

Boot Loader

Spring 2015

- Multi-stage boot loader
- Old Intel PC architecture (still used!) - BIOS
- Master Boot Record located at block 0
- Volume Boot Record
- OS Loader
- Current PC architecture (2005+)
- UEFI knows how to read one or more file systems
- Loads OS loader from a boot partition
- · Embedded systems (e.g., ARM-based devices) - Custom boot firmware on the processor chip

Operating System vs. Kernel

Kernel

- "nucleus" of the OS; main component
- Provides abstraction layer to underlying hardware
- Manages system resources (CPU, file systems, memory, network)
- Enforces policies
- · Rest of the OS
- Utility software, windowing system, print spoolers, etc.
- · Kernel mode vs. user mode execution
- Flag in the CPU
- Kernel mode = can execute privileged instructions

Mode switch

- · Transition from user to kernel mode (and back)
- · Includes a change in flow
- Cannot just execute user's instructions in kernel mode!
- Well-defined addresses set up at initialization
- · Change mode via:
- Hardware interrupt
- Software trap (or syscall)
- Violations (exceptions): illegal instruction or memory reference

Context Switch Mode switch + change executing process

© 2014 Paul Krzyzanowski



· Crucial for:

- Preempting a running process to give someone else a chance (force a context switch)
- Including ability to kill the process
- Giving the OS a chance to poll hardware
- OS bookkeeping

Timer interrupts Windows Typically 64 or 100 interrupts per second Apps can raise this to 1024 interrupts per second Interrupts from Programmable Interval Timer (PIT) or HPET (High Precision Event Timer) and from a local APIC timer (one per CPU) all at the same rate Interrupt frequency varies per kernel and configuration Linux 2.4: 100 Hz Linux 2.6.14: 250 Hz Linux 2.6.18 and beyond: aperiodic – tickless kernel PIT not used for periodic interrupts; just APIC timer interrupts Kernel determines when the next interrupt should take place

© 2014-2015 Paul Krzyzanowsk





Enterta a de site	and all	
 Entering the keep 		
 Hardware inte 		
 Asynchronou 	us events (I/O, clock, etc.)	
 <u>Do not</u> relate 	to the context of the current process [kernel context]	
 Violations 		
 Are related to 	o the context of the current process [process context]	
 Examples: ille 	egal memory access, divide by zero, illegal instruction	
 Software initia 	ated traps (software interrupts)	
 System call f 	from the current process [process context]	
The view of me	emory does not change on a trap	
- The currently	executing process' address space is active on a trap	
 Saving state 		
 Kernel stack s 	switched in upon entering kernel mode	
 Kernel must s 	save machine state before servicing event	
 Registers, fla 	ags (program status word), program counter,	

Processes in a Multitasking Environment

- Multiple concurrent processes
- Each process has a unique identifier: $\ensuremath{\text{Process ID}}$ (PID)
- Asynchronous events (interrupts) may occur
 The OS will have to take care of them
- Processes may request operations that take a long time They have nothing to do now but wait
- Goal: always have some process running
 Context saving/switching
 - Processes may be suspended and resumed
 - Need to save all state about a process so we can restore it



Keeping track of processes

- Process list stores a Process Control Block (PCB) per process
- A Process Control Block contains:
 - Process ID
 - Machine state (registers, program counter, stack pointer)
 - Parent & list of children
 - Process state (ready, running, blocked)
- Memory map
- Open file descriptors
- Owner (user ID) determine access & signaling privileges
- Event descriptor if the process is blocked
- Signals that have not yet been handled
- Policy items: Scheduling parameters, memory limits
- Timers for accounting (time & resource utilization)
- (Process group)

© 2014-2015 Paul Krzyzanowski













Fork Example		
#in	clude <stdio.h></stdio.h>	
mai	n(int argc, char **argv) {	
	int pid;	
	<pre>switch (pid=fork()) {</pre>	
	<pre>case 0: printf("I'm the child\n"); break;</pre>	
	default:	
	printf("I'm the parent of %d\n", pid); break;	
	case -1:	
	perror("fork");	
1	}	
3		















wait system call

- · Suspend execution until a child process exits
- wait returns the exit status of that child.

int pid, my_pid, status;





