CS614: Linux Kernel Programming

Logistics, Introduction

Debadatta Mishra, CSE, IIT Kanpur

Course and instructors

$whereis cs614

Mon, Wed 2PM - 3.15PM RM-101
https://www.cse.iitk.ac.in/users/deba/cs614/

Piazza link: https://piazza.com/iitk.ac.in/secondsemester2023/cs614
Canvas: https://canvas.cse.iitk.ac.in/login

$whereis deba

KD 212, deba@cse.iitk.ac.in , Meeting hours: 3PM - 5PM Thursday

$ ls TAs

Arun KP (kparun@cse.iitk.ac.in), Rohit Singh (rohit@cse.iitk.ac.in)

Course policy

Add/Drop

Course registration and drop:- Ideally before 16th Jan 2023

Class guidelines

Keep your mobile phones switched off / silent

Ask questions and interact (both in class and in Piazza)

Be on time!

Evaluation

1. Quizzes (10%)
2. Assignments (40%)
3. Mid-semester (15%)
4. End-semester (35%)

Evaluation

1. Quizzes (10%)
2. Assignments (40%)
3. Mid-semester (15%)
4. End-semester (35%)

References

Operating Systems: Three Easy Pieces. Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau.
Understanding the Linux Kernel, Daniel P. Bovet, Marco Cesati.
Linux Kernel Development, 3rd Edition, Robert Love.
Linux Device Drivers, 3rd Edition, By Jonathan Corbet, Greg Kroah-Hartman, Alessandro Rubini.
Linux kernel documentation, Research papers

Evaluation

1. Quizzes (10%)
2. Assignments (40%)
3. Mid-semester (15%)
4. End-semester (35%)

“Take pride in honest hard work ”

“Cheating implies accepting defeat”

“If you are here to learn, never defeat the purpose by cheating”

https://www.cse.iitk.ac.in/pages/AntiCheatingPolicy.html

Homework -1

- Personal laptops with decent backup desirable
- **HW1: Setup a Virtual machine for the course (Due before next lecture)**
    - Create a Linux VM (Ubuntu Linux recommended) (KVM is prefered)
    - Download the Linux kernel version - linux-6.1.4
    - Compile and boot the latest kernel

OS refresher: True/False

1. Superuser (e.g., root user in UNIX) in a multi-tasking OS can execute all instructions provided by the hardware instruction set architecture (ISA).
2. Every process in a computer system is guaranteed to be in running state at least once during its life time.
3. A critical section consisting of a single instruction may require mutual exclusion.
4. A user process interrupted by a device interrupt is always scheduled immediately after the interrupt is serviced.
5. A page fault can be handled without changing any page table entry.

OS refresher: Quiz

In a uniprocessor system, in which of the following case(s), a process executing in user mode can cause an entry into the OS?

A. accessing a general purpose register like RAX
B. executing a JMP (jump) instruction
C. decrementing an unsigned integer value stored in a register beyond zero
D. executing a printf( ) statement
E. returning from a function

Revisiting Hello World!

hello.c

```
int main(void)
{
    printf("Hello world");
    while(1);
}
```

- What will be the execution behavior and why?
- Alternate ways to print "hello world" on screen?

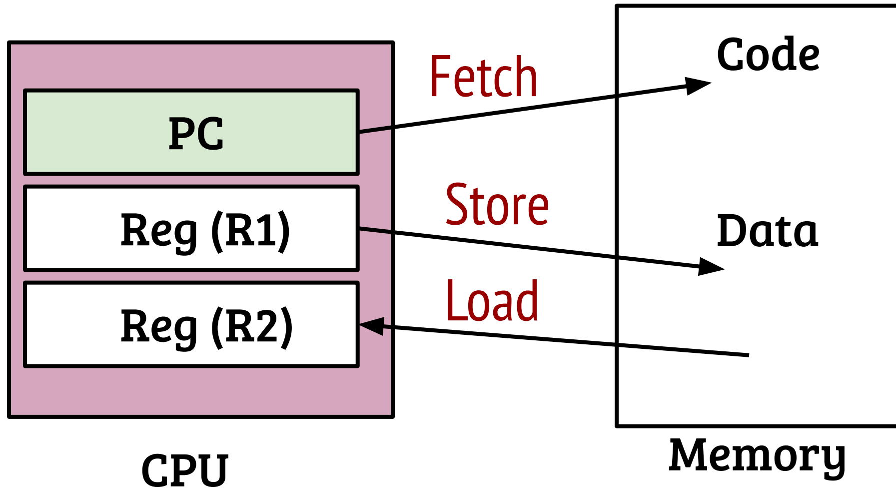
Program execution

You said only CPU can execute!

Inside program execution

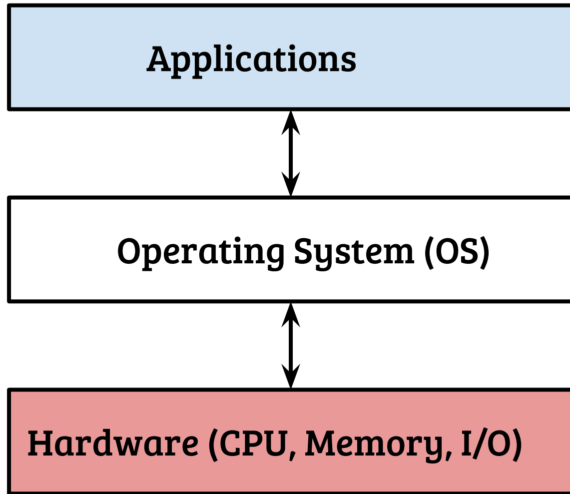
You said only CPU can execute!

CPU execution (from hardware perspective)



- Loads instruction pointed to by PC
- Decode instruction
- Load operand into registers
- Execute instruction (ALU)
- Store results

Role of the OS



- OS bridges the *semantic gap* between the notions of application execution and real execution
  - OS loads an executable from disk to memory, allocates/frees memory dynamically
  - OS initializes the CPU state i.e., the PC and other registers
  - OS provides interfaces to access I/O devices
- OS facilitates hardware resource sharing and management (How?)

Virtual view of resources

- Process
    - Each running process thinks that it owns the CPU

Virtual view of resources

- Process
    - Each running process thinks that it owns the CPU
- Address space
    - Each process feels like it has a huge address space

Virtual view of resources

- Process
  - Each running process thinks that it owns the CPU
- Address space
  - Each process feels like it has a huge address space
- File system tree
  - The user feels like operating on the files directly

Virtual view of resources

- Process
  - Each running process thinks that it owns the CPU
- Address space
  - Each process feels like it has a huge address space
- File system tree
  - The user feels like operating on the files directly
- What are the OS responsibilities in providing the above virtual notions?

Virtual view of resources

- Process
  - Each running process thinks that it owns the CPU
- Address space
  - Each process feels like it has a huge address space
- File system tree
  - The user feels like operating on the files directly
- What are the OS responsibilities in providing the above virtual notions?
  - The OS performs multiplexing of physical resources efficiently
  - Maintains mapping of virtual view to physical resource

Virtualization: Efficiency/performance

- Resource virtualization should not add *excessive* overheads
- Efficient when programs use the resources directly, infrequent OS mediation
    - Example: when a process is scheduled on CPU, it should execute without OS intervention
- What is the catch?

Virtualization: Efficiency/performance

- Resource virtualization should not add *excessive* overheads
- Efficient when programs use the resources directly, infrequent OS mediation
  - Example: when a process is scheduled on CPU, it should execute without OS intervention
- What is the catch?
  - Loss of control e.g., process running an infinite loop on a CPU
  - Isolation issues e.g., process accessing/changing OS data structures

Virtualization: Efficiency/performance

- Resource virtualization should not add *excessive* overheads
- Efficient when programs use the resources directly, infrequent OS mediation
  - Example: when a process is scheduled on CPU, it should execute without OS intervention
- What is the catch?
  - Loss of control e.g., process running an infinite loop on a CPU
  - Isolation issues e.g., process accessing/changing OS data structures

Conclusion: Some limits to direct access must be enforced.

Limited direct execution

- Can the OS enforce limits to an executing process by itself?

A process in execution

I want to take control of the CPU from this process which is executing an infinite loop, but how?

OS

I want to restrict this process accessing memory of other processes, but how? Monitoring each memory access is not efficient!

Process (a.out)
Register state (R1, R2, PC)
Memory state
Other states

PC
Reg (R1)
Reg (R2)
CPU

Fetch
Store
Load

Code
Data
Memory

A process in execution

I want to take control of the CPU from this process which is executing an infinite loop, but how?

Help me!

OS

I want to restrict this process accessing memory of other processes, but how?
Monitoring each memory access is not efficient!

Process (a.out)
Register state (R1, R2, PC)
Memory state
Other states

Limited direct execution

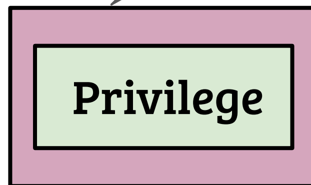
- Can the OS enforce limits to an executing process by itself?
- No, the OS can not enforce limits by itself and still achieve efficiency
- OS requires support from hardware!

Limited direct execution

- Can the OS enforce limits to an executing process?
- No, the OS can not enforce limits by itself and still achieve efficiency
- OS requires support from hardware!
- What kind of support is needed from the hardware?

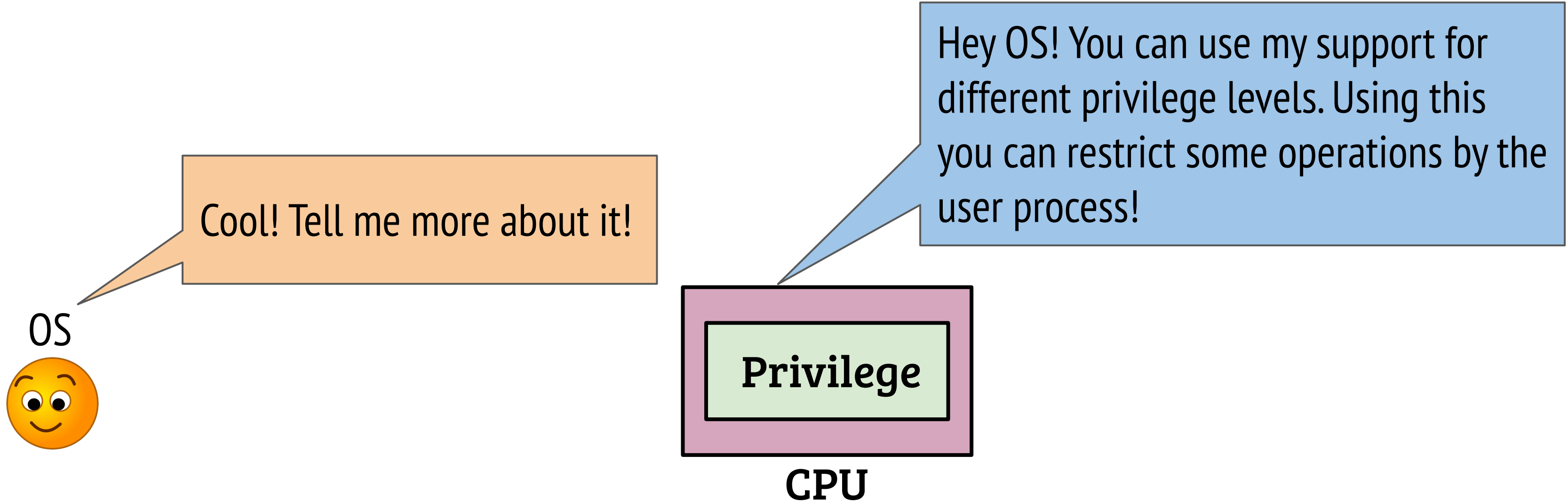
Hardware support: Privilege levels

Help me!

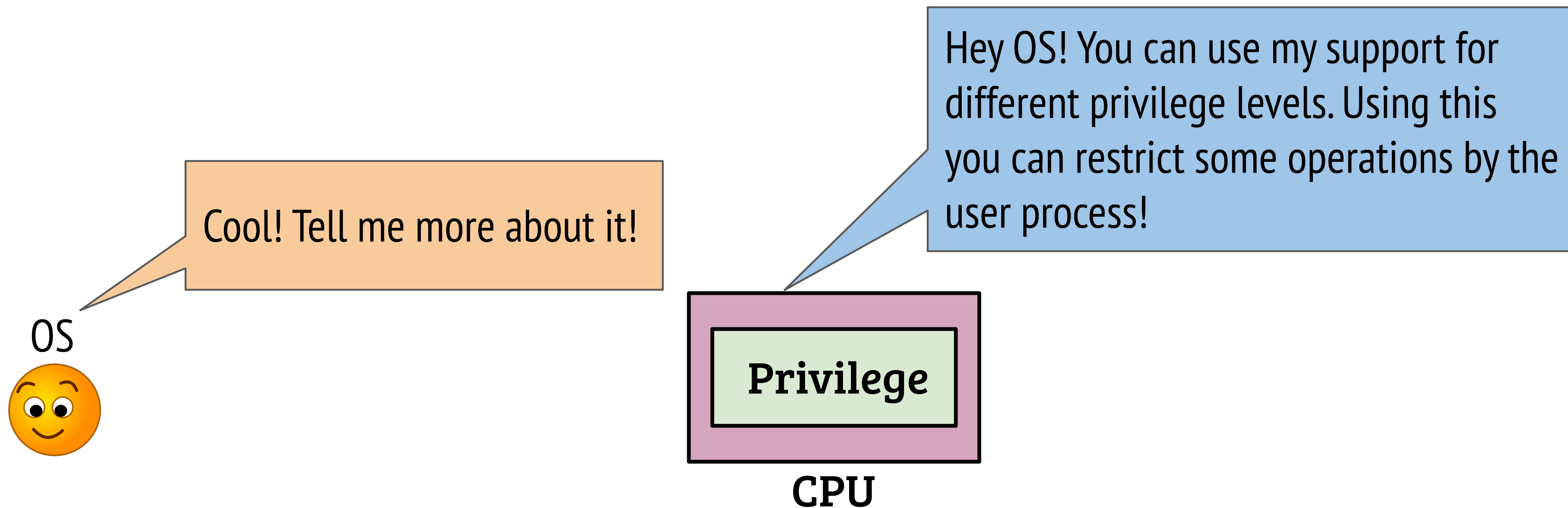


Hey OS! You can use my support for different privilege levels. Using this you can restrict some operations by the user process!

Hardware support: Privilege levels

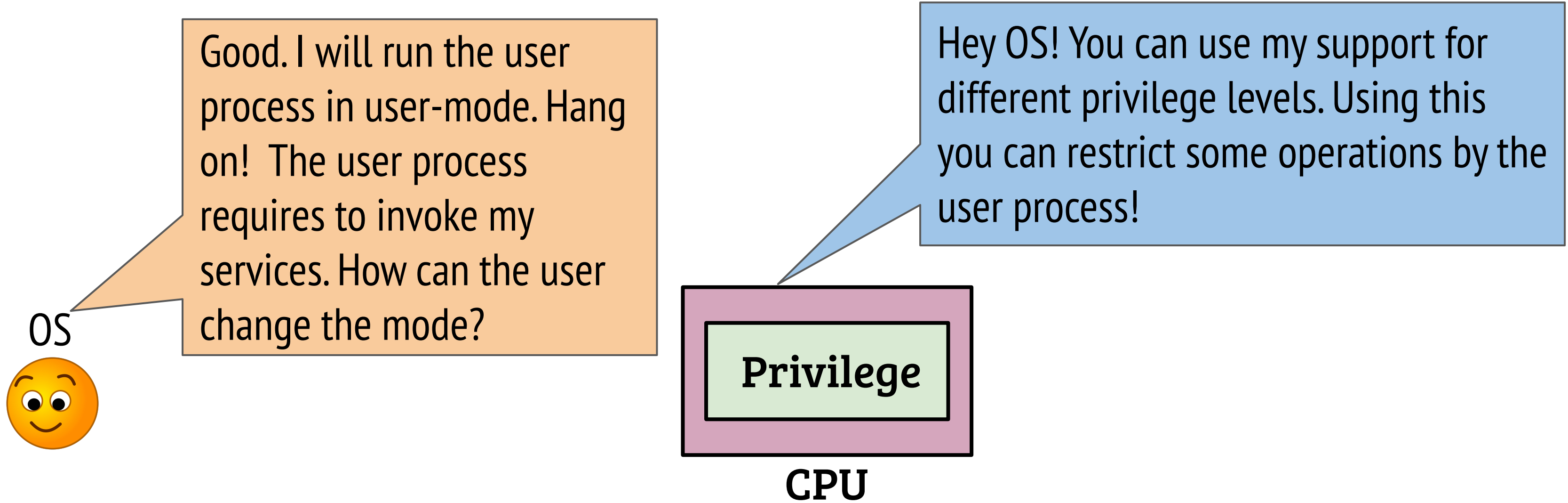


Hardware support: Privilege levels

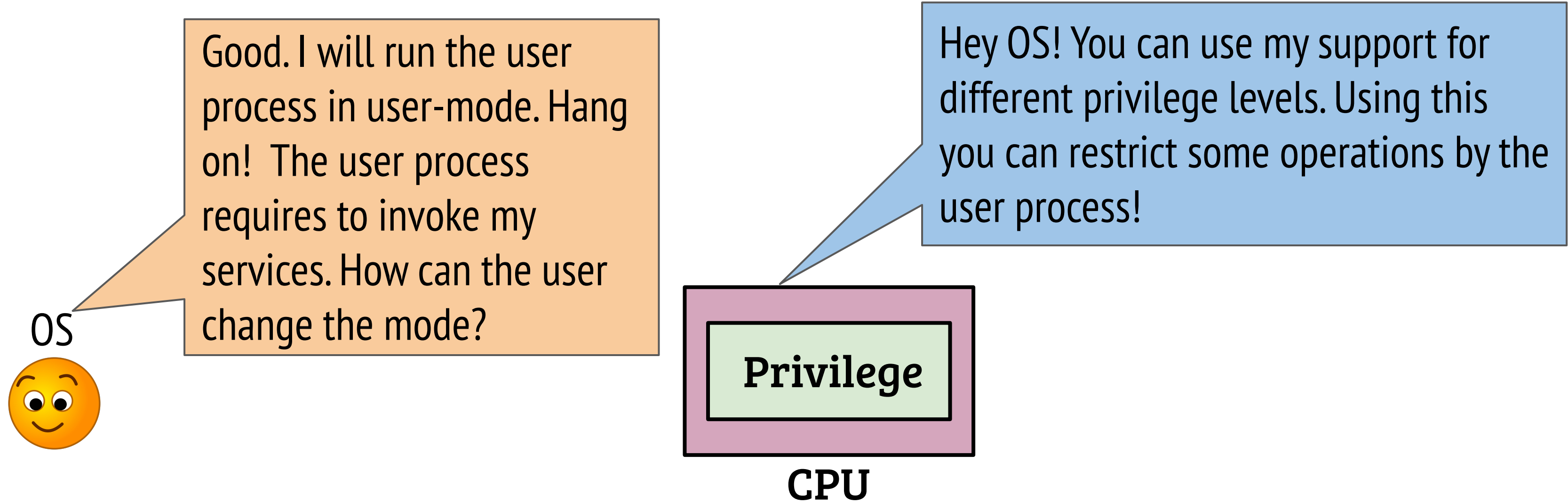


- CPU can execute in two modes: *user-mode* and *kernel-mode*
- Some operations are allowed only from kernel-mode (privileged OPs)
  - If executed from user mode, hardware will notify the OS by raising a fault/trap

Hardware support: Privilege levels



Hardware support: Privilege levels



- From user-mode, privilege level of CPU can not be changed directly
- The hardware provides entry instructions from the user-mode which causes a mode switch
- The OS can define the handler for different entry gates

Hardware support: Privilege levels

Okay. You said that if the process does some mischief from the user mode, you will notify me. That means, I can define handlers for faults and exceptions too.

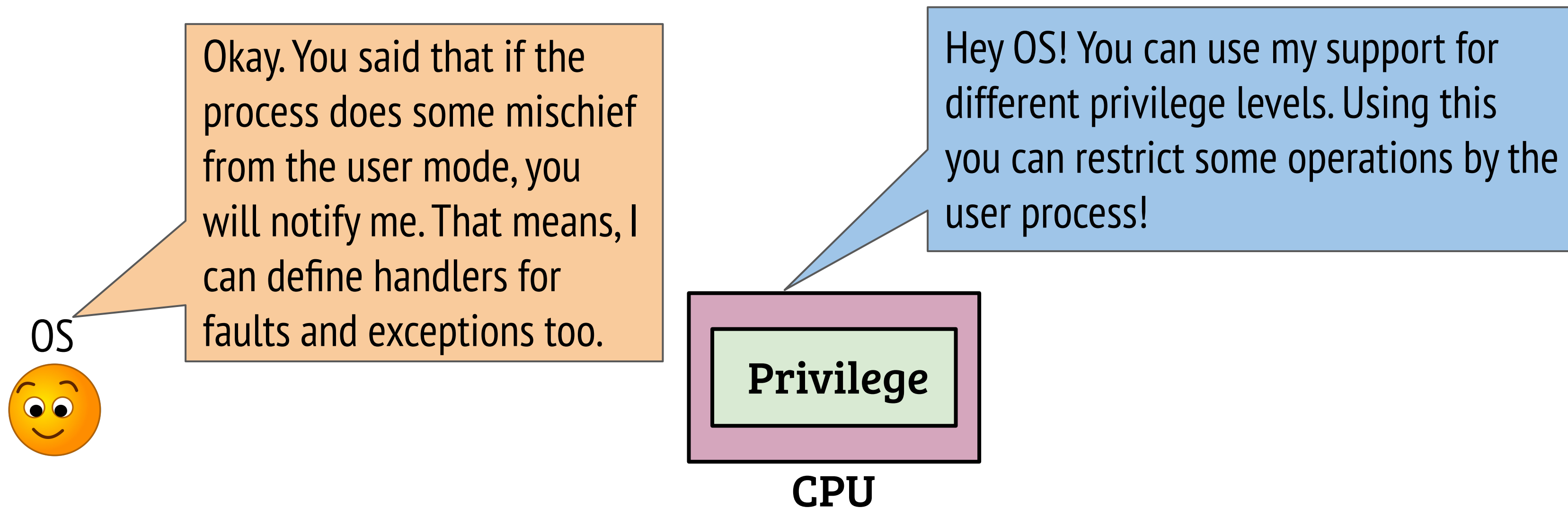
OS

Hey OS! You can use my support for different privilege levels. Using this you can restrict some operations by the user process!

Privilege

CPU

Hardware support: Privilege levels

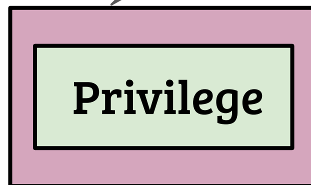


- The OS can register the handlers for faults and exceptions
- The OS can also register handlers for device interrupts
- *Registration of handlers is privileged!*

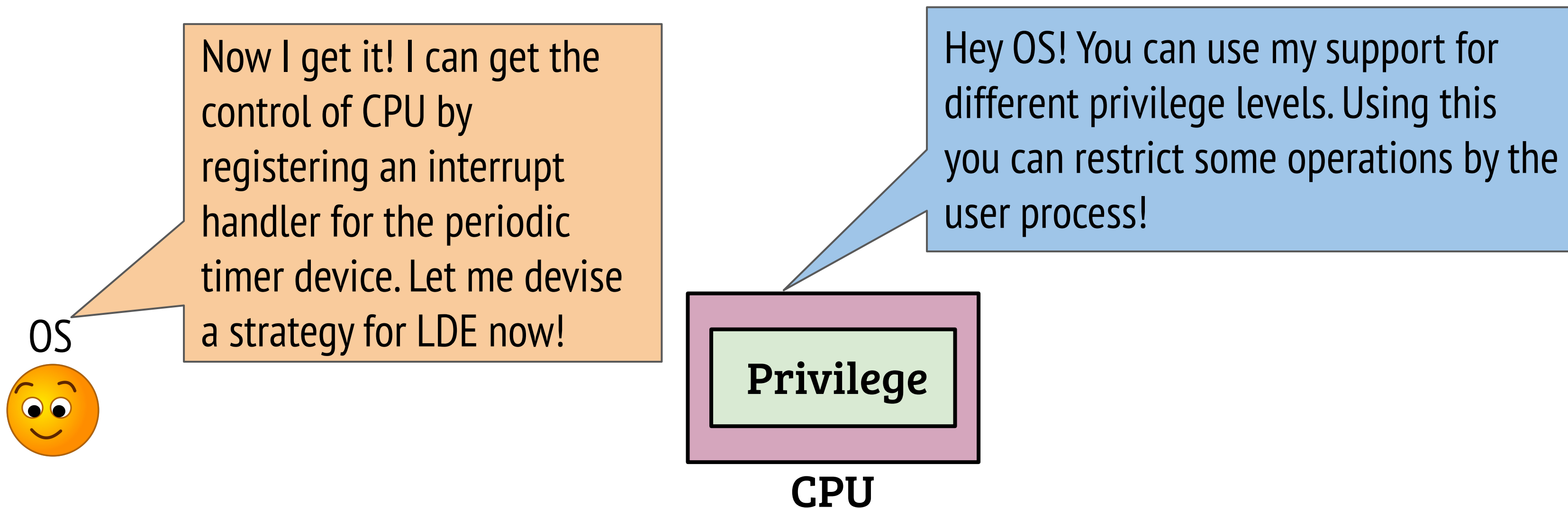
Hardware support: Privilege levels

Now I get it! I can get the control of CPU by registering an interrupt handler for the periodic timer device. Let me devise a strategy for LDE now!

Hey OS! You can use my support for different privilege levels. Using this you can restrict some operations by the user process!

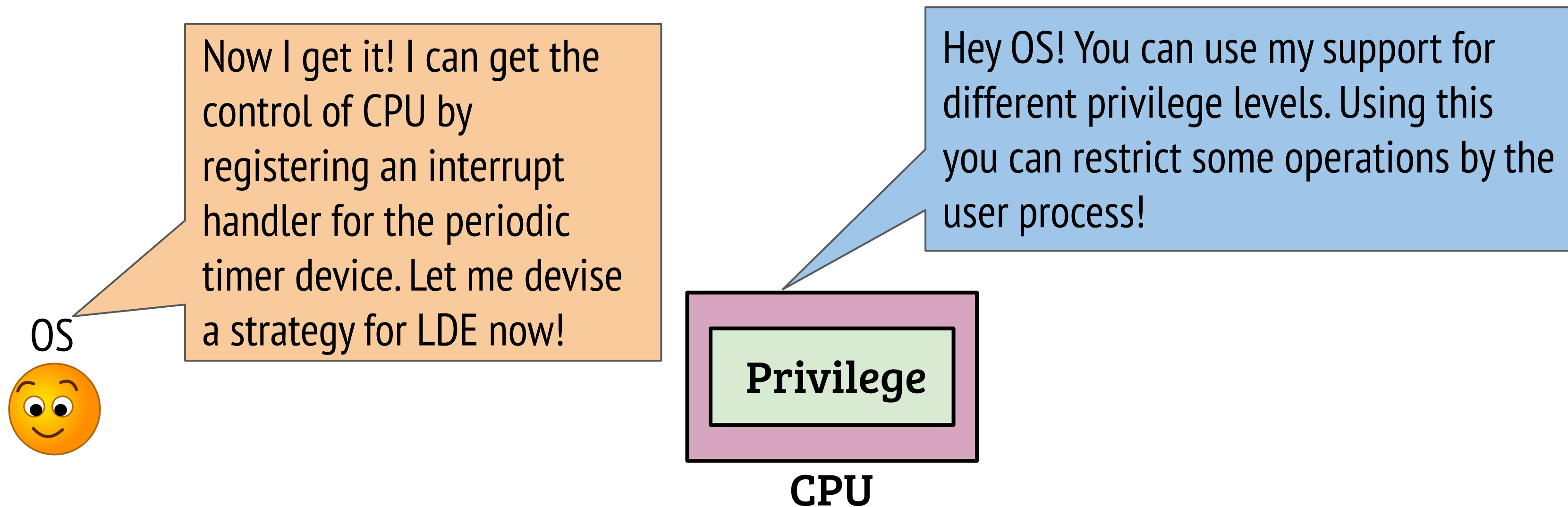


Hardware support: Privilege levels



- After the boot, the OS needs to configure the handlers for system calls, exceptions/faults and interrupts

Hardware support: Privilege levels



- After the boot, the OS needs to configure the handlers for system calls, exceptions/faults and interrupts
- The handler code is invoked by the OS when user-mode process invokes a system call or an exception or an external interrupt

Limited direct execution

- Can the OS enforce limits to an executing process?
- No, the OS can not enforce limits by itself and still achieve efficiency
- OS requires support from hardware!
- What kind of support is needed from the hardware?
- CPU privilege levels: user-mode vs. kernel-mode
- Switching between modes, entry points and handlers

Evidence based Proof of LDE!

- “Proof” is a term associated with formal world
- “Evidence based proof” is very important for this course
- Proving LDE in the Linux kernel
    - How to prove using a user program executing an infinite loop?

Evidence based Proof of LDE!

- “Proof” is a term associated with formal world
- “Evidence based proof” is very important for this course
- Proving LDE in the Linux kernel
    - How to prove using a user program executing an infinite loop?
    - CPU usage in user mode should dominate
    - Is this evidence enough?

Evidence based Proof of LDE!

- “Proof” is a term associated with formal world
- “Evidence based proof” is very important for this course
- Proving LDE in the Linux kernel
  - How to prove using a user program executing an infinite loop?
  - CPU usage in user mode should dominate
  - Is this evidence enough? OS intervention yet to be shown!
  - If a program does division by zero infinitely, we can prove OS intervention. How to go about it?

Evidence based Proof of LDE!

- "Proof" is a term associated with formal world
- "Evidence based proof" is very important for this course
- Proving LDE in the Linux kernel
  - How to prove using a user program executing an infinite loop?
  - CPU usage in user mode should dominate
  - Is this evidence enough? OS intervention yet to be shown!
  - If a program does division by zero infinitely, we can prove OS intervention. How to go about it?
  - Handle the signal and ignore it or something better (modify division-by-zero handler in OS) ⇒ First hand evidence