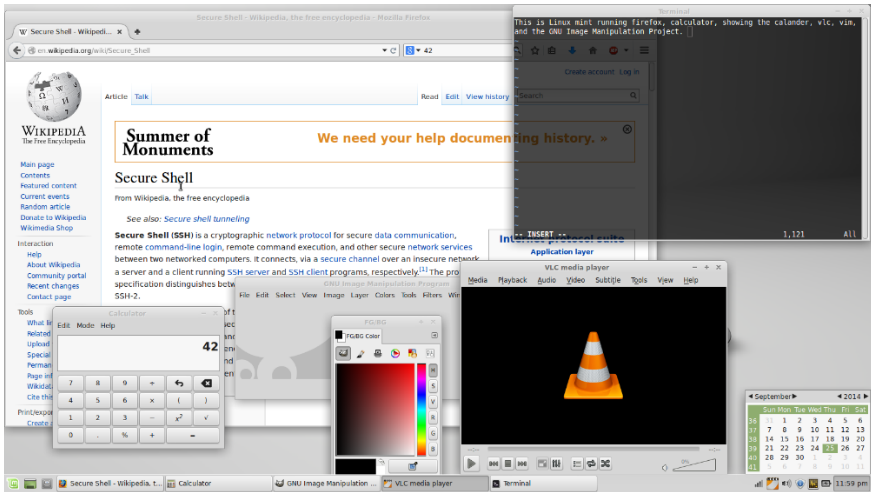
CHAPTER 20 MULTITHREADING AND MULTITASKING

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.



```
This is Linux mint running firefox, calculator, showing the calander, vlc, vim,
and the GNU Image Manipulation Project.
```

Secure Shell

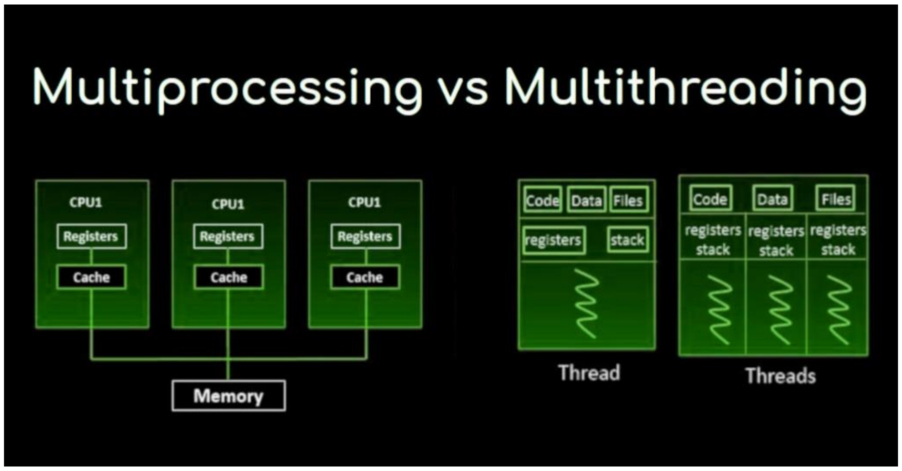
From Wikipedia, the free encyclopedia

See also: Secure shell tunneling

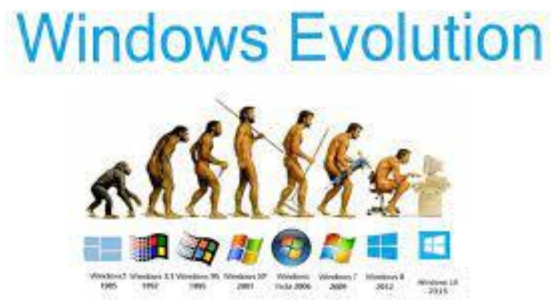
Secure Shell (SSH) is a cryptographic network protocol for secure data communication, remote command-line login, remote command execution, and other secure network services between two networked computers. It connects, via a secure channel over an insecure network, a server and a client running SSH server and SSH client programs, respectively.[1] The pro specification distinguishes betw SSH-2.

Secure Shell - Wikipedia, the free encyclopedia - Mozilla Firefox
en.wikipedia.org/wiki/Secure_Shell
Summer of Monuments
We need your help documenting history. »
Terminal
-- INSERT --
1,121
All
Calculator
42
GNU Image Manipulation Program
FG/BG Color
VLC media player
September
2014

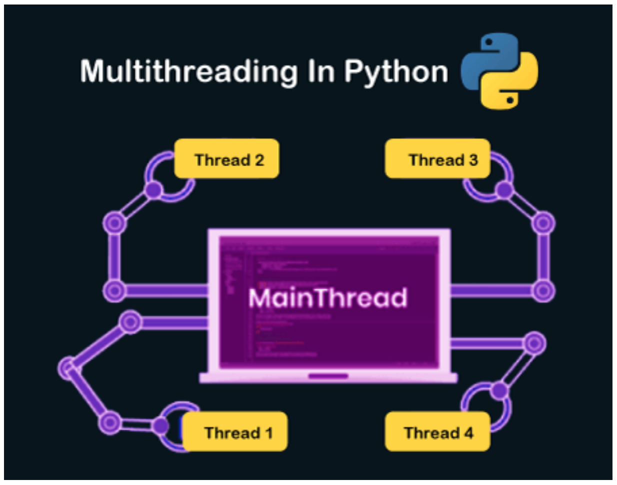
Multithreading: We will explore the ability of a single program to split its execution into multiple threads. Multithreading allows concurrent execution of tasks within the program, enabling background tasks, maintaining responsive user interfaces, and executing concurrent operations.



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.



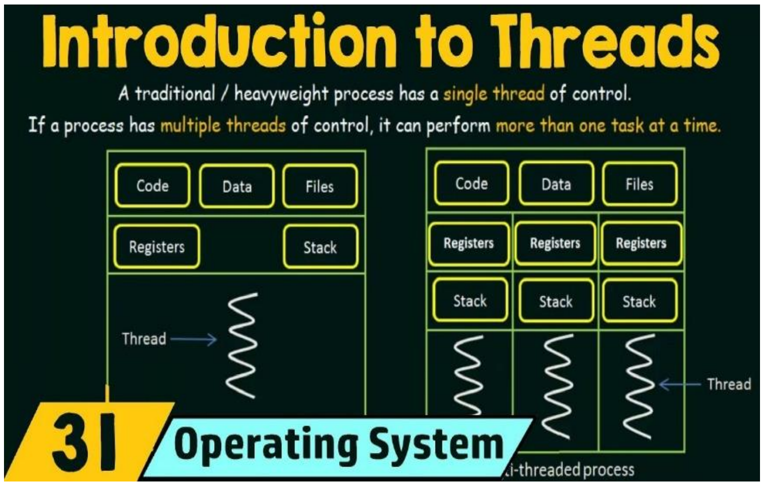
Multithreading Benefits: We will discuss the benefits of multithreading, including the ability to perform background tasks without blocking user interaction, maintaining responsive user interfaces through separate UI update threads, and executing independent tasks simultaneously for improved performance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



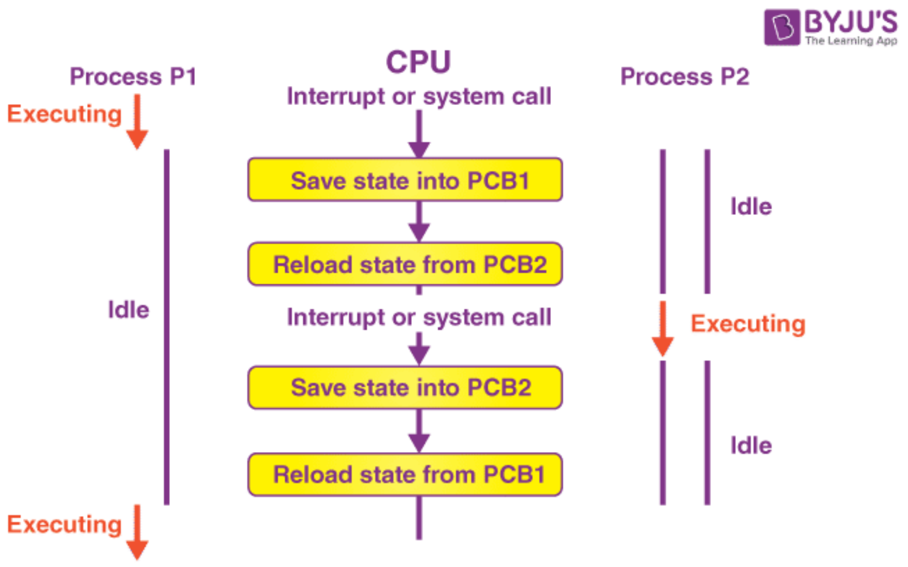
Key Terminology:

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

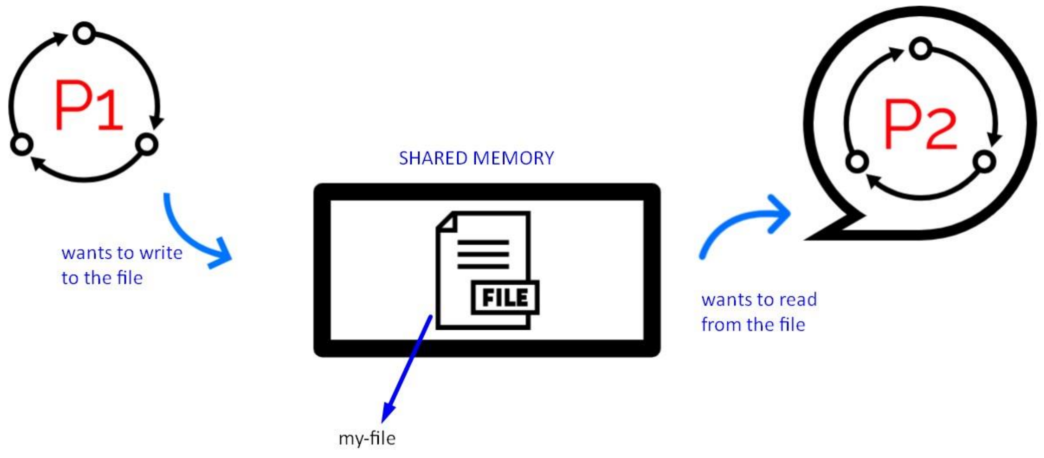
Thread: A lightweight execution unit within a process, sharing the process's memory and resources.


Introduction to Threads
A traditional / heavyweight process has a single thread of control.
If a process has multiple threads of control, it can perform more than one task at a time.
Code
Data
Files
Registers
Stack
Thread
Code
Data
Files
Registers
Registers
Registers
Stack
Stack
Stack
Thread
31
Operating System
multi-threaded process

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:

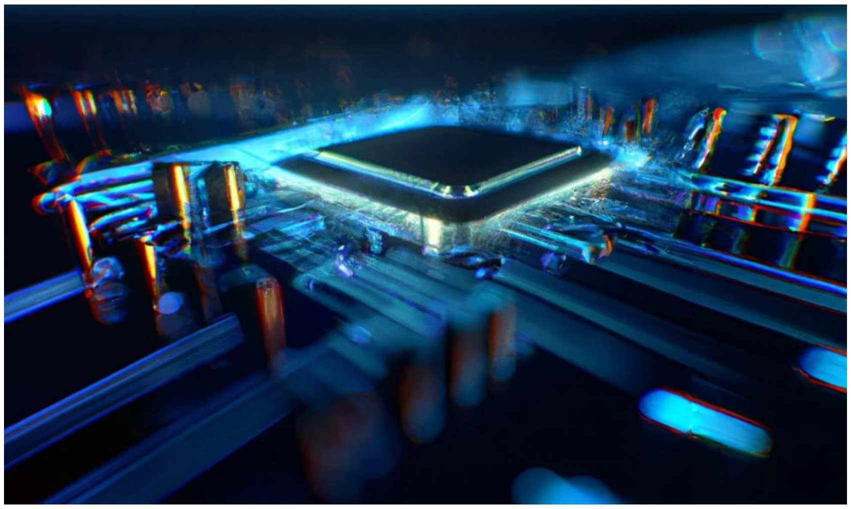
- Thread Creation and Management: We will explore the CreateThread function for creating threads, setting thread priorities, suspending and resuming threads, and terminating threads.
- Synchronization Techniques: We will discuss synchronization mechanisms such as critical sections, mutexes, semaphores, 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.

MULTITASKING IN THE DOS ERA: A TALE OF CREATIVITY AND LIMITATIONS

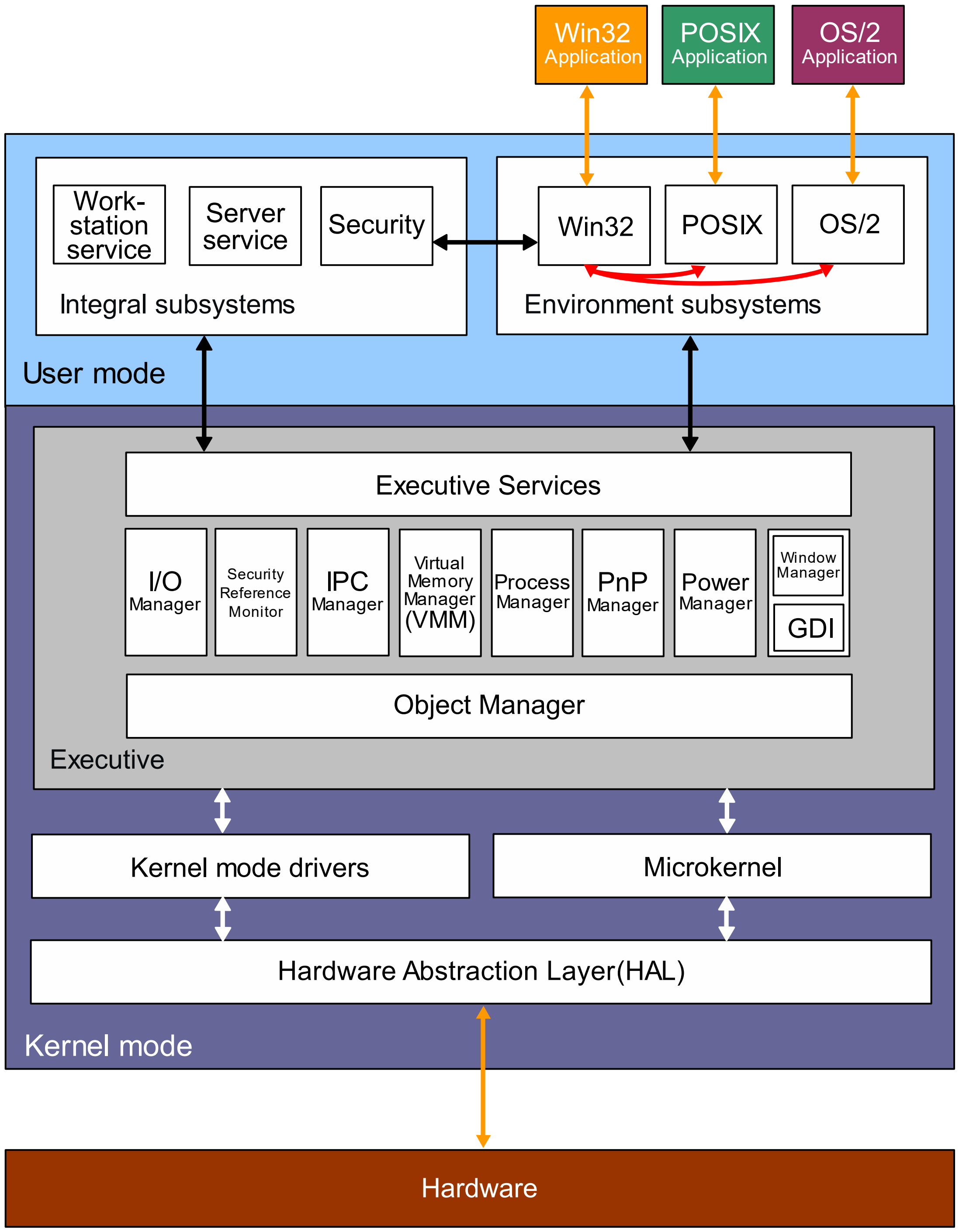
The early days of PC computing presented a fascinating paradox when it came to multitasking. While skeptics questioned its utility on single-user machines, users gradually gravitated towards its benefits, even before its complete realization. Understanding this story necessitates delving into the technical and pragmatic constraints that shaped multitasking's evolution under DOS.

Obstacles to Multitasking under DOS:

Hardware Limitations: The Intel 8088 lacked dedicated features for efficient memory management, crucial for juggling multiple programs and their memory allocations. Moving memory blocks to consolidate free space proved challenging, hindering robust multitasking implementations.



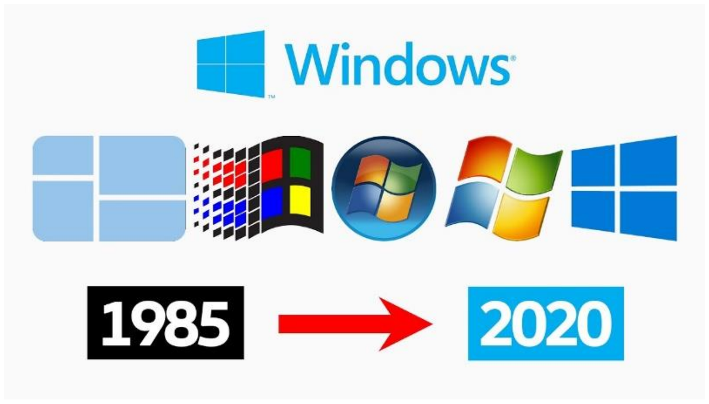
DOS Architecture: Designed for simplicity and minimal resource consumption, DOS offered limited APIs for programs to interact with the system beyond basic file access and program loading. This lack of robust system services hampered developers' ability to implement true multitasking within the OS itself.



Creative Workarounds:

Terminate-and-Stay-Resident (TSR) Programs: These innovative programs occupied a small portion of memory while remaining active in the background, even after switching to another program. Some TSRs, like print spoolers, leveraged the hardware timer interrupt to perform background tasks without impacting the foreground application. Others, like SideKick, employed task switching techniques, temporarily suspending the running program while displaying their own interface.

Enhanced DOS Features: Microsoft progressively added features to DOS, such as memory swapping to disk, that indirectly benefited multitasking by providing more flexible memory management.



Market Response and Limitations:

Limited Success of Task-Switching Shells: Attempts to build task-switching environments on top of DOS, like Quarterdeck's DesqView, offered rudimentary multitasking functionalities. However, their complexity and performance limitations prevented widespread adoption.

Key Takeaways:

- ✓ Multitasking emerged as a desired user experience on PCs despite technical limitations inherent to the platform and DOS architecture.
- ✓ Creative programmers tackled these limitations through TSRs and rudimentary task-switching approaches, paving the way for more advanced solutions.
- ✓ The limitations of these workarounds highlighted the need for a dedicated operating system capable of robust and user-friendly multitasking, leading to the eventual dominance of Windows.

Further Exploration:

- Investigate specific examples of popular TSRs and their functionalities.
- Analyze the technical challenges of memory management and context switching in the DOS environment.
- Compare and contrast the limitations of early DOS-based multitasking solutions with the capabilities of Windows 3.1 and beyond.

By understanding the ingenuity and constraints of the DOS era, we gain deeper appreciation for the advancements in multitasking that laid the groundwork for modern computing experiences.

MULTITASKING IN THE EARLY WINDOWS ERA:

Windows 1.0's Breakthrough: Introduced in 1985, Windows 1.0 offered a more sophisticated multitasking solution than TSRs or task-switching shells, even within the constraints of real mode.

Graphical Interface for Multitasking: Windows' graphical environment distinguished itself from command-line systems like UNIX by enabling multiple programs to run concurrently on the same screen, facilitating seamless switching and data exchange.

Nonpreemptive Multitasking: Cooperation Required:

Message-Based Architecture: Windows programs are primarily driven by messages, often originating from user input. They typically remain idle until a message arrives.

No Preemptive Time Slicing: 16-bit Windows did not enforce time-based task switching. Instead, control switched only when a program voluntarily returned control to Windows after processing a message.

Cooperative Nature: This reliance on programs to "play fair" and yield control earned it the name "cooperative multitasking." A poorly designed or unresponsive program could monopolize the system.

Exceptions and Workarounds:

Preemption for DOS and Multimedia: Windows did employ preemptive multitasking for running DOS programs and for multimedia tasks within DLLs, which required timely responses to hardware events.

Coping with Limitations:

Hourglass Cursor: A visual signal to the user that a program was busy, but not a true solution.

Windows Timer: Allowed periodic execution of code for tasks like animation and clocks.

PeekMessage Function: Enabled programs to periodically check for messages and relinquish control voluntarily, preventing complete unresponsiveness during long operations.

Key Takeaways:

- ✓ 16-bit Windows relied on nonpreemptive multitasking, a cooperative model that depended on programs to yield control regularly.
- ✓ This model had limitations, as a single program could potentially block others.
- ✓ Windows provided some mechanisms to mitigate these issues, but true preemptive multitasking would require a more robust foundation.

Further Exploration:

- Consider the challenges of implementing preemptive multitasking within the constraints of 16-bit Windows.
- Research specific examples of how nonpreemptive multitasking affected user experience and application design in early Windows programs.
- Explore the evolution of multitasking techniques in subsequent Windows versions, leading to the full-fledged preemptive model in 32-bit Windows.