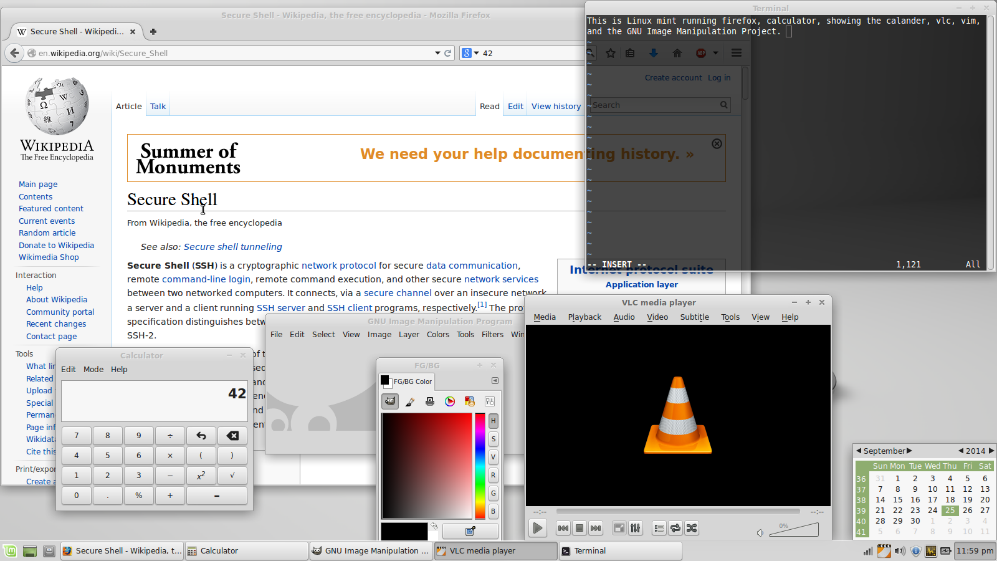
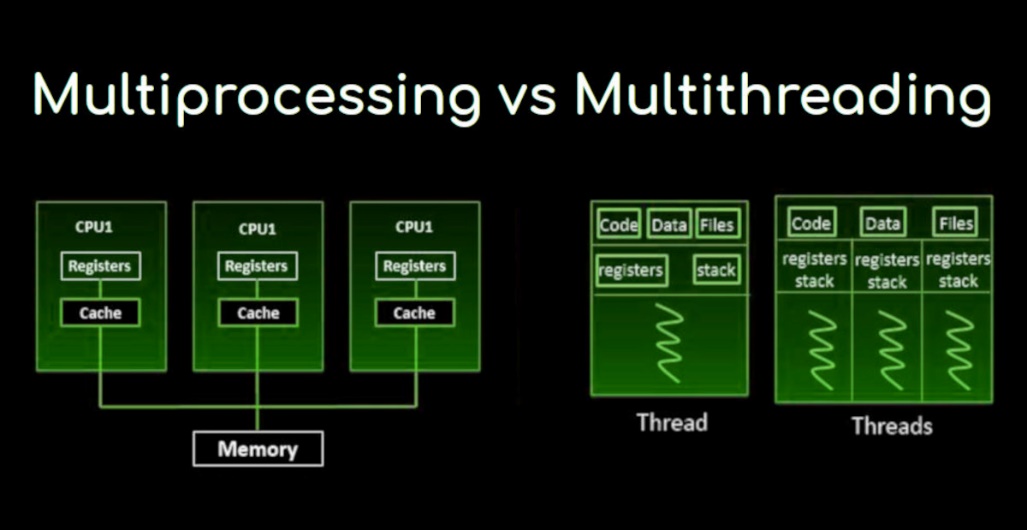
CHAPTER 20 MULTITHREADING AND MULTITASKING

In Chapter 20, we will delve into the intricacies of multitasking and multithreading in the Windows API. We will explore key concepts, provide clear explanations, relevant code examples, and insights from Charles Petzold's book. This chapter will address the following topics:

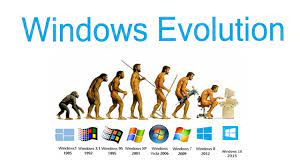
Multitasking: We will discuss the operating system's ability to run multiple programs concurrently, allocating time slices to each process. This creates the illusion of simultaneous execution and enhances system responsiveness.



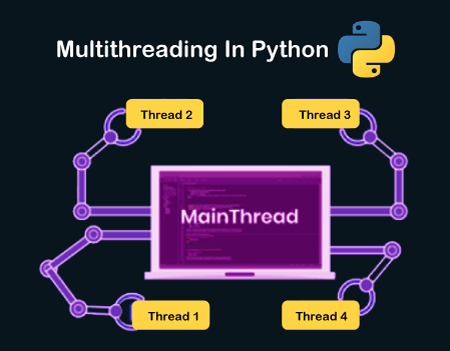
Multithreading: We will explore the ability of a single program to split its execution into multiple threads. Multithreading allows concurrent execution of tasks within the program, enabling background tasks, maintaining responsive user interfaces, and executing concurrent operations.



Windows Multitasking Evolution: We will examine the evolution of multitasking in Windows. In 16-bit Windows, multitasking capabilities were limited due to cooperative multitasking, where programs voluntarily yielded control to others. In 32-bit Windows, true multitasking using preemptive multitasking was introduced. The operating system actively assigns and revokes CPU time slices, ensuring responsiveness and preventing program monopolization.



Multithreading Benefits: We will discuss the benefits of multithreading, including the ability to perform background tasks without blocking user interaction, maintaining responsive user interfaces through separate UI update threads, and executing independent tasks simultaneously for improved performance on multiprocessor systems.

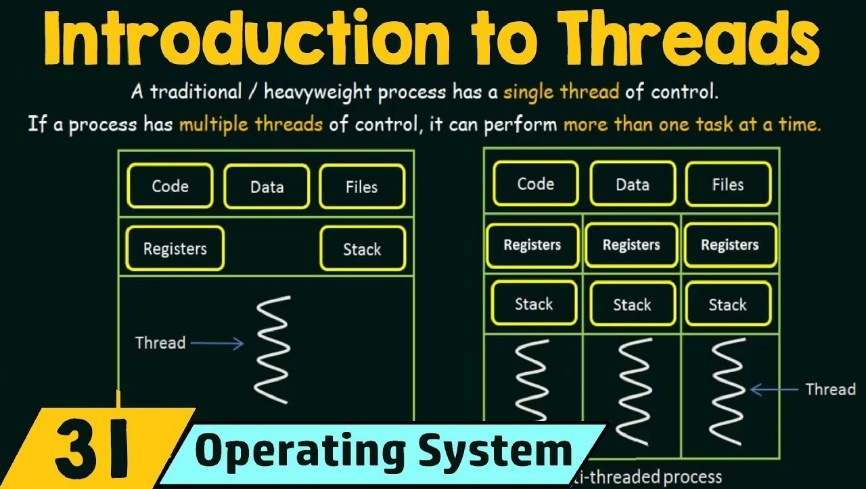


Key Terminology:

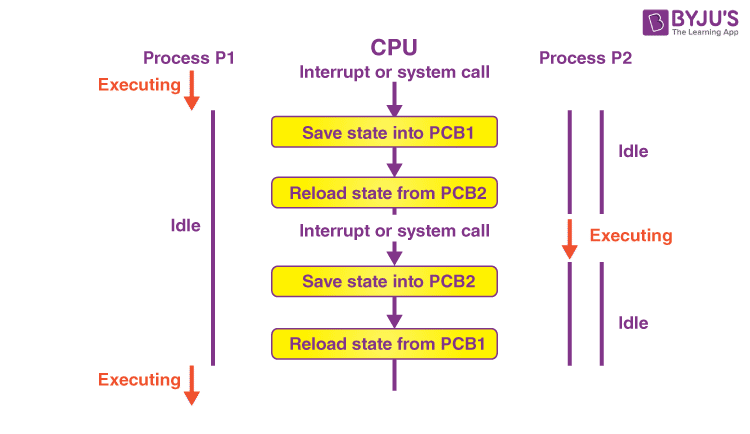
Process: A running instance of a program, with its own memory space and resources.



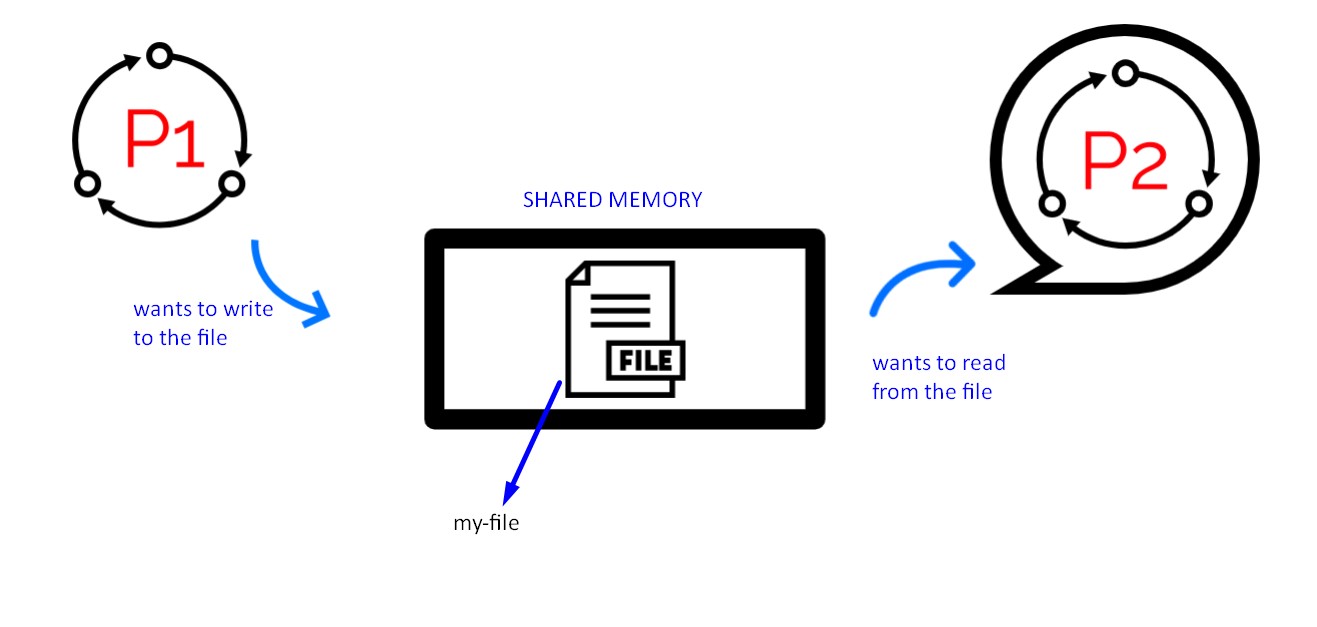
Thread: A lightweight execution unit within a process, sharing the process's memory and resources.



Context Switching: The process of saving and restoring a thread's state when switching between threads.



Synchronization: Mechanisms to coordinate access to shared resources among multiple threads, preventing data corruption and race conditions.



Topics Covered in Chapter 20:

* Thread Creation and Management: We will explore the CreateThread function for creating threads, setting thread priorities, suspending and resuming threads, and terminating threads.
* Synchronization Techniques: We will discuss synchronization mechanisms such as critical sections, mutexes, semaphores, and events. These mechanisms coordinate access to shared resources among multiple threads, preventing data corruption and race conditions.
* Thread-Specific Storage: We will examine thread-specific storage using the TlsAlloc, TlsGetValue, and TlsSetValue functions. Thread-specific storage allows each thread to have its own unique data.
* Win32 Timers: We will explore the use of timers in multithreaded programming using the SetTimer and KillTimer functions. Timers allow you to schedule recurring or one-time events in your program.
* Asynchronous Procedure Calls: We will discuss asynchronous procedure calls using the BeginThreadEx and QueueUserAPC functions. These functions allow you to execute code asynchronously in a separate thread.
* Multithreaded Programming Best Practices: We will provide best practices for multithreaded programming, including avoiding deadlocks, optimizing thread performance, and ensuring thread safety.