CSE 4733/6733 - Operating System 1

Stephen A. Torri, Ph.D.

Mississippi State University

CSE 4733/6733 - Operating System 1

Stephen A. Torri, Ph.D.

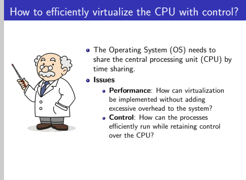
Mississippi State University

Introduction

Introduction

How to efficiently virtualize the CPU with control?

- The Operating System (OS) needs to share the central processing unit (CPU) by time sharing.
- **Issues**
  - **Performance**: How can virtualization be implemented without adding excessive overhead to the system?
  - **Control**: How can the processes efficiently run while retaining control over the CPU?



1.
   - We need to run processes efficiently while retaining control over the CPU.[1]
   - The OS handles the important tasks to protect system resources.[1]
   - Obtaining high performance while maintaining control is one of the central challenges in building an OS.[1]

Direct Execution without limits

Just run the program directly on the CPU

OS	Program
1. Create an entry for process list 2. Allocate memory for program 3. Load program into memory 4. Set up stack with *agrc/agrv* 5. Clear registers 6. Execute call main()	
	7. Run *main()* 8. Execute *return* from *main()*
9. Free memory of process 10. Remove from process lisht	

Without limits on running programs, the OS would not be in control of anything.

Direct Execution without limits
Just run the program directly on the CPU
OS | Program
1. Create an entry for process list 2. Allocate memory for program 3. Load program into memory 4. Set up stack with agrc/agrv 5. Clear registers 6. Execute call main()
7. Run main() 8. Execute return from main()
9. Free memory of process 10. Remove from process lisht
Without limits on running programs, the OS would not be in control of anything.

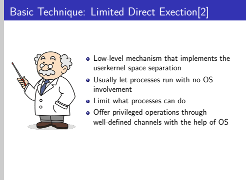
1. Question 1: If we just run a program, how can the OS make sure the program doesn't do anything that we don't want it to do while still running it efficiently?
2. Question 2: When we are running a process, how does the OS stop it from running and switch to another process, thus implementing the time sharing we require to virtualize the CPU?

Limited Direct Execution

Limited Direct Execution

Basic Technique: Limited Direct Exection[2]

- Low-level mechanism that implements the userkernel space separation
- Usually let processes run with no OS involvement
- Limit what processes can do
- Offer privileged operations through well-defined channels with the help of OS



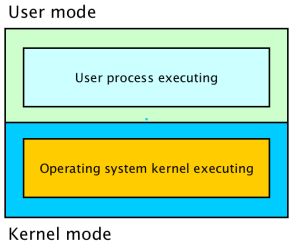
1. To increase a program's execution, OS developers created a technique called limited direct execution. Just run the program directly on the CPU.

Restricted Operation

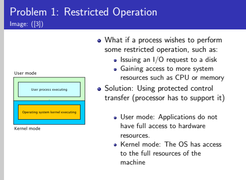
Restricted Operation

Problem 1: Restricted Operation

Image: ([3])

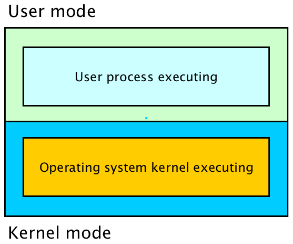


- What if a process wishes to perform some restricted operation, such as:
  - Issuing an I/O request to a disk
  - Gaining access to more system resources such as CPU or memory
- Solution: Using protected control transfer (processor has to support it)
  - User mode: Applications do not have full access to hardware resources.
  - Kernel mode: The OS has access to the full resources of the machine



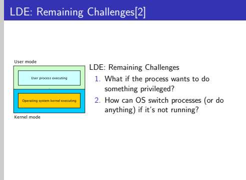
1. Processes start in user mode.
2. OS starts in kernel mode.

LDE: Remaining Challenges[2]



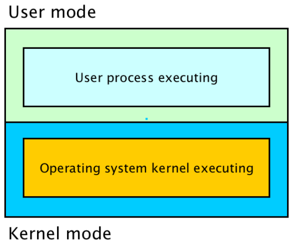
LDE: Remaining Challenges

1. What if the process wants to do something privileged?
2. How can OS switch processes (or do anything) if it's not running?

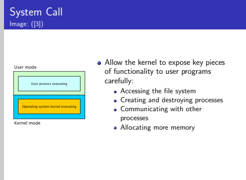


System Call

Image: ([3])

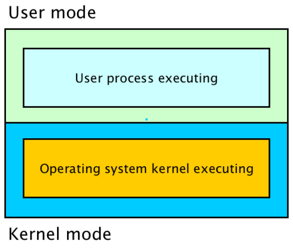


- Allow the kernel to expose key pieces of functionality to user programs carefully:
  - Accessing the file system
  - Creating and destroying processes
  - Communicating with other processes
  - Allocating more memory

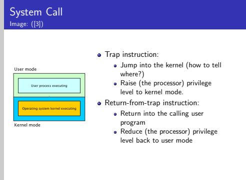


System Call

Image: ([3])



- Trap instruction:
  - Jump into the kernel (how to tell where?)
  - Raise (the processor) privilege level to kernel mode.
- Return-from-trap instruction:
  - Return into the calling user program
  - Reduce (the processor) privilege level back to user mode



Limited Direction Execution Protocol

OS system boot

OS @ boot (kernel mode)	**Hardware**
Initialize trap table	
	remember the address of the syscall handler

Limited Direction Execution Protocol
OS system boot

OS @ boot (kernel mode)	Hardware
Initialize trap table	
	remember the address of the syscall handler

Limited Direction Execution Protocol

OS system call from program

OS @ boot (kernel mode)	**Hardware**	**Program (user mode)**
1: start process (kernel mode)		
	2: Switch to user mode	
		3: Call main then syscall
	4: Switch to kernel mode	
5. Handle trap		
	6.Switch to user mode	
		7. Finish main
	8. Switch to kernel mode	
9. Remove process		

Limited Direction Execution Protocol
OS system call from program

1. Step 1
   - Create entry for process list
   - Allocate memory for program
   - Load program into memory
   - Setup user stack with argv
   - Fill kernel stack with reg/PC
   - return-from-trap
2. Step 2
   - restore regs from kernel stack
   - move to user mode
   - jump to main

Limited Direction Execution Protocol

OS system call from program

OS @ boot (kernel mode)	Hardware	Program (user mode)
1: start process (kernel mode)		
	2: Switch to user mode	
		3: Call main then syscall
	4: Switch to kernel mode	
5. Handle trap		
	6.Switch to user mode	
		7. Finish main
	8. Switch to kernel mode	
9. Remove process		

Limited Direction Execution Protocol
OS system call from program
OS @ boot (kernel mode) | Hardware | Program (user mode)
1: start process (kernel mode)
2: Switch to user mode
3: Call main then syscall
4: Switch to kernel mode
5. Handle trap
6.Switch to user mode
7. Finish main
8. Switch to kernel mode
9. Remove process

1. Step 3
   - Run main()
   - Make system call
   - **trap** into OS
2. Step 4
   - save regs to kernel stack
   - move to kernel mode
   - jump to trap handler
3. Step 5
   - Handle trap
   - Do work of syscall
   - return-from-trap

Limited Direction Execution Protocol

OS system call from program

OS @ boot (kernel mode)	**Hardware**	**Program (user mode)**
1: start process (kernel mode)		
	2: Switch to user mode	
		3: Call main then syscall
	4: Switch to kernel mode	
5. Handle trap		
	6.Switch to user mode	
		7. Finish main
	8. Switch to kernel mode	
9. Remove process		

Limited Direction Execution Protocol
OS system call from program

1. Step 6
   - restore regs from kernel stack
   - move to user mode
   - jump to PC after trap
2. Step 7
   - Return from main
   - trap (via exit)
3. Step 8
   - save regs to kernel stack
   - move to kernel mode
   - jump to trap handler

Limited Direction Execution Protocol

OS system call from program

OS @ boot (kernel mode)	**Hardware**	**Program (user mode)**
1: start process (kernel mode)		
	2: Switch to user mode	
		3: Call main then syscall
	4: Switch to kernel mode	
5. Handle trap		
	6.Switch to user mode	
		7. Finish main
	8. Switch to kernel mode	
9. Remove process		

Limited Direction Execution Protocol
OS system call from program
OS @ boot (kernel mode) | Hardware | Program (user mode)
1: start process (kernel mode)
2: Switch to user mode
3: Call main then syscall
4: Switch to kernel mode
5. Handle trap
6.Switch to user mode
7. Finish main
8. Switch to kernel mode
9. Remove process

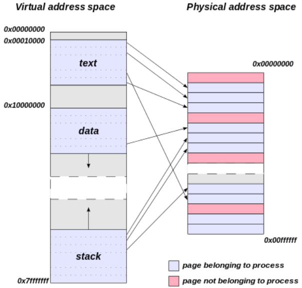
1. Step 9
   - Free the process memory
   - Remove from process list

Address Translation

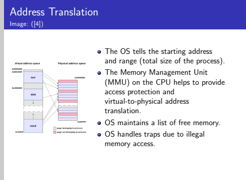
Address Translation

Address Translation

Image: ([4])



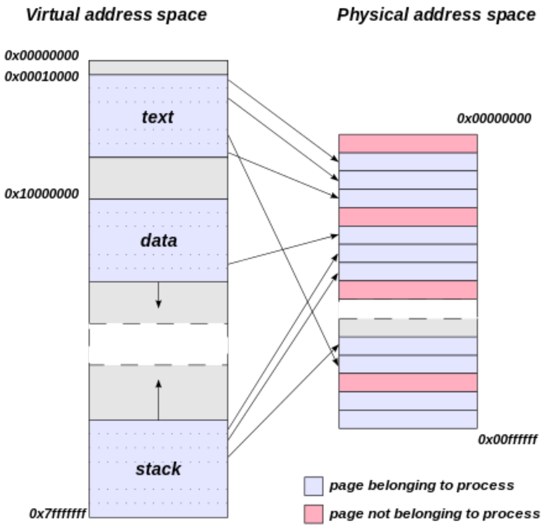
- The OS tells the starting address and range (total size of the process).
- The Memory Management Unit (MMU) on the CPU helps to provide access protection and virtual-to-physical address translation.
- OS maintains a list of free memory.
- OS handles traps due to illegal memory access.

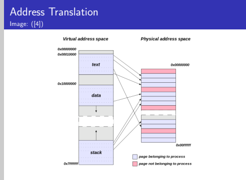


1. The MMU performs the translation for every memory access. It will generate a fault and trap the OS if access is illegal (e.g., a virtual address is out of bounds).

Address Translation

Image: ([4])





References I

E. Perry.
Mechanism: Limited direct execution, 2019.

Y. Cheng.
Cs 471 operating systems, 2019.

Unknown.
Initial definitions, 2023.

Wikipedia.
Page table, 2023.

References I

E. Perry.
Mechanism: Limited direct execution, 2019.
Y. Cheng.
Cs 471 operating systems, 2019.
Unknown.
Initial definitions, 2023.
Wikipedia.
Page table, 2023.