobs, as well as automation of complex assignments—a capability of use to users with massive datasets or repetitive simulations.

Incorporating concurrency into this OS is a must. Concurrency is the running of a multiplicity of processes or threads at a single time, a core characteristic of contemporary computing environments. In modern multi-core processors, concurrent running is not just possible but an absolute must for good performance and efficacy. This enables the system to perform many tasks concurrently, like running a database update, processing analytics, and serving a web client—all without having to wait for one task finishing before initiating another.

The requirement for concurrency results from hardware and software needs. Let us start with multi-user settings: many users can simultaneously gain access and require different services. Without concurrency, the OS would serve these requirements sequentially, causing delays and low system responsiveness. Other applications require background processes—such as real-time data synchronizing or routine security sweeps—that must run concurrently with the main activities of a user without interruption. In this reading, you've learned how concurrency enhances performance by having one process use the CPU while another one uses the disk drive, decreasing global execution time as compared with sequential execution (McClanahan, n.d.).

The benefits of introducing concurrency are considerable. One of the key gains is improved resource utilization. In a non-concurrent system, if an app is not utilizing system resources, these resources are idle. However, a concurrent system flexibly assigns these idle resources to other applications, optimizing efficiency. Another key gain of concurrency is enhanced average response time. A system queuing and running each app completely before running the next results in delays and reduced productivity. Instead, concurrency enables partial execution and context switching, producing a smoother and nimbler user experience.

Further, running numerous applications at once allows multitasking, a requirement of any efficient workplace. An analyst, for instance, may have a financial model running with a virus scan happening in the background as a cloud backup syncs data without any perceived performance lag. This is as much about convenience as about operational requirements of modern digital infrastructure. According to the text, concurrency improves overall performance of a system, especially when applications use different resources (McClanahan, n.d.).

But with concurrency also comes complexity. Programmers have had to cope with race conditions, deadlocks, and resource contention. Race conditions happen when two threads modify common data at the same time and produce a situation with unpredictable results. Deadlocks, which occur when two or more processes are waiting forever each for a resource held by the other, can bring systems to a grinding halt. These can all be overcome, however, by employing adequate concurrency control mechanisms like locking, scheduling algorithms of resources, and process prioritizations. Finally, creating this OS with concurrency isn't advantageous, it's indispensable. It matches state-of-the-art hardware capability and addresses contemporary software use demands. Flexibility of this system supporting both CLI as well as GUI ensures wide usability, and concurrency makes it efficient, responsive, as well as capable of executing versatile jobs critical to a firm’s functionality. Although concurrency problems demand strict handling, what you get is a stronger and more scalable OS.

---

**Reference:**

McClanahan, P. (n.d.). *Operating system: The basics*. Engineering LibreTexts. Retrieved June 27, 2025, from https://eng.libretexts.org/Courses/Delta_College/Operating_System%3A_The_Basics

**Wordcount:** 616