Principles of Distributed Systems

inft-3507

Dr. J.Burns

ADA University

Autumn 2025

Section 3: Processes

This content is based on the following public resources: https://www.distributed-systems.net/index.php/books/ds4/

Threads

Introduction to Processes and Threads

- **Definition**: A process is an independent program in execution with its own memory space, while a thread is a lightweight unit of execution within a process, sharing the process's memory.
- **Memory**: A process has its own isolated memory space (address space), whereas threads within the same process share the same memory space, including code, data, and resources.
- **Overhead**: Processes are heavier, requiring more system resources and time for creation and context switching, while threads are lighter, with lower overhead for creation and switching.
- **Communication**: Inter-process communication (IPC) is complex and slower (e.g., pipes, sockets), while threads communicate faster via shared memory but require synchronization (e.g., locks).

Context switching

Observations

1. Threads share the same address space. Thread context switching is much faster than process context switching:
    1. only registers and program counter need to be saved and restored
2. Process context switching is more expensive (in time and space) as
    1. TLB needs to be flushed
    2. page table is reloaded
    3. address space changes
3. Creating and destroying threads is much cheaper than doing so for processes.

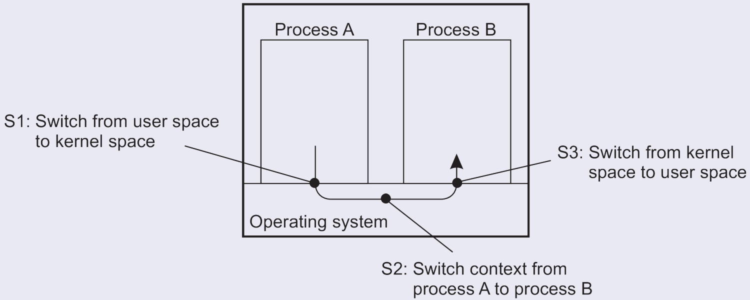
Why use threads

Some simple reasons

- Avoid needless blocking: a single-threaded process will block when doing I/O; in a multithreaded process, the operating system can switch the CPU to another thread in that process.
- Exploit parallelism: the threads in a multithreaded process can be scheduled to run in parallel on a multiprocessor or multicore processor.
- Avoid process switching: structure large applications not as a collection of processes, but through multiple threads.

Avoid process switching

Avoid expensive context switching



Trade-offs

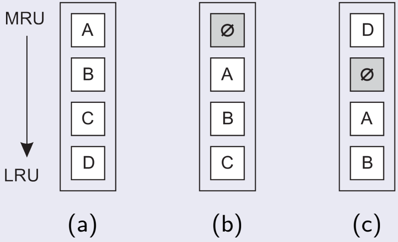
- Threads use the same address space: more prone to errors
- No support from OS/HW to protect threads using each other's memory
- Thread context switching may be faster than process context switching

The cost of a context switch

Consider a simple clock-interrupt handler

- direct costs: actual switch and executing code of the handler
- indirect costs: other costs, notably caused by messing up the cache

What a context switch may cause: indirect costs



(a) before the context switch
(b) after the context switch
(c) after accessing block D.

Threads and operating systems

Main issue

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

User-space solution

- All operations can be completely handled within a single process $\Rightarrow$ implementations can be extremely efficient.
- All services provided by the kernel are done on behalf of the process in which a thread resides $\Rightarrow$ if the kernel decides to block a thread, the entire process will be blocked.
- Threads are used when there are many external events: threads block on a per-event basis $\Rightarrow$ if the kernel can't distinguish threads, how can it support signaling events to them?

Linux Kernel Threads

- **Task Struct Representation**: In the Linux kernel, threads are implemented as lightweight processes, each represented by a `task_struct` (defined in `include/linux/sched.h`), sharing memory but maintaining separate execution contexts for scheduling.
- **Scheduling with CFS**: The Completely Fair Scheduler (CFS) in `kernel/sched/fair.c` manages kernel threads, treating them as virtual processors and allocating CPU time fairly using a red-black tree.
- **POSIX Threads Integration**: User-space threads (e.g., via `pthread_create`) map to kernel threads, enabling Java's 1:1 threading model to leverage Linux's scheduling for efficient concurrency.

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.
- Note: if calls are to different servers, we may have a linear speed-up.

Multithreaded clients: does it help?

Thread-level parallelism: TLP

Let c_i denote the fraction of time that exactly i threads are being executed simultaneously.

$$TLP = \frac{\sum_{i=1}^{N} i \cdot c_i}{1 - c_0}$$

with N the maximum number of threads that (can) execute at the same time.

Practical measurements

A typical Web browser has a TLP value between 1.5 and 2.5 $\Rightarrow$ threads are primarily used for logically organizing browsers.

Using threads at the server side

Improve performance

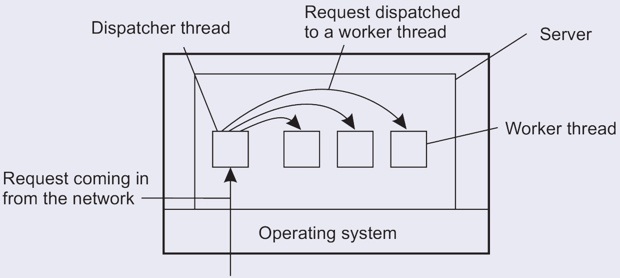
- Starting a thread is cheaper than starting a new process.
- Having a single-threaded server prohibits simple scale-up to a multiprocessor system.
- As with clients: hide network latency by reacting to next request while previous one is being replied.

Better structure

- Most servers have high I/O demands. Using simple, well-understood blocking calls simplifies the structure.
- Multithreaded programs tend to be smaller and easier to understand due to simplified flow of control.

Why multithreading is popular: organization

Dispatcher/worker model



Overview

Multithreading	Parallelism, blocking system calls
Single-threaded process	No parallelism, blocking system calls

Virtualization

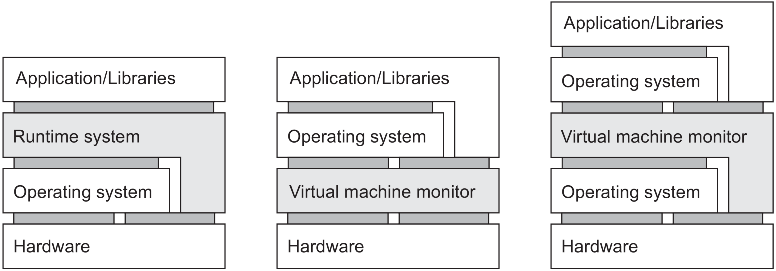
Virtualization

- Equivalence: Virtual machines (VMs) provide an environment identical to physical hardware, ensuring software runs unmodified.
- Resource Control: The virtual machine monitor (VMM) maintains full control over system resources, preventing VMs from accessing unauthorized resources.
- Efficiency: Most instructions execute directly on hardware for near-native performance, with sensitive instructions trapped by the VMM.
- Isolation: VMs are fully isolated, ensuring no interference or data leaks between VMs or the VMM, using mechanisms like memory and I/O virtualization.

Privileged instructions

Privileged instructions: allowed to be executed only by the kernel.

Ways of virtualization



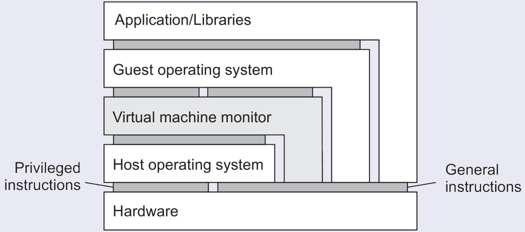
(a) Process VM (b) Native VMM (c) Hosted VMM

Differences

(a) Separate set of instructions, an interpreter/emulator, running atop an OS.
(b) Low-level instructions, along with bare-bones minimal operating system
(c) Low-level instructions, but delegating most work to a full-fledged OS.

Zooming into VMs: performance

Refining the organization



- Privileged instruction: if and only if executed in user mode, it causes a trap to the operating system
- Nonpriviliged instruction: the rest

Special instructions

- Control-sensitive instruction: may affect configuration of a machine (e.g., one affecting relocation register or interrupt table).
- Behavior-sensitive instruction: effect is partially determined by context (e.g., POPF sets an interrupt-enabled flag, but only in system mode).

Condition for virtualization

Necessary condition

For any conventional computer, a virtual machine monitor may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions.

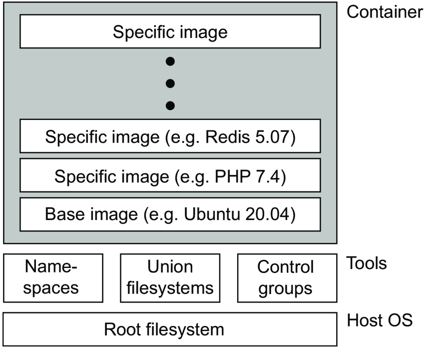
Problem: condition is not always satisfied

There may be sensitive instructions that are executed in user mode without causing a trap to the operating system.

Solutions

- Emulate all instructions
- Wrap nonprivileged sensitive instructions to divert control to VMM
- Paravirtualization: modify guest OS, either by preventing nonprivileged sensitive instructions, or making them nonsensitive (i.e., changing the context).

Containers



- Namespaces: a collection of processes in a container is given their own view of identifiers
- Union file system: combine several file systems into a layered fashion with only the highest layer allowing for `write` operations (and the one being part of a container).
- Control groups: resource restrictions can be imposed upon a collection of processes.

VMs and cloud computing

Three types of cloud services

- Infrastructure-as-a-Service covering the basic infrastructure
- Platform-as-a-Service covering system-level services
- Software-as-a-Service containing actual applications

IaaS

Instead of renting out a physical machine, a cloud provider will rent out a VM (or VMM) that may be sharing a physical machine with other customers $\Rightarrow$ almost complete isolation between customers (although performance isolation may not be reached).

Summary

Summary and Conclusions

We have discussed processes and threads in Distributed Systems, namely:

- Processes and Threads
- Context Switching
- Multithreading
- Virtualization
- Containerization