Limited Direct Execution

Glenn Bruns
CSUMB

Lecture Objectives

At the end of this lecture, you should be able to:

- Describe how Linux/Unix supports multi-programming

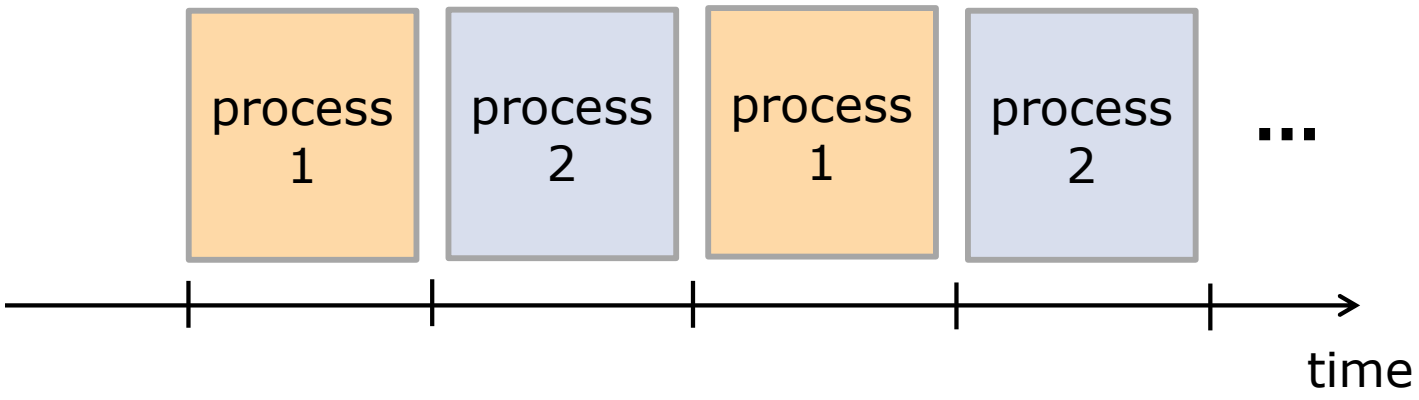
Reminder: our guiding questions

- What are the virtual resources or services offered to users? Are they easy to use?
- How to ensure fair sharing of resources between users?
- How to protect users from each other, and protect the system from users?
- What workloads do we use to measure performance?
- What metrics do we use to measure performance?
- How efficiently are resources managed?

Multi-programming

Multi-programming – allow multiple processes to run "at the same time"

At least give the illusion of it.



Each process runs for a short "time slice" of perhaps 1/10 sec.

Problem

How to start and stop programs?

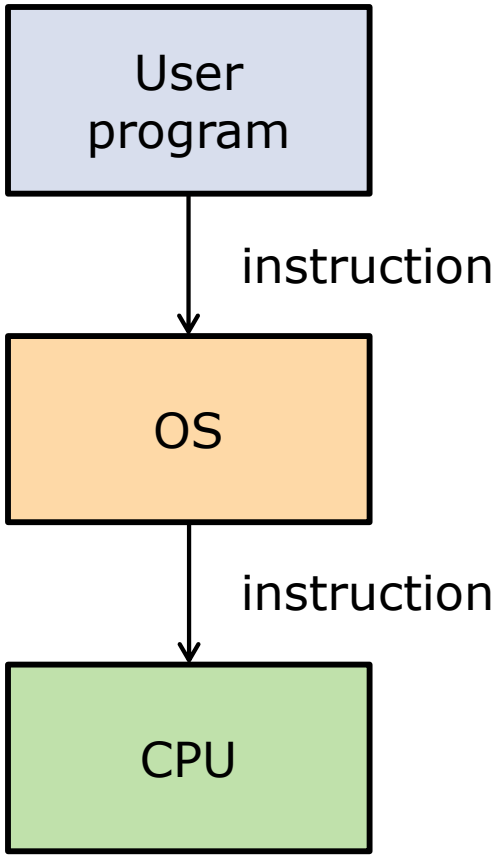
How to keep them from doing things they shouldn't?

Idea 1

OS acts as an interpreter

It can easily switch between programs.

It can easily prevent programs from doing bad things.

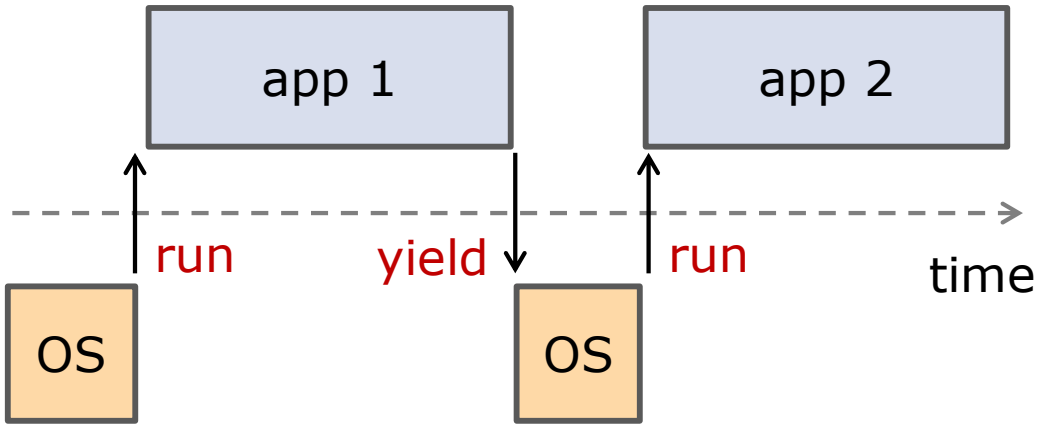


What is the main problem with this idea?

Idea 2

Programs "yield" to OS

When a user program executes the "yield" instruction, control returns to OS.

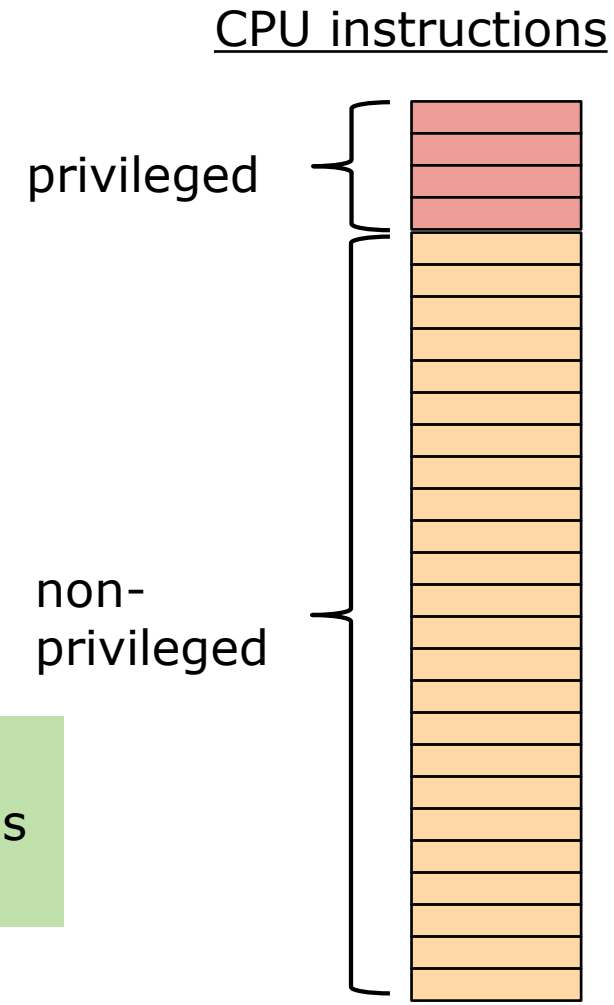


What is the main problem with this idea?

Idea 3

User programs run directly on CPU, but CPU has two modes:

- OS kernel runs in kernel mode: all instructions can be run
- user processes run in user mode: only non-privileged instructions can be run



Example:

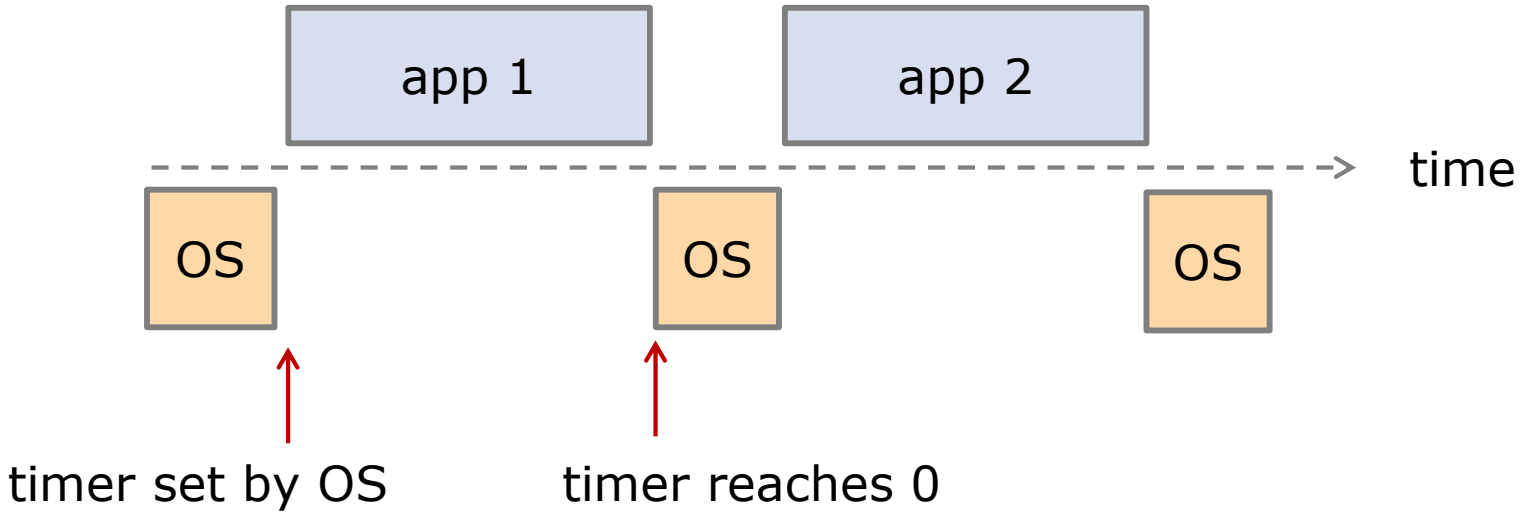
- I/O instructions are privileged instructions
- User programs must ask the OS them

What is the main limitation of this idea?

Idea 4

Stop applications using hardware:

- OS starts timer
- OS starts application
- when timer goes to 0, CPU gets a hardware signal
- CPU then starts running OS



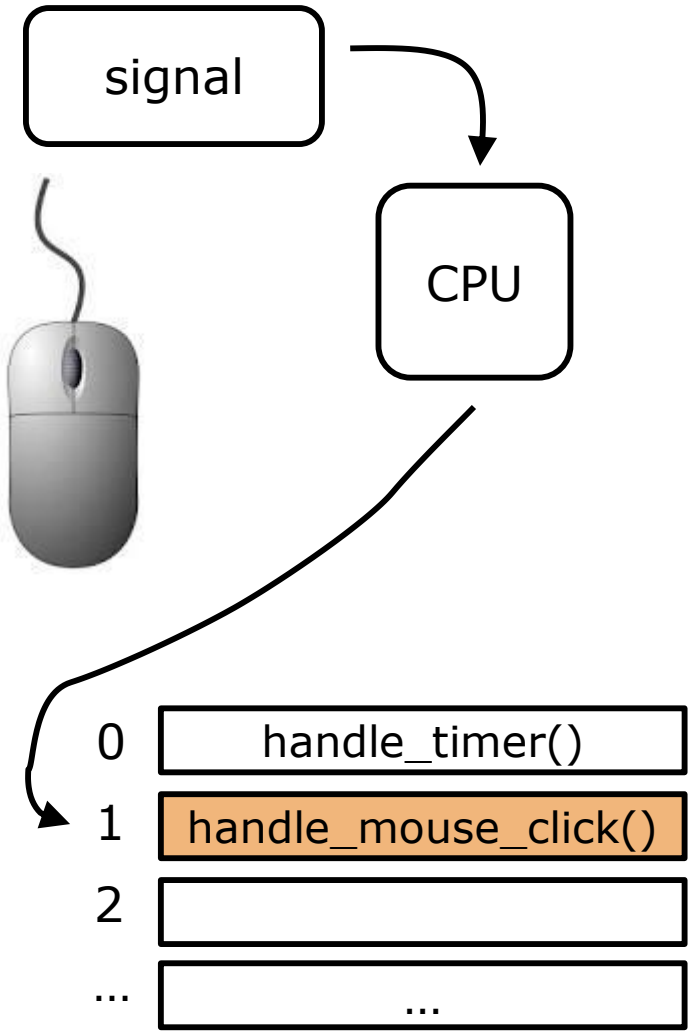
How do timer interrupts works?

When certain hardware events occur, the CPU:

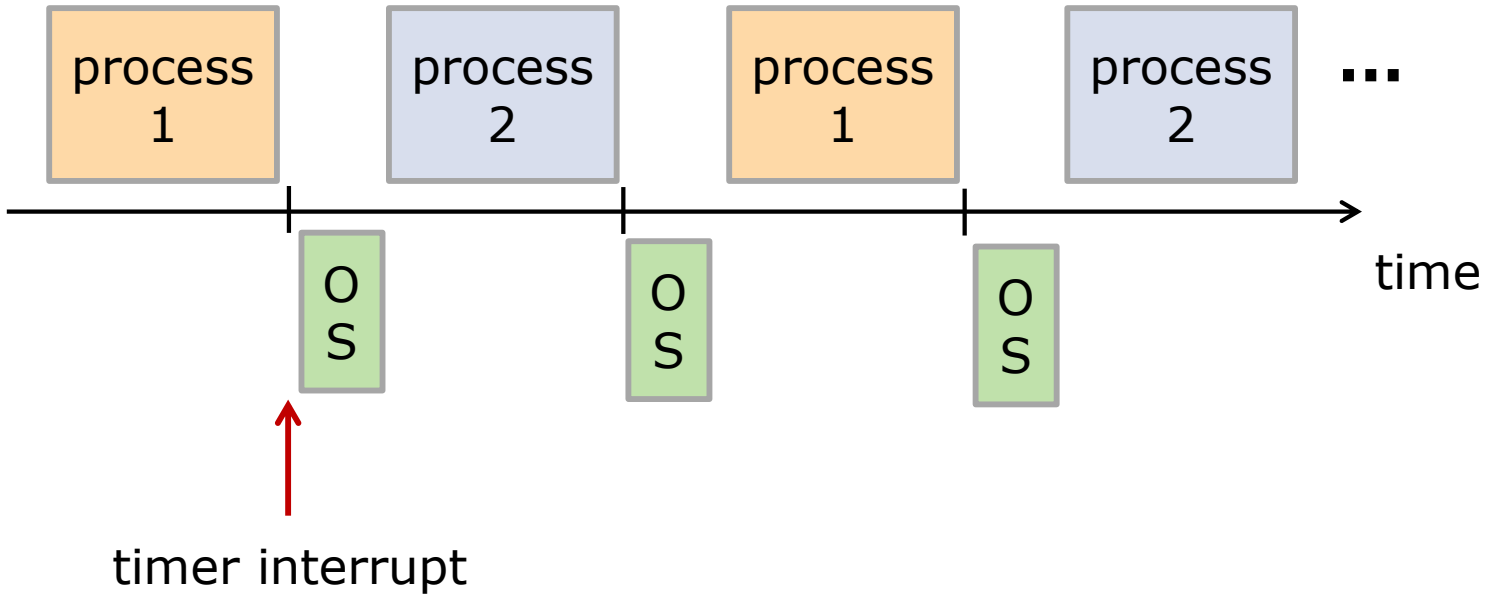
- stops the current process, saves its state, enters kernel mode
- run the event's interrupt handler in an interrupt vector table

When the handler finishes, it performs a special instruction, and the CPU:

- restores the process, enters user mode, and restarts the process



Context switching with timer interrupts



1. hardware (CPU):
   - saves process 1 state
   - switches to kernel mode
   - jumps to interrupt handler

2. OS:
   - saves register values to process 1 proc-struct
   - restore register values from process 2 proc-struct
   - RETURN-FROM-INTR

3. hardware:
   - restores process 2 state
   - switches to user mode
   - restarts process 2

How do processes request OS help?

User processes are run in user mode to provide "protection".

How do processes get the OS to help them – for example to perform I/O?

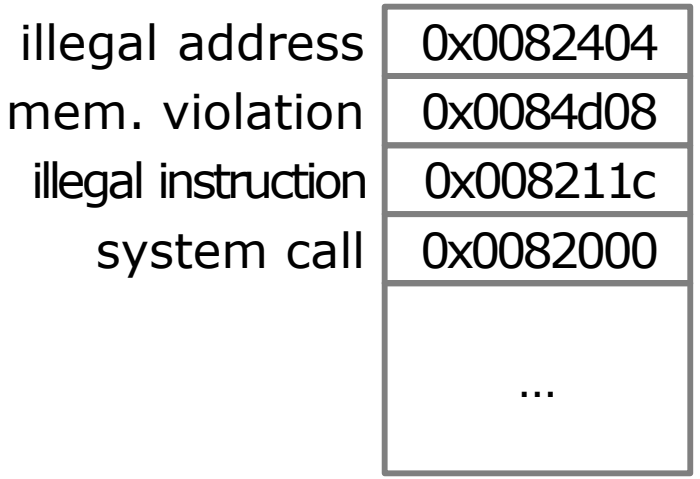
Traps ("software interrupts")

- when a processor exception happens, or a TRAP instruction is run, the OS:
- stops the current process, saves its state, enters kernel mode
- transfers control to a trap handler

When the handler finishes, it performs RET-FROM-TRAP, and the CPU:

- restores process state, enters user mode, and restarts the process

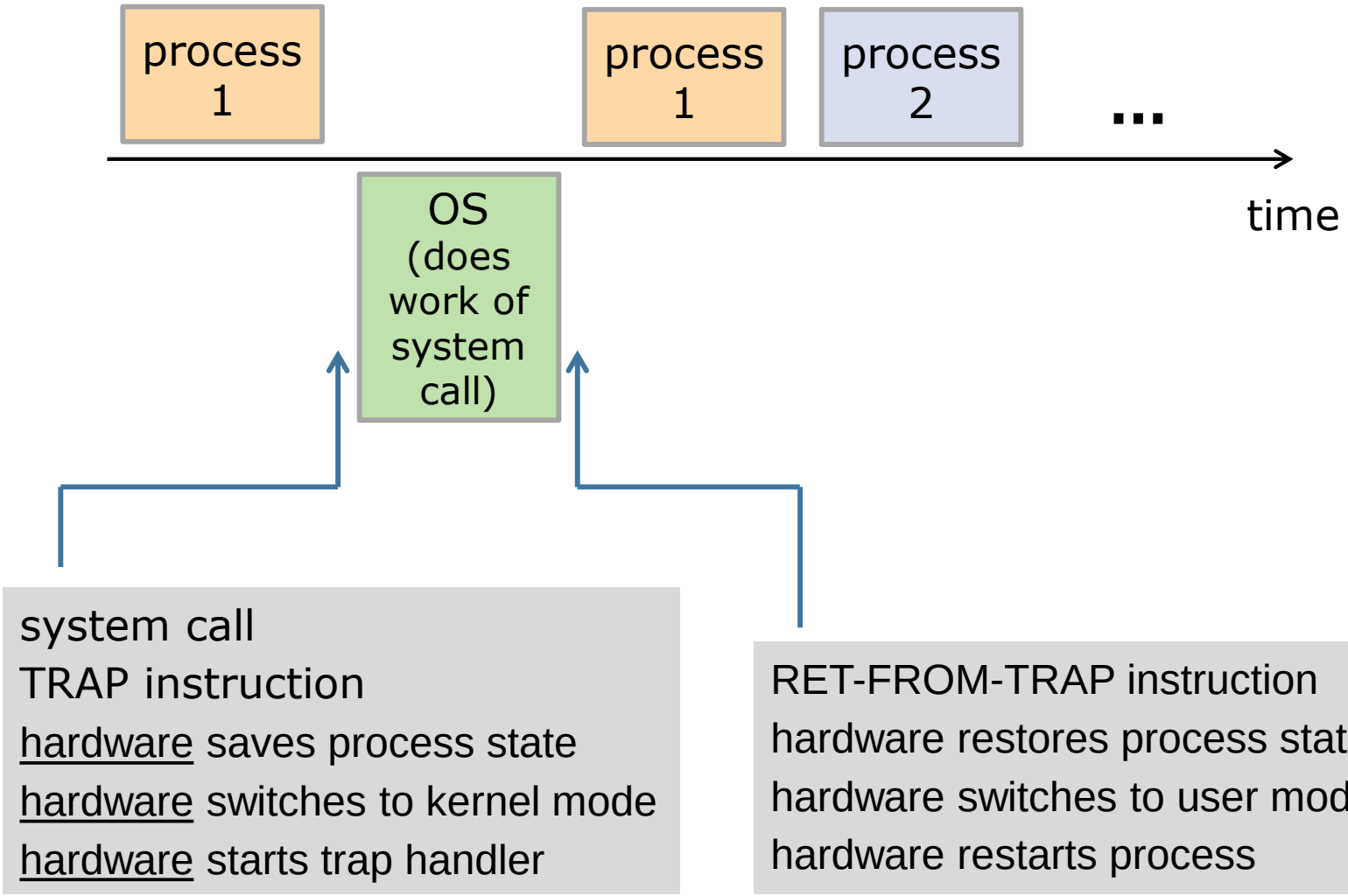
trap handler table



Here, 0x82404 is address of `handle_illegal_addr()`

material on right from: slides, CS5460, Univ. of Utah

Execution with system calls



Details on traps/interrupts

At boot time, OS sets up interrupt handler and trap tables in kernel mode.

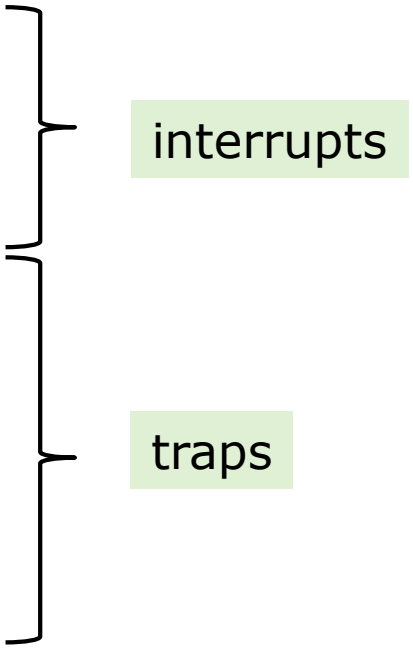
For example, the OS uses an instruction to indicate location of the table.

Question: what if interrupt table could be modified in user mode?

Question: what happens if an interrupt occurs when an interrupt handler is running?

Traps versus Interrupts

- interrupts
  - examples: mouse click, timer, hard drive
- processor exceptions
  - examples: floating-point error, invalid memory access
- system calls
  - examples: read file



Interrupts and traps are handled similarly: through routines in the interrupt vector table.

Traps are synchronous: they are tied to code execution.

Interrupts are asynchronous: they aren't synchronized with the code

Traps are also called software interrupts

Summary

- We've looked at multi-programming using limited direct execution
- direct because user processes run directly on the CPU
- limited because user processes run in user mode
- Hardware ingredients:
  - kernel/user mode of CPU
  - traps
  - interrupts

Concepts to understand:

- trap
- interrupt
- interrupt handler
- interrupt vector table
- context switching
- multi-programming