CSE 560
Computer Systems Architecture

Virtual Memory

1

This Unit: Virtual Memory

App App App
System software
Mem CPU I/O
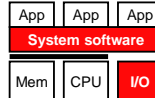
- The operating system (OS)
  - A super-application
  - Hardware support for an OS
- Virtual memory
  - Page tables and address translation
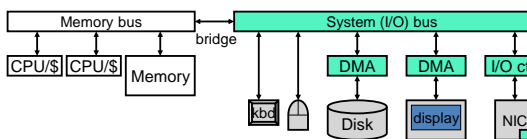 - TLBs and memory hierarchy issues

2

A Computer System: Hardware

CPUs and memories
- Connected by memory bus

I/O peripherals: storage, input, display, network, …
              (NIC = Network Interface Controller)
- With separate or built-in DMA (direct memory access)
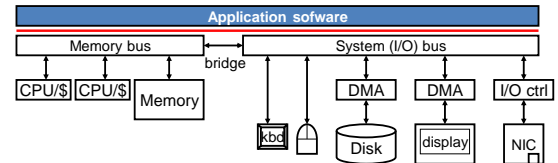- Connected by **system bus** (which is connected to memory bus)

3

A Computer System: + App Software

- **Application software**: computer must do something

4

A Computer System: + OS

Operating System (OS): virtualizes hardware for apps
- **Abstraction**: provides **services** (*e.g.*, threads, files, *etc.*)
  + Simplifies app programming model, raw hardware is nasty
- **Isolation**: gives each app illusion of private CPU, memory, I/O
  + Simplifies app programming model
  + Increases hardware resource utilization
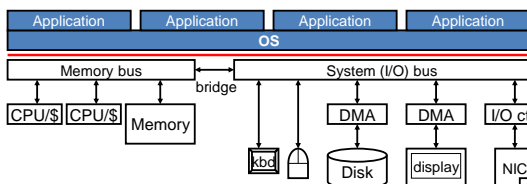
5

Operating System (OS) and User Apps

- Sane system development requires a split

- **Operating System (OS)**: a super-privileged process
  - Manages hw resource allocation/revocation for all processes
  - Has direct access to resource allocation features
  - **Aware of:** many nasty hardware details, other processes
  - Talks directly to input/output devices (device driver software)

- **User-level apps**: ignorance is bliss
  - **Unaware of:** most nasty hardware details, other apps, OS
  - Explicitly denied access to resource allocation features

6

System Calls

System Call: a user-level app "function call" to OS
- Leave description of what you want done in registers
- SYSCALL instruction (also called TRAP or INT)
  - User-level apps not allowed to invoke arbitrary OS code
  - Restricted set of legal OS addresses to jump to (**trap vector**)

1. Processor jumps to OS via trap vector (begin privileged mode)
2. OS performs operation
3. OS does a "return from system call" (end privileged mode)

7

Interrupts

Exceptions: synchronous, generated by running app
- *E.g.*, illegal instruction, divide by zero, *etc.*

Interrupts: asynchronous events generated externally
- *E.g.*, timer, I/O request/reply, *etc.*

Timer: programmable on-chip interrupt
- Initialize with some number of micro-seconds
- Timer counts down and interrupts when reaches 0

"Interrupt" handling: same mechanism for both
- "Interrupts" are on-chip signals/bits
  - Either internal (*e.g.*, timer, exceptions) or from I/O devices
- Processor continuously monitors interrupt status, when true…
- HW jumps to some preset address in OS code (interrupt vector)
- Like an asynchronous, non-programmatic SYSCALL

8

Virtualizing Processors

How do multiple apps (and OS) share the processors?
Goal: applications think there are an infinite # of processors

Solution: time-share the resource
- Trigger a **context switch** at a regular interval (~1ms)
  - **Pre-emptive**: app doesn't yield CPU, OS forcibly takes it
    + Stops greedy apps from starving others
- **Architected state**: PC, registers
  - Save and restore them on context switches
  - Memory state?
- **Non-architected state**: caches, predictor tables, *etc.*
  - Ignore or flush
- Operating System responsible for handling context switching
  - Hardware support is just a timer interrupt

9

Motivations for Virtual Memory

- **Use Physical DRAM as a Cache for the Disk**
  - Address space of a process can exceed physical memory size
  - Sum of address spaces of multiple processes can exceed physical memory
- **Simplify Memory Management**
  - Multiple processes resident in main memory
    - Each process with its own address space
  - Only "active" code and data is actually in memory
    - Allocate more memory to process as needed
- **Provide Protection**
  - One process can't interfere with another
    - because they operate in different address spaces
  - User process cannot access privileged information
    - different sections of address spaces have different permissions

10

Levels in Memory Hierarchy
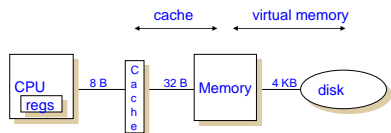
	Register	Cache	Memory	Disk Memory
size:	32 B	32 KB-4MB	1024 MB	100 GB
speed:	1 ns	2 ns	30 ns	8 ms
$/Mbyte:		$125/MB	$0.20/MB	$0.001/MB
line size:	8 B	32 B	4 KB	

larger, slower, cheaper

11

Virtualizing Main Memory

How do multiple apps (and the OS) share main memory?
Goal: each application thinks it has private memory

App's insn/data footprint > main memory ?
- **Requires main memory to act like a cache**
  - With disk as next level in memory hierarchy (slow)
  - Write-back, write-allocate, large blocks or "pages"

Solution:
- Part #1: treat memory as a "cache"
- Part #2: add a level of indirection (address translation)

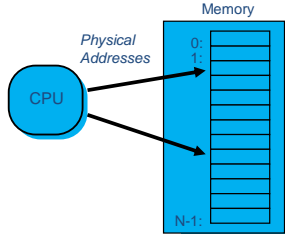| Parameter	I$/D$	L2	Main Memory
t_{hit}	2ns	10ns	**30ns**
t_{miss}	**10ns**	**30ns**	**10ms (10M ns)**
Capacity	8–64KB	128KB–2MB	**64MB–64GB**
Block size	16–32B	32–256B	**4+KB**
Assoc./Repl.	1–4, NMRU	4–16, NMRU	**Full, "working set"**

12

A System with Physical Memory Only

Examples:
- most Cray machines, early PCs, many embedded systems, etc.
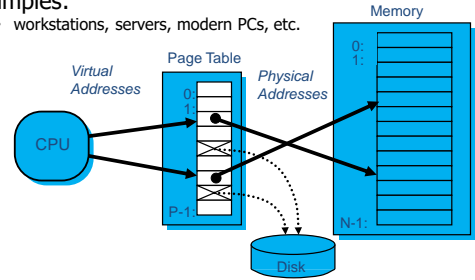

Addresses generated by the CPU correspond directly to bytes in physical memory

13

A System with Virtual Memory

Examples:
- workstations, servers, modern PCs, etc.

Address Translation: Hardware converts virtual addresses to physical addresses via OS-managed lookup table (page table)
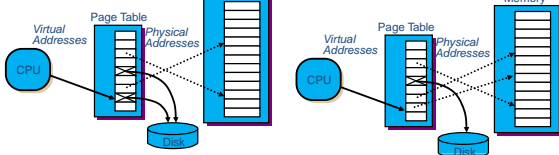
14

Page Faults (like "Cache Misses")

- What if an object is on disk rather than in memory?
  - Page table entry indicates virtual address not in memory
  - OS exception handler invoked to move data from disk into memory
    - current process suspends, others can resume
    - OS has full control over placement, etc.

Before fault


After fault

15

Virtual Memory (VM)

Program

- Programs use **virtual addresses (VA)**
  - $0...2^N-1$
  - VA size also referred to as machine size
  - *E.g.*, 32-bit (embedded) or 64-bit (server)
- Memory uses **physical addresses (PA)**
  - $0...2^M-1$ (typically M<N, especially if N=64)
  - 2^M is most physical memory machine supports
- VA→PA at **page** granularity (VP→PP)
  - By "system" (OS + HW)
  - Mapping need not preserve contiguity
  - VP need not be mapped to any PP
  - Unmapped VPs live on disk (swap)

16

16

Virtual Memory (VM)

Virtual Memory (VM):
- Level of indirection
- Application generated addresses are **virtual addresses (VAs)**
  - Each process *thinks* it has its own 2^N bytes of address space
- Memory accessed using **physical addresses (PAs)**
- VAs translated to PAs at some coarse granularity
- OS controls VA to PA mapping for itself and all other processes
- Logically: translation performed before every insn fetch, load, store
- Physically: hardware acceleration removes translation overhead

OS App1 App2 VAs

OS controlled VA→PA mappings

PAs (physical memory)
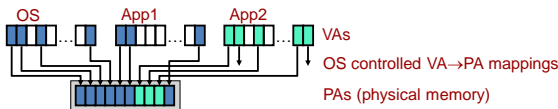
17

17

Uses of Virtual Memory

- **Isolation** and **Multi-programming (Memory Management)**
  - Each app thinks it has 2^N B of memory that starts @ 0
  - Apps can't read/write each other's memory
    - Can't even address the other program's memory!
- **Protection**
  - Each page has read/write/execute permission set by OS
  - Enforced by hardware
- **Inter-process communication**
  - Map same physical pages into multiple virtual address spaces
  - Or share files via the UNIX `mmap()` call
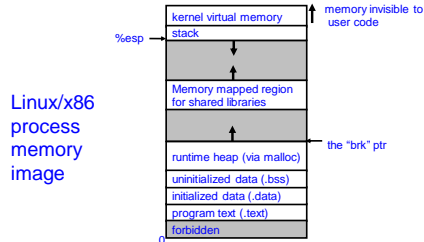

OS App1 App2
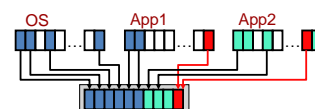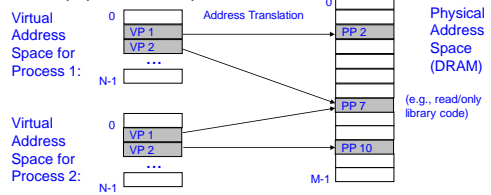
18

18

Memory Management

- Multiple processes can reside in physical memory.
- How do we resolve address conflicts?
  - what if two processes access something at the same address?

Linux/x86 process memory image

kernel virtual memory	memory invisible to user code
stack	
Memory mapped region for shared libraries	
runtime heap (via malloc)	the "brk" ptr
uninitialized data (.bss)	
initialized data (.data)	
program text (.text)	
forbidden	

%esp (points to stack)

0

19

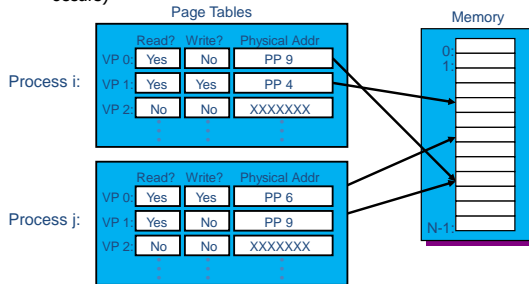## Solution: Separate Virt. Addr. Spaces

- Virtual and physical address spaces divided into equal-sized blocks
  - blocks are called "pages" (both virtual and physical)
- Each process has its own virtual address space
  - operating system controls how virtual pages as assigned to physical memory

Virtual Address Space for Process 1: 0 ... N-1
VP 1, VP 2 ...

Address Translation

Physical Address Space (DRAM) 0 ... M-1
PP 2
PP 7 (e.g., read/only library code)

Virtual Address Space for Process 2: 0 ... N-1
VP 1, VP 2 ...
PP 10

20

Protection

- Page table entry contains access rights information
  - hardware enforces this protection (trap into OS if violation occurs)

Page Tables

Memory

Process i:

	Read?	Write?	Physical Addr
VP 0:	Yes	No	PP 9
VP 1:	Yes	Yes	PP 4
VP 2:	No	No	XXXXXXX

Process j:

	Read?	Write?	Physical Addr
VP 0:	Yes	Yes	PP 6
VP 1:	Yes	No	PP 9
VP 2:	No	No	XXXXXXX

Memory: 0, 1, ... N-1

21