EEL-4736/5737
Principles of Computer System Design

Lecture Slides 8

Textbook Chapter 5

Client-service Organization Using Virtualization

Introduction

- Client-service model enforces modularity
- However, the assumption of separate physical entities would limit applicability
- In practice, one needs to support various modules in the same computer
- We will study virtualization-based approaches to accomplish this

Virtualization – core ideas

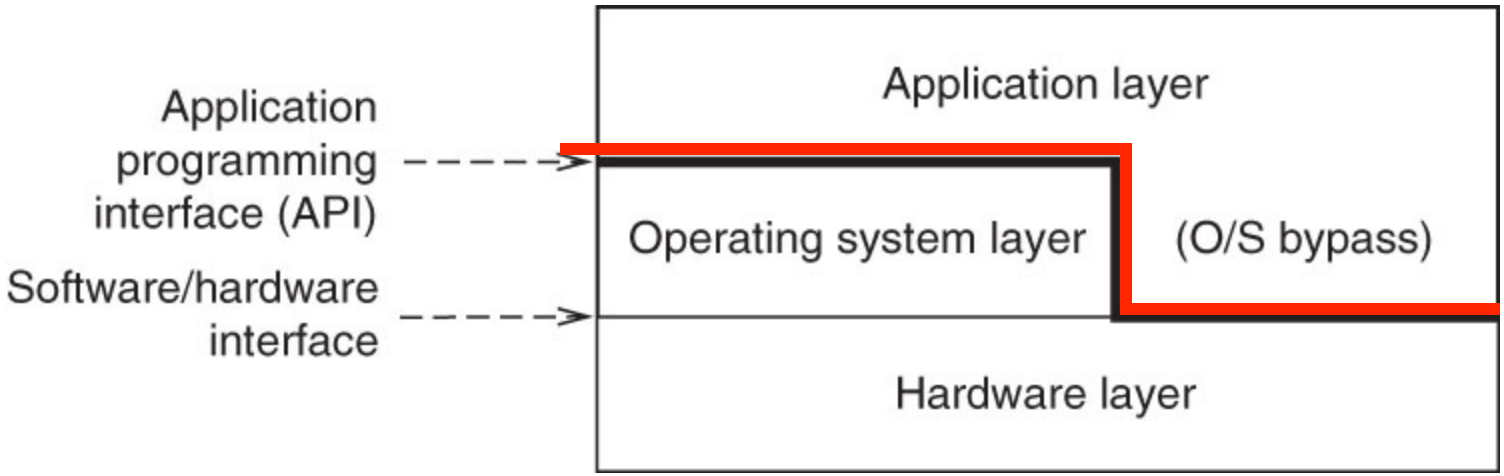
- *Primary goal: preserve an existing interface*
  - Such that existing modules designed to use an object do not need to be modified
- Virtualizing a physical system
  - Enables the creation of *multiple instances* of the system
  - Enables their *multiplexing* onto a physical system, or
  - Provide one large virtual system by *aggregating* many instances

Examples

Virtualization method	Physical resource	Virtual resource
Multiplexing	Server	Web site
Multiplexing	Processor	Thread
Multiplexing	Real memory	Virtual memory
Multiplexing and emulation	Real memory and disk	Virtual memory with paging
Multiplexing	Communication channel	Virtual circuit
Aggregation	Communication channel	Channel bonding
Aggregation	Disk	RAID
Emulation	Memory	RAM disk
Emulation	PowerPC Mac	x86 Virtual PC

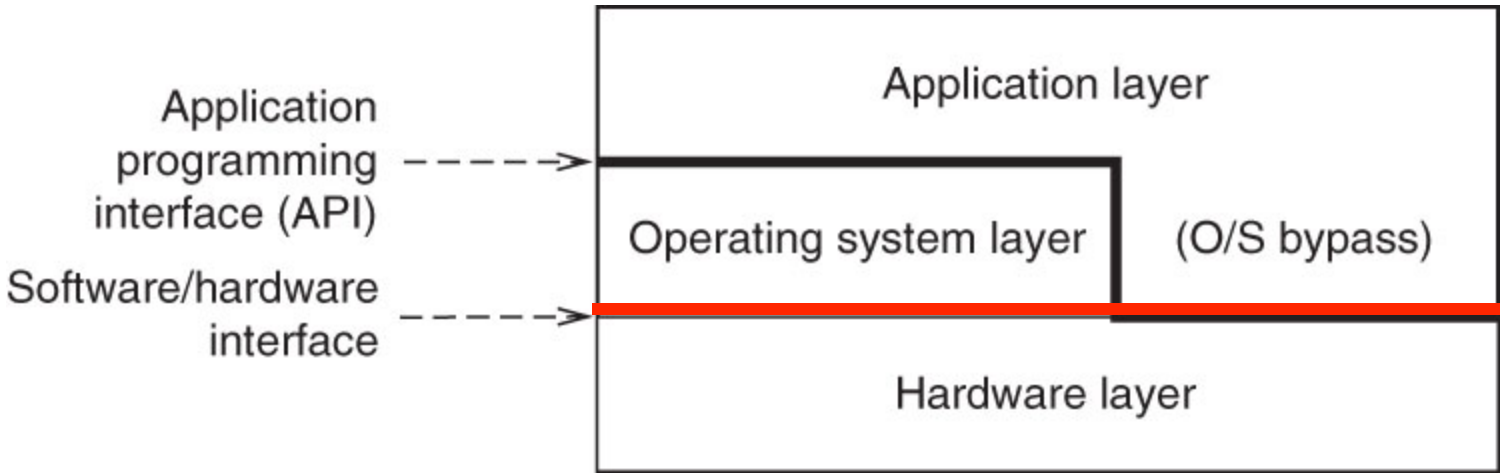
Examples

- Threads/processes in typical O/S
  - Present a “virtual processor” that provides the “application binary interface” (O/S bypass and system calls)



Examples

- Virtual machines - VMware/VirtualBox
  - Present a “virtual processor” that provides the instruction set architecture interface



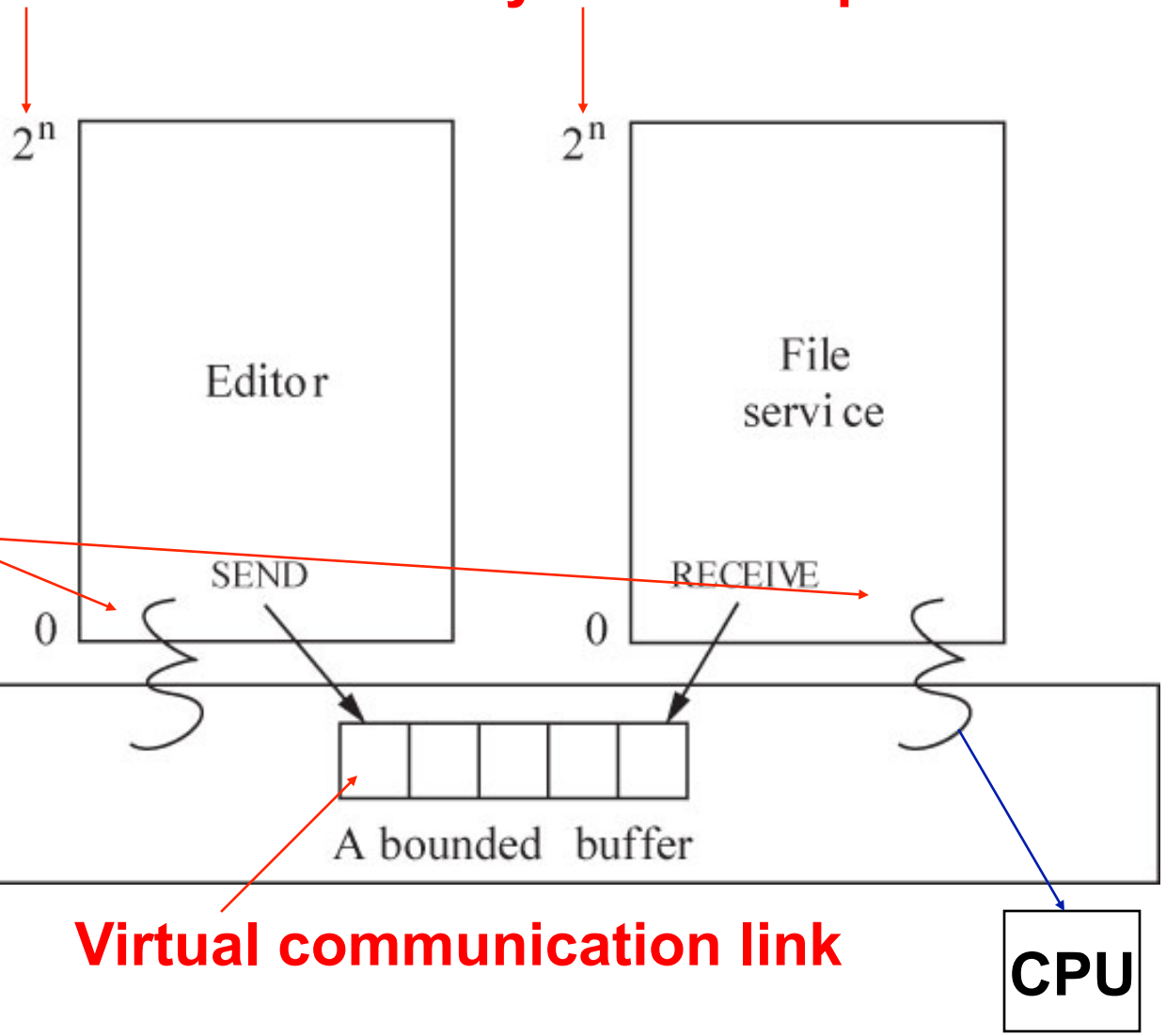
Abstractions

- Recall our three fundamental abstractions
  - Interpreter
  - Memory
  - Communication links
- Virtualization
  - Threads of execution
  - Virtual memory
  - SEND/RECEIVE with bounded buffers
- Operating systems implement these

Abstractions - example

Phy Mem

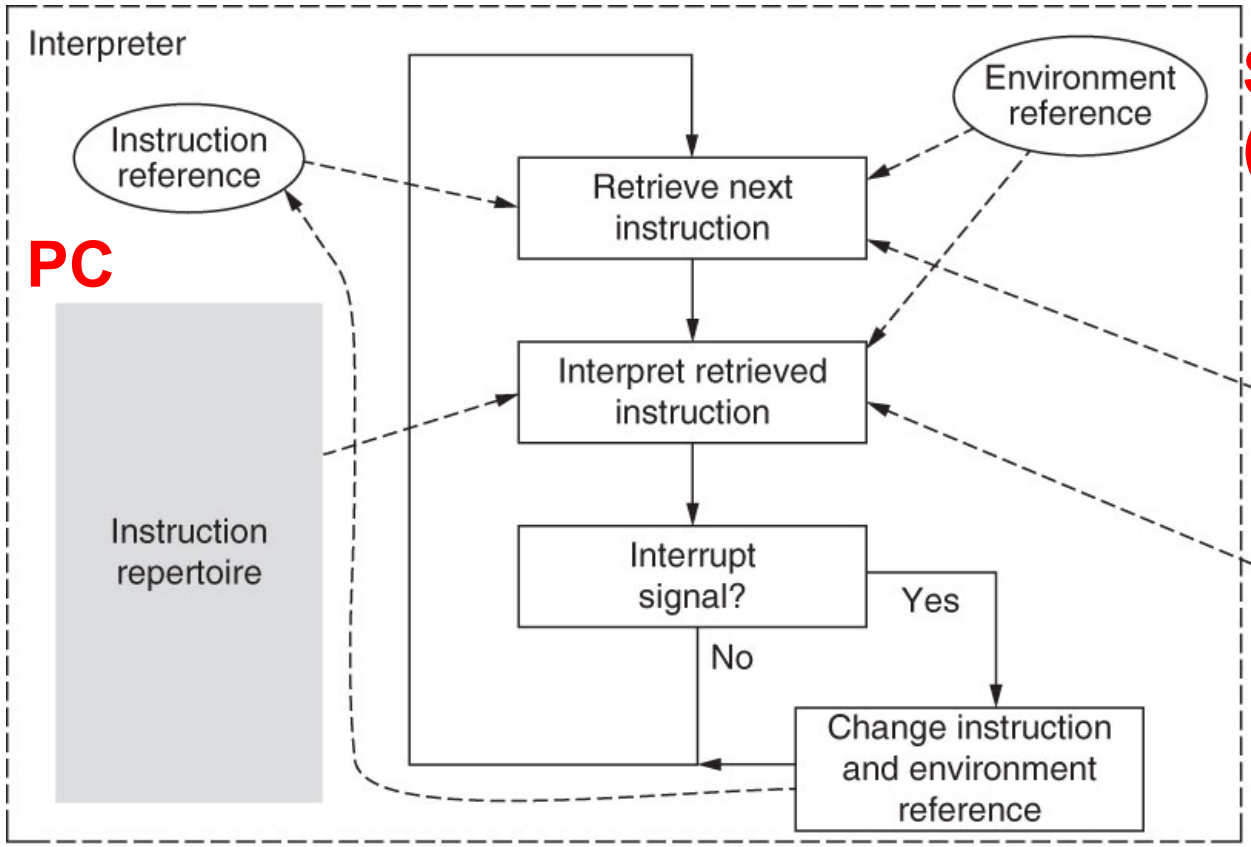
Virtual memory address spaces



Thread of Execution

- Encapsulates the execution state of an active computation
  - The state of an interpreter that executes the computation
  - Variables/state internal to the interpreter
- In a microprocessor – subset of processor registers
  - Reference to next instruction – PC
  - Reference to environment – e.g. stack, memory heap, general purpose registers

Interpreter abstraction



Stack pointer (Page table)

PC

```
1    procedure INTERPRET()
2        do forever
3            instruction ← READ (instruction_reference)
4            perform instruction in the context of environment_reference
5            if interrupt_signal = TRUE then
6                instruction_reference ← entry point of INTERRUPT_HANDLER
7                environment_reference ← environment ref of INTERRUPT_HANDLER
```

Threads

- Encapsulates state such that:
  - A thread can be stopped and its state can be saved
  - A thread can be restarted from saved state
  - Efficiently doing so is key to achieving *multiplexing* – time-sharing
- A thread has single control flow – i.e. execution within a thread is *sequential*
  - Multiple threads can execute concurrently with respect to each other – within or across modules

Thread manager

- Typically there are more threads than physical processors
  - Some module needs to manage which threads run and which wait
- Recall interpreter abstraction, interrupts
  - Provides a key mechanism for thread manager to “kick in” and make scheduling decisions
  - Manager itself is a thread – will see how to enforce different privileges later

Virtual memory

- Threads sharing memory in same name space would not preserve the illusion of a processor with its own memory
  - Sharing is useful, but *uncontrolled sharing* would defeat modularity
  - E.g. an error in a thread could cause it to overwrite the value of stack of another thread
- Soft modularity not sufficient
  - Virtual memory enforces modularity in memory, preserving the same interface of physical memory
    - And more

Virtual memory

- Each module (e.g. a thread or collection of threads) is given its own name space (*address space)*
- Any reference by an instruction becomes a reference in the context of this address space
- Separation among address spaces is enforced by *virtual memory manager*
  - Sharing can occur if agreed upon and programmed explicitly
- A thread and its address space: often referred to as “process”

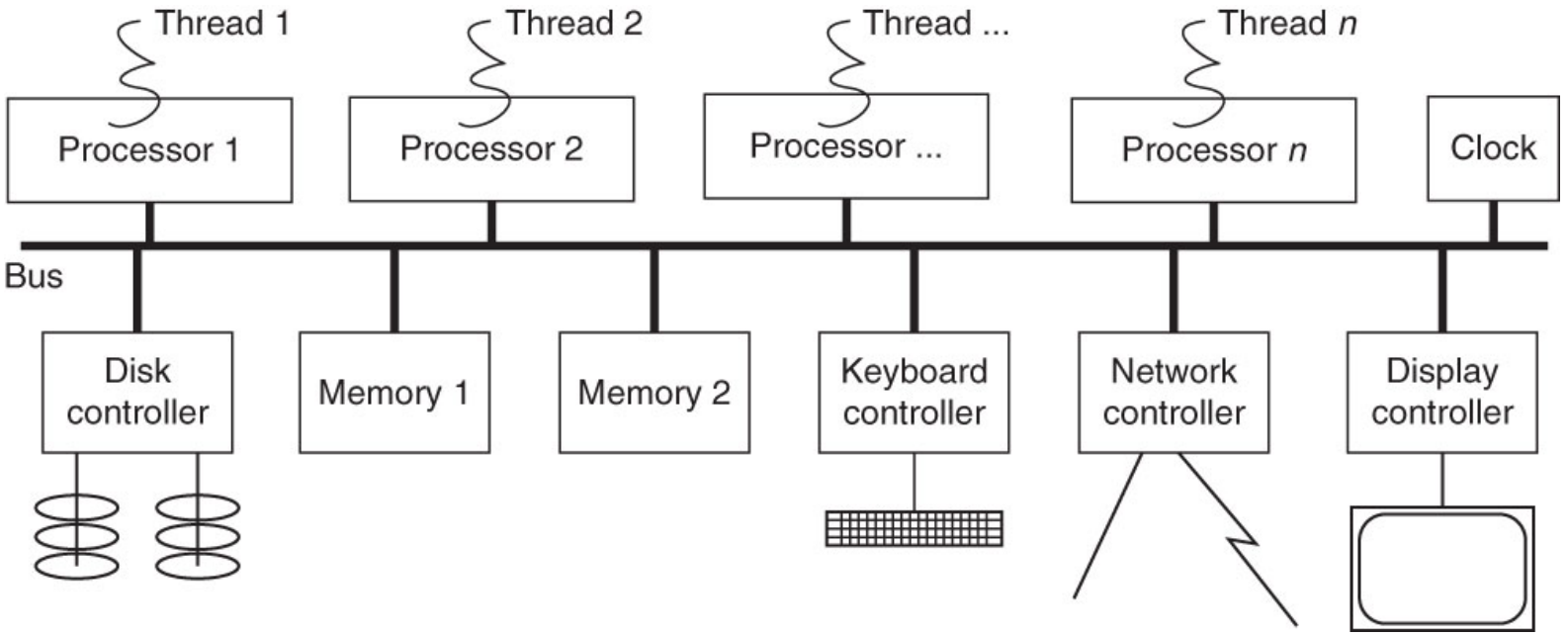
Bounded buffer

- Supports the SEND/RECEIVE primitives we have seen earlier
- The communication link has a bounded buffer:
  - SENDing thread blocks if buffer is full
    - Thread waits until there is room in buffer
  - Receiving thread invokes RECEIVE to retrieve a message from the buffer
    - If there are no messages, wait

Roadmap

- We will focus on core interfaces typically found in the “virtual managers” (O/S kernel, virtual machine monitor)
  - Common but not particular to any implementation
- Initially, will assume more processors than threads, and single address space
- Progressively removing restrictions: separate address spaces, processor multiplexing

Initial context



Core O/S Interface

- Memory abstraction:
  - CREATE_ADDRESS_SPACE
  - DELETE_ADDRESS_SPACE
  - ALLOCATE_BLOCK
  - FREE_BLOCK
  - MAP
  - UNMAP

Core O/S Interface

- Interpreter:
  - ALLOCATE_THREAD
  - EXIT_THREAD
  - DESTROY_THREAD
  - YIELD
  - AWAIT
  - ADVANCE
  - TICKET
  - ACQUIRE
  - RELEASE

Core O/S Interface

- Communication:
  - ALLOCATE_BOUNDED_BUFFER
  - DEALLOCATE_BOUNDED_BUFFER
  - SEND
  - RECEIVE

Reading

- Sections 5.2-5.4