

SIGOps

Vijay Ramesh

September 8, 2008

Information

SIGOps

Introduction

Operating System

Operating System Layout

Different approaches

Kernel

Introducing the kernel

Userland

Userland

Applications

Applications

Hardware

Needs

Solutions

Privileged Instructions

Preemption

Virtual Memory

OS Designs

Monolithic Kernel

Microkernel

Hybrid Kernels

Project Ideas

Projects

Information

- ▶ Chairs :
Vijay Ramesh [vramesh2@illinois.edu]
Nathan Lawrence [nlawren2@illinois.edu]
David Majnemer [dmajnem2@illinois.edu]
- ▶ Mailing list at sigops-l@acm.uiuc.edu

Operating System

- ▶ What is an OS and what does an OS do?

Operating System

- ▶ What is an OS and what does an OS do?
- ▶ Provides abstraction from system.

Operating System

- ▶ What is an OS and what does an OS do?
- ▶ Provides abstraction from system.
- ▶ Manages resources.

Layout

- ▶ Kernel

Layout

- ▶ Userland
- ▶ Kernel

Layout

- ▶ Applications
- ▶ Userland
- ▶ Kernel

Designs

- ▶ Monolithic [Linux, Windows [old], Unix]

Designs

- ▶ Monolithic [Linux, Windows [old], Unix]
- ▶ Microkernel [Minix, L4, Singularity]

Designs

- ▶ Monolithic [Linux, Windows [old], Unix]
- ▶ Microkernel [Minix, L4, Singularity]
- ▶ Hybrid [OS X, Windows [new]]

Kernel

- ▶ Linux is a kernel, not an OS.

Kernel

- ▶ Linux is a kernel, not an OS.
- ▶ The kernel is the most essential piece of the operating system needed.

Kernel

- ▶ Linux is a kernel, not an OS.
- ▶ The kernel is the most essential piece of the operating system needed.
- ▶ Generally consists of setup code on the platform.

Kernel

- ▶ Linux is a kernel, not an OS.
- ▶ The kernel is the most essential piece of the operating system needed.
- ▶ Generally consists of setup code on the platform.
- ▶ Depending on the OS design, contains core processes [scheduler, memory manager, etc.].

Kernel

- ▶ Linux is a kernel, not an OS.
- ▶ The kernel is the most essential piece of the operating system needed.
- ▶ Generally consists of setup code on the platform.
- ▶ Depending on the OS design, contains core processes [scheduler, memory manager, etc.].
- ▶ All kernels provide some mechanism to run tasks.

Kernel

- ▶ Linux is a kernel, not an OS.
- ▶ The kernel is the most essential piece of the operating system needed.
- ▶ Generally consists of setup code on the platform.
- ▶ Depending on the OS design, contains core processes [scheduler, memory manager, etc.].
- ▶ All kernels provide some mechanism to run tasks.
- ▶ Even in microkernel design, the kernel provides a mechanism for ITC.

Userland

- ▶ Userland generally holds the rest of the abstractions and framework.

Userland

- ▶ Userland generally holds the rest of the abstractions and framework.
- ▶ Userland has some means of interacting with the kernel [libraries].

Userland

- ▶ Userland generally holds the rest of the abstractions and framework.
- ▶ Userland has some means of interacting with the kernel [libraries].
- ▶ In microkernel design core processes are often moved to the userland.

Applications

- ▶ Applications use these utilities to access hardware .

Applications

- ▶ Applications use these utilities to access hardware .
- ▶ They don't have to be aware that they are running along side other tasks.

Applications

- ▶ Applications use these utilities to access hardware .
- ▶ They don't have to be aware that they are running along side other tasks.
- ▶ Isolation of task state.

Applications

- ▶ Applications use these utilities to access hardware .
- ▶ They don't have to be aware that they are running along side other tasks.
- ▶ Isolation of task state.
- ▶ They think they have unlimited/unrealistic resources when that isn't the case [VM]

Needs

- ▶ Privileged instructions needed.

Needs

- ▶ Privileged instructions needed.
- ▶ Method of preemption needed.

Needs

- ▶ Privileged instructions needed.
- ▶ Method of preemption needed.
- ▶ Method of isolation needed.

Needs

- ▶ Privileged instructions for device/bus accesses.

Needs

- ▶ Privileged instructions for device/bus accesses.
- ▶ Hardware support for virtual memory.

Needs

- ▶ Privileged instructions for device/bus accesses.
- ▶ Hardware support for virtual memory.
- ▶ Hardware interrupts used for preemption.

Privilege levels

- ▶ Most architectures, especially today, have 2 main privilege modes.

Privilege levels

- ▶ Most architectures, especially today, have 2 main privilege modes.
- ▶ In x86 these are referred to as rings.

Privilege levels

- ▶ Most architectures, especially today, have 2 main privilege modes.
- ▶ In x86 these are referred to as rings.
- ▶ There are 2 main rings you need to know about.

Privilege levels

- ▶ Most architectures, especially today, have 2 main privilege modes.
- ▶ In x86 these are referred to as rings.
- ▶ There are 2 main rings you need to know about.
- ▶ Ring 0 and ring 3.

Privilege levels

- ▶ Most architectures, especially today, have 2 main privilege modes.
- ▶ In x86 these are referred to as rings.
- ▶ There are 2 main rings you need to know about.
- ▶ Ring 0 and ring 3.
- ▶ Ring 0 tasks have access to write to the page table and access to some other instruction.

Privilege levels

- ▶ Most architectures, especially today, have 2 main privilege modes.
- ▶ In x86 these are referred to as rings.
- ▶ There are 2 main rings you need to know about.
- ▶ Ring 0 and ring 3.
- ▶ Ring 0 tasks have access to write to the page table and access to some other instruction.
- ▶ Ring 3 tasks don't have access to write to the page table.

Preemption

- ▶ Once you, as the kernel, let a process run, you yield power to it.

Preemption

- ▶ Once you, as the kernel, let a process run, you yield power to it.
- ▶ In a cooperative multitasking system, you wait until it hands the processor back to you.

Preemption

- ▶ Once you, as the kernel, let a process run, you yield power to it.
- ▶ In a cooperative multitasking system, you wait until it hands the processor back to you.
- ▶ This doesn't work too well when you have a malicious program.

Preemption

- ▶ Once you, as the kernel, let a process run, you yield power to it.
- ▶ In a cooperative multitasking system, you wait until it hands the processor back to you.
- ▶ This doesn't work too well when you have a malicious program.
- ▶ Now, kernels force or preempt processes for CPU time.

Preemption (Cont.)

- ▶ Preemption is generally accomplished through hardware interrupts at given intervals.
 - ▶ The kernel sets a timer and hands the CPU to the process.

Preemption (Cont.)

- ▶ Preemption is generally accomplished through hardware interrupts at given intervals.
 - ▶ The kernel sets a timer and hands the CPU to the process.
 - ▶ The process runs until the timer goes off.

Preemption (Cont.)

- ▶ Preemption is generally accomplished through hardware interrupts at given intervals.
 - ▶ The kernel sets a timer and hands the CPU to the process.
 - ▶ The process runs until the timer goes off.
 - ▶ The timer generates an interrupt.

Preemption (Cont.)

- ▶ Preemption is generally accomplished through hardware interrupts at given intervals.
 - ▶ The kernel sets a timer and hands the CPU to the process.
 - ▶ The process runs until the timer goes off.
 - ▶ The timer generates an interrupt.
 - ▶ The CPU calls the interrupt handler, which happens to be in privileged mode.

Preemption (Cont.)

- ▶ Preemption is generally accomplished through hardware interrupts at given intervals.
 - ▶ The kernel sets a timer and hands the CPU to the process.
 - ▶ The process runs until the timer goes off.
 - ▶ The timer generates an interrupt.
 - ▶ The CPU calls the interrupt handler, which happens to be in privileged mode.
 - ▶ Now, the kernel is in control again.

Virtual Memory

- ▶ All applications think they are the only application on the system.

Virtual Memory

- ▶ All applications think they are the only application on the system.
- ▶ So, they all think they can address all of memory.

Virtual Memory

- ▶ All applications think they are the only application on the system.
- ▶ So, they all think they can address all of memory.
- ▶ This isn't necessarily true, since
 - ▶ They aren't the only task running.
 - ▶ There might not even be that much physical memory to begin with.

Virtual Memory

- ▶ All applications think they are the only application on the system.
- ▶ So, they all think they can address all of memory.
- ▶ This isn't necessarily true, since
 - ▶ They aren't the only task running.
 - ▶ There might not even be that much physical memory to begin with.
- ▶ On 32 bit platforms, you can physically address 4 GB of ram.

Virtual Memory

- ▶ All applications think they are the only application on the system.
- ▶ So, they all think they can address all of memory.
- ▶ This isn't necessarily true, since
 - ▶ They aren't the only task running.
 - ▶ There might not even be that much physical memory to begin with.
- ▶ On 32 bit platforms, you can physically address 4 GB of ram.
- ▶ In order to emulate this, most OSES today provide virtual memory.

Virtual Memory

- ▶ All applications think they are the only application on the system.
- ▶ So, they all think they can address all of memory.
- ▶ This isn't necessarily true, since
 - ▶ They aren't the only task running.
 - ▶ There might not even be that much physical memory to begin with.
- ▶ On 32 bit platforms, you can physically address 4 GB of ram.
- ▶ In order to emulate this, most OSes today provide virtual memory.
- ▶ Virtual memory provides an abstraction and process isolation.

Monolithic Kernel

- ▶ A monolithic kernel is a big kernel with everything in kernel mode [privileged mode].

Monolithic Kernel

- ▶ A monolithic kernel is a big kernel with everything in kernel mode [privileged mode].
- ▶ This includes the task executer, the scheduler, the memory manager, all drivers, etc.

Monolithic Kernel

- ▶ A monolithic kernel is a big kernel with everything in kernel mode [privileged mode].
- ▶ This includes the task executer, the scheduler, the memory manager, all drivers, etc.
- ▶ Monolithic kernels are fast, due to fewer context switches.

Monolithic Kernel

- ▶ A monolithic kernel is a big kernel with everything in kernel mode [privileged mode].
- ▶ This includes the task executer, the scheduler, the memory manager, all drivers, etc.
- ▶ Monolithic kernels are fast, due to fewer context switches.
- ▶ Monolithic kernels are unsafe due to large/untrusted applications living in one address space in privileged mode.

Microkernel

- ▶ A microkernel is a tiny kernel with nothing by the executer in privileged mode.

Microkernel

- ▶ A microkernel is a tiny kernel with nothing by the executer in privileged mode.
- ▶ All other parts of the OS, even some trusted components, like the scheduler will live in userland.

Microkernel

- ▶ A microkernel is a tiny kernel with nothing by the executer in privileged mode.
- ▶ All other parts of the OS, even some trusted components, like the scheduler will live in userland.
- ▶ Microkernels are slow due to heavy context switching.

Microkernel

- ▶ A microkernel is a tiny kernel with nothing by the executer in privileged mode.
- ▶ All other parts of the OS, even some trusted components, like the scheduler will live in userland.
- ▶ Microkernels are slow due to heavy context switching.
- ▶ Microkernels are safer due to separate address spaces for drivers. If a driver dies, it doesn't BSOD; you can restart it.

Hybrid Kernels

- ▶ A hybrid kernel tries to get the best of both worlds.

Hybrid Kernels

- ▶ A hybrid kernel tries to get the best of both worlds.
- ▶ Task executer and core components [scheduler, etc.] are in privileged mode.

Hybrid Kernels

- ▶ A hybrid kernel tries to get the best of both worlds.
- ▶ Task executer and core components [scheduler, etc.] are in privileged mode.
- ▶ Drivers are in user mode.

Hybrid Kernels

- ▶ A hybrid kernel tries to get the best of both worlds.
- ▶ Task executer and core components [scheduler, etc.] are in privileged mode.
- ▶ Drivers are in user mode.
- ▶ Can still restart drivers if they fail, but can't really do anything if the scheduler dies.

Hybrid Kernels

- ▶ A hybrid kernel tries to get the best of both worlds.
- ▶ Task executer and core components [scheduler, etc.] are in privileged mode.
- ▶ Drivers are in user mode.
- ▶ Can still restart drivers if they fail, but can't really do anything if the scheduler dies.
- ▶ Faster than a microkernel but slower than a monolithic kernel.

Hybrid Kernels

- ▶ A hybrid kernel tries to get the best of both worlds.
- ▶ Task executer and core components [scheduler, etc.] are in privileged mode.
- ▶ Drivers are in user mode.
- ▶ Can still restart drivers if they fail, but can't really do anything if the scheduler dies.
- ▶ Faster than a microkernel but slower than a monolithic kernel.
- ▶ Safer than a monolithic kernel but more vulnerable than a microkernel.

Projects

- ▶ Popcorn – A microkernel OS.

Projects

- ▶ Popcorn – A microkernel OS.
- ▶ VoWFS – Version on Write File System

Projects

- ▶ Popcorn – A microkernel OS.
- ▶ VoWFS – Version on Write File System
- ▶ ModalWMM – A window manager

Projects

- ▶ Popcorn – A microkernel OS.
- ▶ VoWFS – Version on Write File System
- ▶ ModalWMM – A window manager
- ▶ OpenMoko – A mobile phone OS.

Projects

- ▶ Popcorn – A microkernel OS.
- ▶ VoWFS – Version on Write File System
- ▶ ModalWMM – A window manager
- ▶ OpenMoko – A mobile phone OS.
- ▶ Your ideas.