

CS162 Operating Systems and Systems Programming Lecture 2

Concurrency: Processes, Threads, and Address Spaces

August 29, 2012

Ion Stoica

<http://inst.eecs.berkeley.edu/~cs162>

Virtual Machines (Recap)

- Software emulation of an abstract machine
 - Give programs illusion they own the machine
 - Make it look like hardware has features you want
- Two types of VMs
 - System VM: supports the execution of an entire OS and its applications (e.g., VMWare Fusion, Parallels Desktop, Xen)
 - Process VM: supports the execution of a single program; this functionality is typically provided by OS



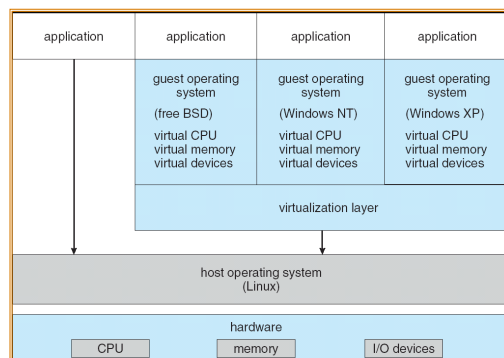
8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.2

System Virtual Machines: Layers of OSs

- Useful for OS development
 - When OS crashes, restricted to one VM
 - Can aid testing programs on other OSs



8/29/12

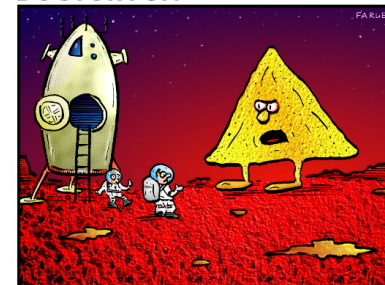
Ion Stoica CS162 ©UCB Fall 2012

Lec 1.3

Nachos: Virtual OS Environment

- You will be working with Nachos
 - Simulation environment: Hardware, interrupts, I/O
 - Execution of User Programs running on this platform
 - See the “Projects and Nachos” link off the course home page

DOCTOR FUN



"This is the planet where nachos rule."

8/29/12

Lec 1.4

What does an Operating System do?

- Silberschatz and Gavin: “An OS is Similar to a government”
 - Begs the question: does a government do anything useful by itself?
- OS as a Traffic Cop:
 - Manages all resources
 - Settles conflicting requests for resources
 - Prevent errors and improper use of the computer
- OS as a facilitator (“useful” abstractions):
 - Provides facilities/services that everyone needs
 - Standard libraries, windowing systems
 - Make application programming easier, faster, less error-prone
- Some features reflect both tasks:
 - File system is needed by everyone (Facilitator) ...
 - ... but File system must be protected (Traffic Cop)

8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.5

What is an Operating System,... Really?

- Most Likely:
 - Memory Management
 - I/O Management
 - CPU Scheduling
 - Synchronization / Mutual exclusion primitives
 - Communications? (Does Email belong in OS?)
 - Multitasking/multiprogramming
- What about?
 - File System?
 - Multimedia Support?
 - User Interface?
 - Internet Browser? ☺
- Is this only interesting to Academics??

8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.6

Operating System Definition (Cont'd)

- No universally accepted definition
- “*Everything a vendor ships when you order an operating system*” is good approximation
 - But varies wildly
- “*The one program running at all times on the computer*” is the **kernel**
 - Everything else is either a system program (ships with the operating system) or an application program

8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.7

Very Brief History of OS

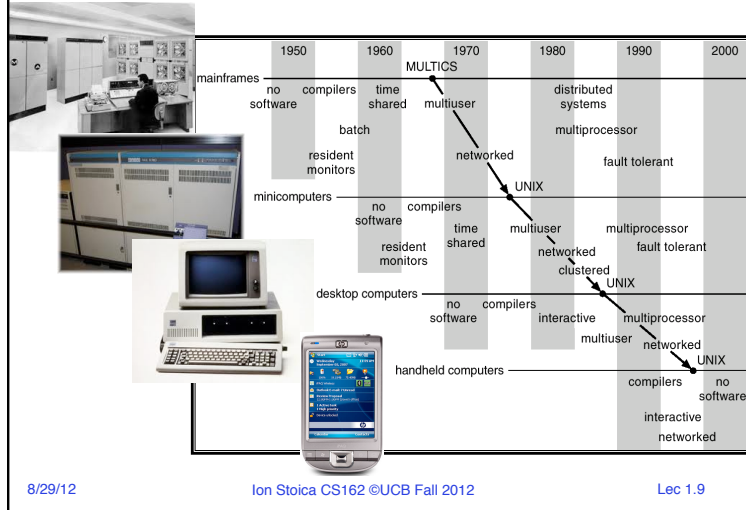
- Several Distinct Phases:
 - Hardware Expensive, Humans Cheap
 - » Eniac, ... Multics
 - Hardware Cheaper, Humans Expensive
 - » PCs, Workstations, Rise of GUIs
 - Hardware Really Cheap, Humans Really Expensive
 - » Ubiquitous devices, Widespread networking
- Rapid Change in Hardware Leads to changing OS
 - Batch ⇒ Multiprogramming ⇒ Timesharing ⇒ Graphical UI ⇒ Ubiquitous Devices
 - Gradual Migration of Features into Smaller Machines
- Situation today is much like the late 60s
 - Small OS: 100K lines/Large: 10M lines (5M browser!)
 - 100-1000 people-years

8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.8

Migration of OS Concepts and Features



Goals for Today

- How do we provide multiprogramming?
- What are **processes**?
- How are they related to **threads** and **address spaces**?

Note: Some slides and/or pictures in the following are adapted from slides ©2005 Silberschatz, Galvin, and Gagne. Slides courtesy of Anthony D. Joseph, John Kubiawicz, AJ Shankar, George Necula, Alex Aiken, Eric Brewer, Ras Bodik, Ion Stoica, Doug Tygar, and David Wagner.

8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.10

Threads

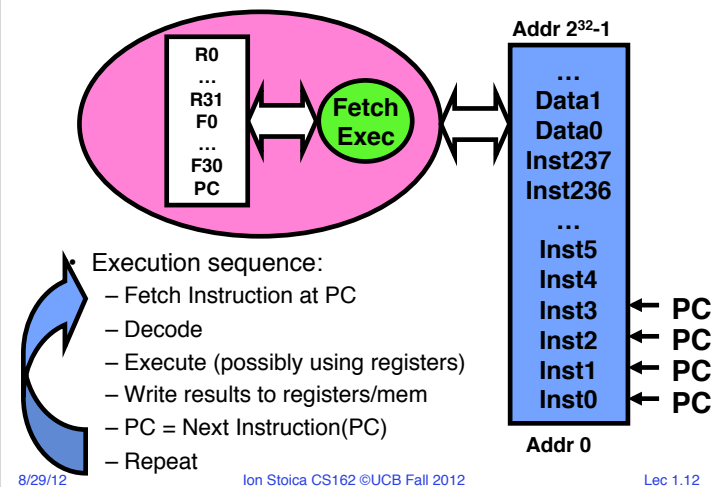
- Unit ("thread") of execution:
 - Independent Fetch/Decode/Execute loop
 - Unit of scheduling
 - Operating in some **address space**

8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.11

Recall (61C): What happens during execution?



Uniprograming vs. Multiprograming

- Uniprogramming: *one thread at a time*
 - MS/DOS, early Macintosh, batch processing
 - Easier for operating system builder
 - Get rid of concurrency (only one thread accessing resources!)
 - Does this make sense for personal computers?
- Multiprogramming: *more than one thread at a time*
 - Multics, UNIX/Linux, OS/2, Windows NT – 7, Mac OS X
 - Often called “multitasking”, but multitasking has other meanings (talk about this later)
- ManyCore ⇒ Multiprogramming, right?

8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.13

Challenges of Multiprograming

- Each application wants to own the machine → **virtual machine abstraction**
- Applications compete with each other for resources
 - Need to arbitrate access to shared resources → **concurrency**
 - Need to protect applications from each other → **protection**
- Applications need to communicate/cooperate with each other → **concurrency**

8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.14

Processes

- **Process:** unit of resource allocation **and** execution
 - Owns memory (address space)
 - Owns file descriptors, file system context, ...
 - Encapsulate one or more threads sharing process resources
- Why **processes**?
 - Navigate fundamental tradeoff between protection and efficiency
 - Processes provides memory protection while threads don't (share a process memory)
 - Threads more efficient than processes (later)
- Application instance consists of one or more processes

8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.15

The Basic Problem of Concurrency

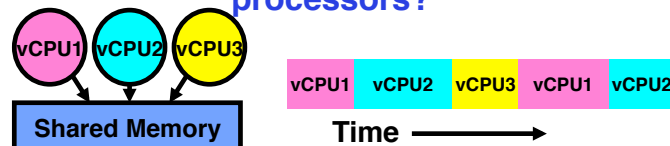
- The basic problem of concurrency involves resources:
 - Hardware: single CPU, single DRAM, single I/O devices
 - Multiprogramming API: processes think they have exclusive access to shared resources
- OS has to coordinate all activity
 - Multiple processes, I/O interrupts, ...
 - How can it keep all these things straight?
- Basic Idea: Use Virtual Machine abstraction
 - Simple machine abstraction for processes
 - Multiplex these abstract machines
- Dijkstra did this for the “THE system”
 - Few thousand lines vs 1 million lines in OS 360 (1K bugs)

8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.16

How can we give the illusion of multiple processors?



- Assume a single processor. How do we provide the illusion of multiple processors?
 - Multiplex in time!
- Each virtual “CPU” needs a structure to hold:
 - Program Counter (PC), Stack Pointer (SP)
 - Registers (Integer, Floating point, others...?)
- How switch from one virtual CPU to the next?
 - Save PC, SP, and registers in current state block
 - Load PC, SP, and registers from new state block
- What triggers switch?
 - Timer, voluntary yield, I/O, other things

8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.17

Properties of this simple multiprogramming technique

- All virtual CPUs share same non-CPU resources
 - I/O devices the same
 - Memory the same
- Consequence of sharing:
 - Each thread can access the data of every other thread (good for sharing, bad for protection)
 - Threads can share instructions (good for sharing, bad for protection)
 - Can threads overwrite OS functions?
- This (unprotected) model common in:
 - Embedded applications
 - Windows 3.1/Early Macintosh (switch only with yield)
 - Windows 95—ME (switch with both yield and timer)

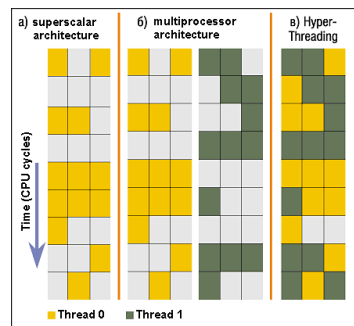
8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.18

Modern Technique: SMT/Hyperthreading

- Hardware technique
 - Exploit natural properties of superscalar processors to provide illusion of multiple processors
 - Need to replicate registers, but higher utilization of processor resources
- Can schedule each thread as if were separate CPU
 - But, non-linear speedup!
- Original technique called “Simultaneous Multithreading”
 - See <http://www.cs.washington.edu/research/smt/>
 - SPARC, Pentium 4/Xeon (“Hyperthreading”), Power 5



8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.19

How to protect threads from one another?

1. Protection of memory
 - Every thread does not have access to all memory
2. Protection of I/O devices
 - Every thread does not have access to every device
3. Protection of access to processor: preemptive switching from thread to thread
 - Use of timer
 - Must not be possible to disable timer from usercode

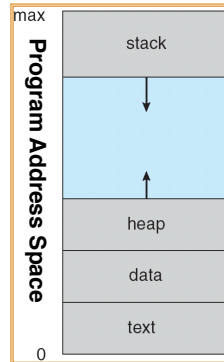
8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.20

Recall: Program's Address Space

- Address space \Rightarrow the set of accessible addresses + associated states:
 - For a 32-bit processor there are $2^{32} = 4$ billion addresses
- What happens when you read or write to an address?
 - Perhaps nothing
 - Perhaps acts like regular memory
 - Perhaps ignores writes
 - Perhaps causes I/O operation
 - » (Memory-mapped I/O)
 - Perhaps causes exception (fault)

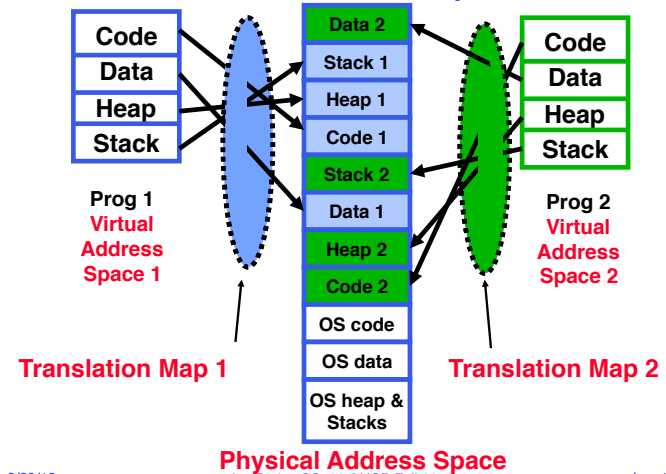


8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.21

Providing Illusion of Separate Address Space: Load new Translation Map on Switch



8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.22

Administrivia

- We are using Piazza instead of the newsgroup
 - Got to <http://www.piazza.com/berkeley/fall2012/cs162>
 - Make an account and join Berkeley, CS 162
 - Please ask questions on Piazza instead of emailing TAs
 - Only 133 enrolled, please enroll today!
- Already registered and need an account form?
 - See a TA after class or email cs162@cory
- Don't know Java well?
 - Take CS 9G self-paced Java course
- We may have pop quizzes...

8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.23

Administrivia: Project Signup

- Project Signup: Use "Group/Section Signup" Link
 - 4-5 members to a group, *everyone must attend the same section*
 - The sections assigned to you by Telebears are temporary!
 - Only submit once per group! **Due Monday (9/3) by 11:59PM**
 - Everyone in group must have logged into their cs162-xx accounts once before you register the group, *Select at least 2 potential sections*
- New section assignments: Watch "Group/Section Assignment" Link
 - Attend new sections NEXT week

Section	Time	Location	TA
101	Th 10:00A-11:00A	71 Evans	HY
102	Th 11:00A-12:00P	71 Evans	Edward
103	Th 1:00P-2:00P	87 Evans	HY
104	Th 2:00P-3:00P	87 Evans	Edward
105	Th 3:00P-4:00P	71 Evans	Patrick
106	Th 4:00P-5:00P	71 Evans	Patrick

8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.24

Administrivia: Projects

- First two projects are based on Nachos
 - Start reading walkthrough and code NOW
- Second two projects will add more OS and systems components: in-memory key-value store
 - Project 3: single server key-value store:
 - » PUT/GET RPCs, in-memory hash-table management
 - » EC2 deployments
 - Project 4: distributed key-value store:
 - » two phase commit for replication, data/communication encryption

8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.25

Administrivia: Laptop Policy

- Discussion sections: closed-laptop policy
- Lecture:
 - Closed laptops, **highly** preferred
 - If you really have to use a laptop, please stay in the back of the class (to minimize disruption)

8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.26

5min Break

8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.27

Traditional UNIX Process

- Process: *Operating system abstraction to represent what is needed to run a single program*
 - Often called a “HeavyWeight Process”
 - Formally: a single, sequential stream of execution in its own address space
- Two parts:
 - Sequential program execution stream
 - » Code executed as a *single, sequential* stream of execution (i.e., thread)
 - » Includes State of CPU registers
 - Protected resources:
 - » Main memory state (contents of Address Space)
 - » I/O state (i.e. file descriptors)
- Important: There is no concurrency in a heavyweight process

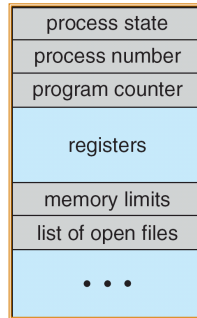
8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.28

How do we Multiplex Processes?

- The current state of process held in a process control block (PCB):
 - This is a “snapshot” of the execution and protection environment
 - Only one PCB active at a time
- Give out CPU time to different processes (**Scheduling**):
 - Only one process “running” at a time
 - Give more time to important processes
- Give pieces of resources to different processes (**Protection**):
 - Controlled access to non-CPU resources
 - Example mechanisms:
 - Memory Mapping: Give each process their own address space
 - Kernel/User duality: Arbitrary multiplexing of I/O through system calls



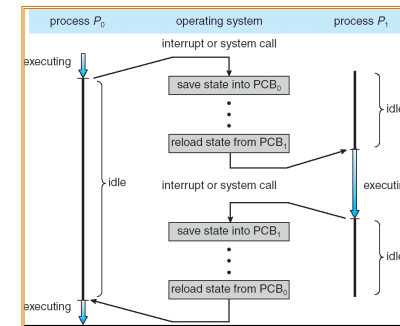
Process Control Block

8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.29

CPU Switch From Process to Process



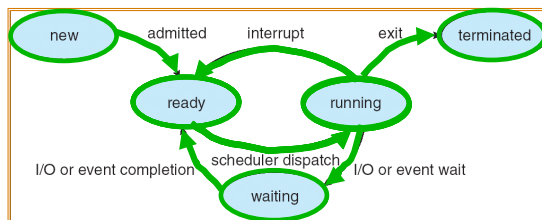
- This is also called a “context switch”
- Code executed in kernel above is overhead
 - Overhead sets minimum practical switching time
 - Less overhead with SMT/Hyperthreading, but... contention for resources instead

8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.30

Lifecycle of a Process



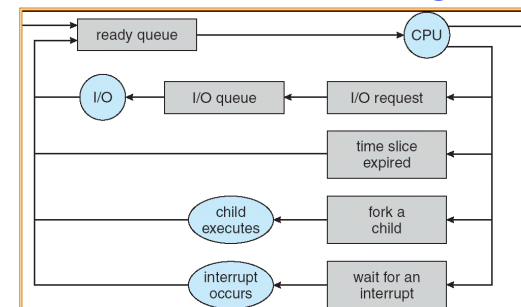
- As a process executes, it changes state:
 - new**: The process is being created
 - ready**: The process is waiting to run
 - running**: Instructions are being executed
 - waiting**: Process waiting for some event to occur
 - terminated**: The process has finished execution

8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.31

Process Scheduling



- PCBs move from queue to queue as they change state
 - Decisions about which order to remove from queues are **Scheduling** decisions
 - Many algorithms possible (few weeks from now)

8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.32

What does it take to create a process?

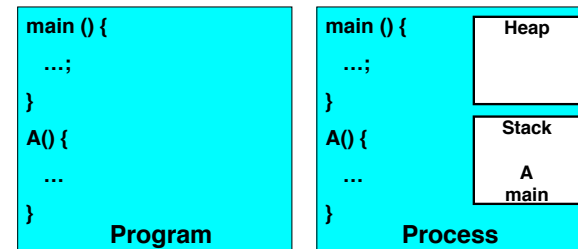
- Must construct new PCB
 - Inexpensive
- Must set up new page tables for address space
 - More expensive
- Copy data from parent process? (Unix `fork()`)
 - Semantics of Unix `fork()` are that the child process gets a complete copy of the parent memory and I/O state
 - Originally *very* expensive
 - Much less expensive with “copy on write”
- Copy I/O state (file handles, etc)
 - Medium expense

8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.33

Process =? Program



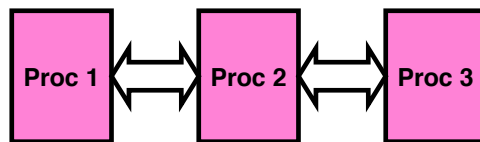
- More to a process than just a program:
 - Program is just part of the process state
 - I run emacs on lectures.txt, you run it on homework.java – same program, different processes
- Less to a process than a program:
 - A program can invoke more than one process
 - cc starts up cpp, cc1, cc2, as, and ld

8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.34

Multiple Processes Collaborate on a Task



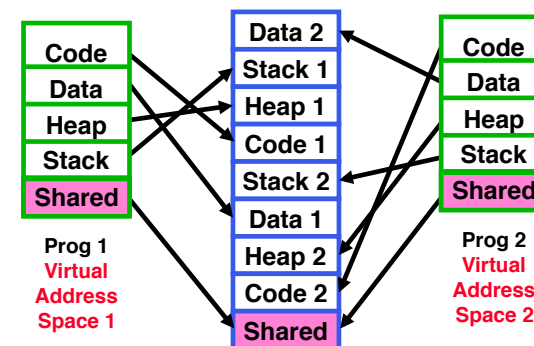
- Need Communication mechanism:
 - Separate address spaces isolates processes
 - Shared-Memory Mapping
 - » Accomplished by mapping addresses to common DRAM
 - » Read and Write through memory
 - Message Passing
 - » `send()` and `receive()` messages
 - » Works across network

8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.35

Shared Memory Communication



- Communication occurs by “simply” reading/writing to shared address page
 - Really low overhead communication
 - Introduces complex synchronization problems

8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.36

Inter-process Communication (IPC)

- Mechanism for processes to communicate and to synchronize their actions
- Message system – processes communicate with each other without resorting to shared variables
- IPC facility provides two operations:
 - `send(message)` – message size fixed or variable
 - `receive(message)`
- If P and Q wish to communicate, they need to:
 - establish a *communication channel* between them
 - exchange messages via `send/receive`
- Implementation of communication link
 - physical (e.g., shared memory, hardware bus, syscall/trap)
 - logical (e.g., logical properties)

8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.37

Modern “Lightweight” Process with Threads

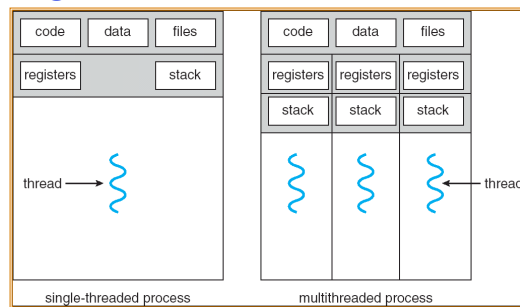
- Thread: *a sequential execution stream within process* (Sometimes called a “Lightweight process”)
 - Process still contains a single Address Space
 - No protection between threads
- Multithreading: *a single program made up of a number of different concurrent activities*
 - Sometimes called multitasking, as in Ada ...
- Why separate the concept of a thread from that of a process?
 - Discuss the “thread” part of a process (concurrency)
 - Separate from the “address space” (protection)
 - Heavyweight Process = Process with one thread

8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.38

Single and Multithreaded Processes



- Threads encapsulate concurrency: “Active” component
- Address spaces encapsulate protection: “Passive” part
 - Keeps buggy program from trashing the system
- Why have multiple threads per address space?

8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.39

Examples of multithreaded programs

- Embedded systems
 - Elevators, Planes, Medical systems, Wristwatches
 - Single Program, concurrent operations
- Most modern OS kernels
 - Internally concurrent because have to deal with concurrent requests by multiple users
 - But no protection needed within kernel
- Database Servers
 - Access to shared data by many concurrent users
 - Also background utility processing must be done

8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.40

Examples of multithreaded programs (con't)

- Network Servers
 - Concurrent requests from network
 - Again, single program, multiple concurrent operations
 - File server, Web server, and airline reservation systems
- Parallel Programming (More than one physical CPU)
 - Split program into multiple threads for parallelism
 - This is called Multiprocessing
- Some multiprocessors are actually uniprogrammed:
 - Multiple threads in one address space but one program at a time

8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.41

Classification

# threads Per AS:	# of addr spaces:	One	Many
		One	Many
One		MS/DOS, early Macintosh	Traditional UNIX
Many		Embedded systems (Geoworks, VxWorks, JavaOS, etc) JavaOS, Pilot(PC)	Mach, OS/2, Linux Win NT to 7, Solaris, HP-UX, OS X

- Real operating systems have either
 - One or many address spaces
 - One or many threads per address space

8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.42

Summary

- Processes have two parts
 - Threads (Concurrency)
 - Address Spaces (Protection)
- Concurrency accomplished by multiplexing CPU Time:
 - Unloading current thread (PC, registers)
 - Loading new thread (PC, registers)
 - Such context switching may be voluntary (`yield()`, I/O operations) or involuntary (timer, other interrupts)
- Protection accomplished restricting access:
 - Memory mapping isolates processes from each other
 - Dual-mode for isolating I/O, other resources
- Book talks about processes
 - When this concerns concurrency, really talking about thread portion of a process
 - When this concerns protection, talking about address space portion of a process

8/29/12

Ion Stoica CS162 ©UCB Fall 2012

Lec 1.43