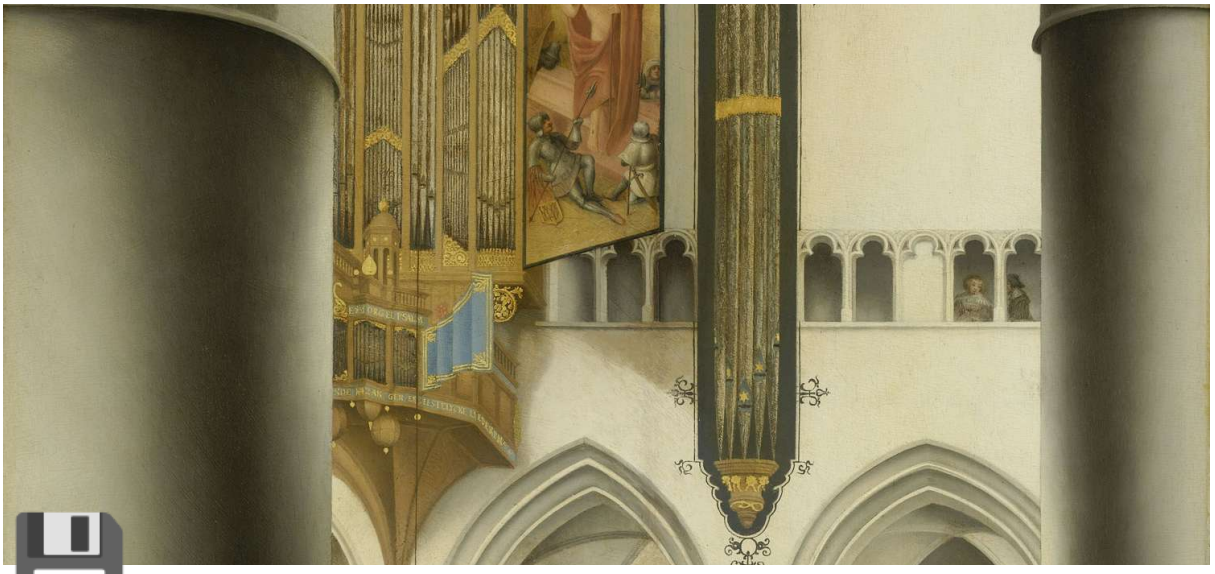


Distributed Systems Chapter III

Through Maya Mnaizel

We build virtual processors in software, on top of physical processors:

- Processor: It provides a set of instructions that can be automatically executed.
- Thread: A minimal software processor executes a series of instructions within its context. When a thread context is saved, the current execution is halted and all necessary data is stored to resume at a later time.
- Process: A software processor can execute one or more threads, which involves executing a series of instructions within the context of that thread.

Context Switching

- Processor context
  - The minimal set of values stored in a processor's registers for executing instructions.
- Thread Context
  - The minimal set of values stored in registers and memory is used to execute instructions.

- Process Context
  - The minimal set of values stored in registers and memory that is used for executing a thread.

  Threads share the same address space

  Thread context switching can be done entirely independent of the OS

  Process switching is generally more expensive as it involves getting the OS in the loop

  Creating and destroying threads is much cheaper than doing so for processes

  Why use threads?
  - Avoid needless blocking
  - Exploit parallelism
  - Avoid process switching
  - Threads use the same address space
  - maybe faster than process switching

The cost of a context switch

- Direct costs: actual switch and executing code of the handler
- Indirect cost: other costs

Should an OS kernel provide threads, or should they be implemented as user-level packages?

User-Space solution

- All operations can be handled efficiently within a single process.
- All services provided by the kernel are executed on behalf of the process in which a thread is residing.

- Threads are used when there are many external events

Kernel Solution

- Operations that block a thread are no longer a problem
  - The kernel schedules another available thread within the same process
- Handling external events is simple
  - The kernel schedules the thread associated with the event
- The problem is the loss of efficiency because each thread operation requires a trap to the kernel

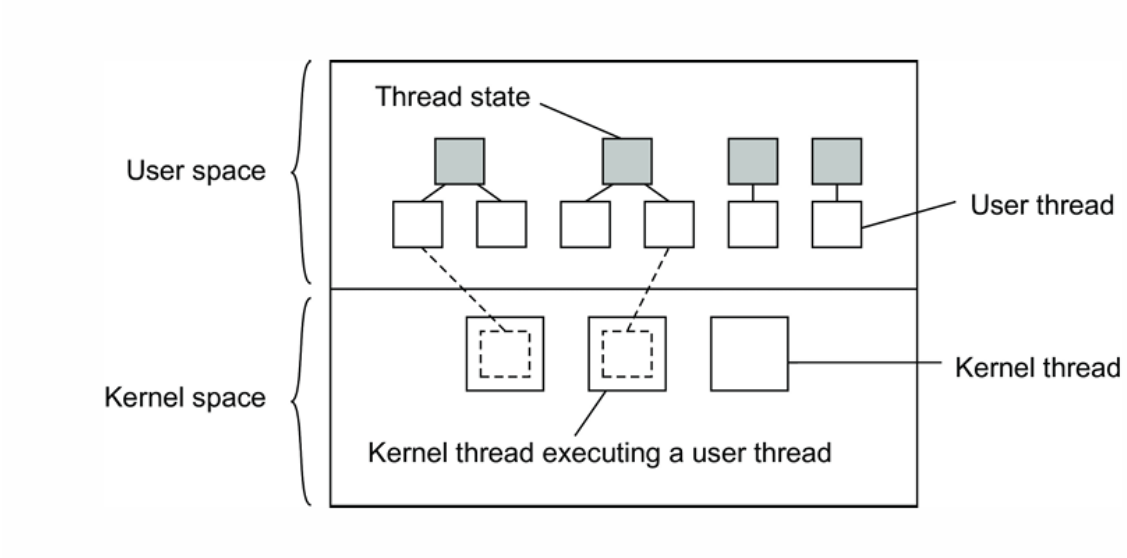
BUT

Attempting to combine user-level and kernel-level threads can increase complexity without necessarily improving performance.

User & Kernel Threads Combined

Two-Level Threading Approach

Kernel threads that can execute user-level threads



Principle operation

- The user thread that initiates a system call blocks, while remaining bound to the kernel thread executing it.
- The kernel can schedule another kernel thread that has a runnable user thread bound to it. Note that this user thread can switch to any other runnable user thread currently in user space.
- The user thread, which calls a blocking user-level operation, is switched to a runnable user thread, which is bound to the same kernel thread.
- When there are no user threads to schedule, a kernel thread may remain idle, and may even be removed (destroyed) by the kernel.

Using Threads at the client side

Multithreaded web client

Hiding network latencies:

- Web browser scans an incoming HTML page, and finds that more files need to be fetched
- Each file is fetched by a separate thread, each doing a (blocking) HTTP request
- As files come in, the browser displays them

Multiple request-response calls to other machines (RPC)

- A client does several calls at the same time, each one by a different thread
- It then waits until all results have been returned

Using threads on the server side

Improve performance

- Starting a thread is cheaper than starting a new process
- When a server is single-threaded, it cannot be easily scaled up to a multiprocessor system.
- When working with clients, the network latency can be hidden by responding to the next request while the previous one is still being answered.

Better Structure

- Many servers have high Input/output (I/O) rates. To simplify the server's structure, it is recommended to use simple and well-understood blocking calls.
- Programs using multithreading tend to have a simplified flow of control, making them smaller and easier to understand.

Model	Characteristics
Multithreading	Parallelism, blocking system calls
Single-threaded process	No Parallelism, blocking system calls
Finite-state machine	Parallelism, Nonblocking system calls