CSE 451: Operating Systems

Spring 2022

Module 9

A Kind of Review

John Zahorjan

Basic Questions

1. Why have an operating system?

2. What is/are the primary function(s) of the OS?

3. What hardware support is “required” to run an OS?

4. Is the compiler/language part of the OS? OS aware?

5. What OS functionality has to be part of the kernel?
   What OS functionality doesn’t?

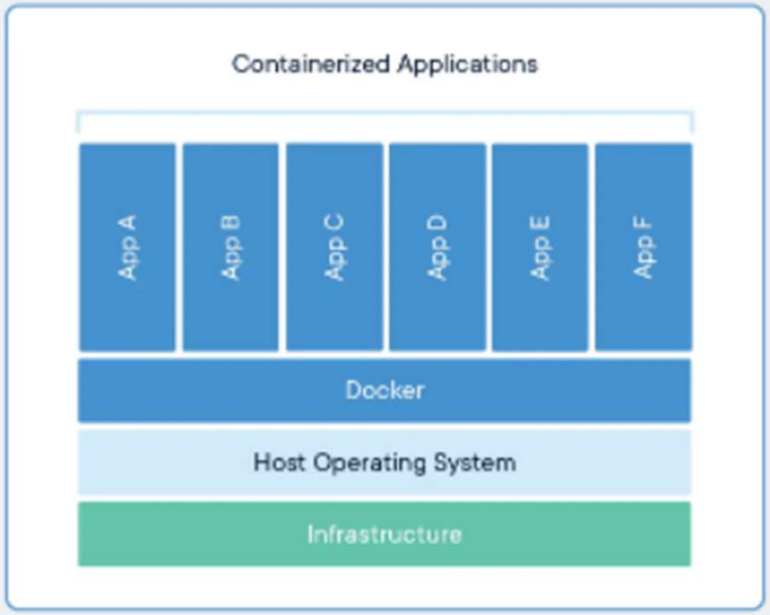
Execution Environments

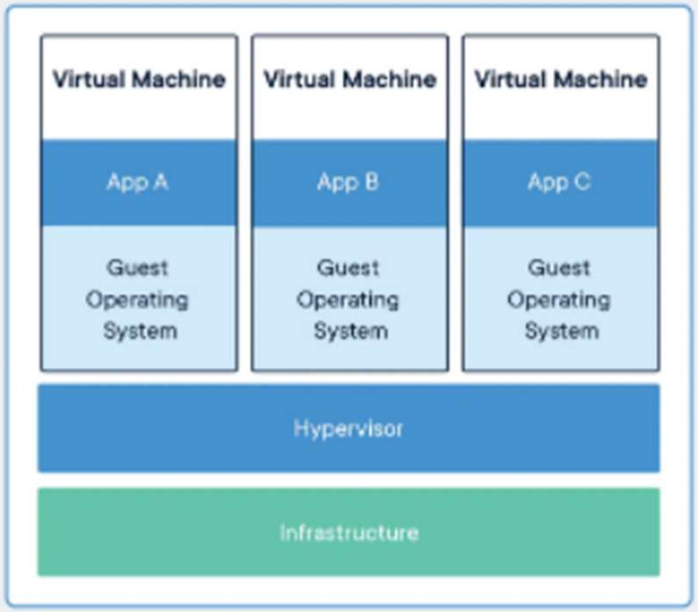
- What is the process concept?
- What are the/some major functions the OS has to implement to provide processes?
- What aspects of the computer system is a process isolated from? What aspects of the computer system isn’t it isolated from?
- What facilities does the OS provide to get around the isolation?
- Is there any alternative to “process”?

Comparing Containers and Virtual Machines

Containers and virtual machines have similar resource isolation and allocation benefits, but function differently because containers virtualize the operating system instead of hardware. Containers are more portable and efficient.

https://www.docker.com/resources/what-container/





CONTAINERS

Containers are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space. Containers take up less space than VMs (container images are typically tens of MBs in size), can handle more applications and require fewer VMs and Operating systems.

VIRTUAL MACHINES

Virtual machines (VMs) are an abstraction of physical hardware turning one server into many servers. The hypervisor allows multiple VMs to run on a single machine. Each VM includes a full copy of an operating system, the application, necessary binaries and libraries – taking up tens of GBs. VMs can also be slow to boot.

Concurrency

- Why might a programmer want to express an individual application as a set of concurrent control flows?
- What mechanisms are available to the programmer to express those concurrent control flows?
- Why isn’t just one mechanism enough?
- Why might “the system” want its workload expressed as concurrent control flows?

Sharing the CPU

- Why multiplex a single core across multiple control flows? Why not just assign each control flow its own core?
- Is specialized hardware support required to multiplex a core?
  - If so, what is that support and what can be achieved with that support that couldn't be achieved without it?
  - If not, could better performance or more robust functionality be achievable if there were specialized support?

Sharing the CPU

- When an execution stream cannot make use of a core right now, what should it do?
- How can it do that?
- When it does that, how does it stop doing it?

Concurrency: Synchronization

- What does “simultaneous” mean?
- How do you prevent two instructions from being simultaneous?
- What is a critical section?
- What properties must a mutual exclusion solution have?
- Why is it easier to implement mutual exclusion on a one core system (like xk) than on a multi-core system?
- Why “must” there be hardware support to implement fast mutual exclusion mechanisms?
  - What is the basic issue?
  - What are some example solutions?

Concurrency: granularity

- What is the idea of “granularity”?
- What is the impact of granularity?
- What is a (canonical) example of a granularity decision?
- How does the web server implementation you did in 332 relate to granularity?

Architecture: Main Memory Caches

- Why are there main memory caches?
- Why do they operate “transparently” (to program execution)?
  - Why not put them under program control?
- Can you think of (an) examples of how caches affect implementation (related to this course)?

Architecture: Multicores

- What is sequential consistency?
- Is it achievable on single core systems?
- Why isn’t it achievable (in any practical way) on multicore systems?

Architecture: Multicores

- What is cache consistency?
- What is memory consistency?
- How do you deal with memory consistency that is weaker than sequential consistency?
- What hardware support is used to deal with the issue?

Dependences

- Why does the compiler want to rewrite (reorder the instructions of) your program?
- Why does the hardware want to execute instructions out of program order?
- Which reorderings are considered legal?

Virtual Memory

- What is address translation?
    - Why is it useful?
    - What’s another reason?
    - And another reason?
- What is paging?
    - Why is it useful?
- True or False:
  A program execution that exhibits good page locality will exhibit good cache locality

Mechanism vs. Policy

- What is the distinction between mechanism and policy?

- There is a potential policy decision whenever the code has to make a choice

- Have you seen any policies implemented in xk?
    - What seem like situations in which you might want to implement a policy?

Deferring Policy

- Sometimes (often) you can parameterize execution
    - You pass in arguments that select from among pre-programmed alternatives
- The most general scheme is to defer decisions to code from the layer above
    - Feels like exception handling
        - Upper layer registers a method to be invoked when a decision has to be made
        - Lower layer notices when the decision has to be made and does an “upcall” to the handler
    - Signals
    - How might you apply that approach to paging?

Synchronization Primitives and Policy

- Locks
  - spin locks
  - blocking locks (mutexes)
  - spin then block locks
  - readers-writers locks
  - MCS locks
  - [RCU locks]

- Priority inversion and locks

Handling Latency

- What is “latency”?
- What are the approaches for dealing with it?
  - Concurrency
  - Caching
  - Batching
  - Speculation
  - Lazy evaluation
- Have we seen a situation in this course where “the OS” might confront latency and be able to do something about it?