Introduction to Operating Systems
CS 1550

Process Synchronization

Spring 2023

Sherif Khattab

ksm73@pitt.edu

(Some slides are from **Silberschatz, Galvin and Gagne ©2013)**

Announcements

- Homework 1 is due this Friday at 11:59 pm
- Recitations start this week
- Steps of a Syscall posted on Canvas
- TA Student Support Hours available on the syllabus page
- Muddiest points are anonymous to all

Muddiest Points

- **How does the OS handle another interrupt coming in while it is in the middle of dealing with another?**
- Some interrupts can be interrupted!
- The same interrupt process occurs, except that the the return-from-interrupt instruction does not necessarily return to user mode

Muddiest Points

- **the functionality of the functions in memory that aren’t f7(), and/or the process of how the IDT array (in memory) is used**
- Some of other functions are Interrupt Service Routines
  - pointed to from IDT table
  - end with return-from-interrupt instruction
- Some other functions are System Call Implementations
  - pointed to from the syscall table
  - end with regular return instruction

Muddiest Points

- **memory-resident virus scenario**
- A boot-loader virus can change the IDT entries and make some or all of them point to the virus code

Muddiest Points

- **I was confused why we do not add an entry to the IDT when adding a syscall**
- The IDT size is limited by hardware
  - We can potentially have too many syscalls

Muddiest Points

- **Understanding the meaning of kernel.**
- The functions and data structures that include:
  - IDT and syscall tables
  - Interrupt Service Routines and Syscall Implementation functions
- Kernel code runs in the privileged kernel mode
- Compare that to the Shell
  - The command-line or GUI interface through which we can issue commands to the system
  - Shell includes support utilities, such as ls, file system check, etc.

Muddiest Points

- **A lot of new information and new words without simple terms to explain it.**
- Sorry about that. I hope things got more clear in the previous lecture.

Muddiest Points

- **if it an expensive operation what does that really mean?**
- Interrupt handling is an expensive process meaning that it takes much more time *compared to* the speed of the CPU
  - CPU does things in nano seconds
    - We now have a 6 GHz CPU?
  - Memory access occurs in microseconds
    - That's ~1/1000 of CPU speed

Muddiest Points

- **Is that what we still use to this day?**
- Yes. The explained interrupt processing steps is essentially that happens in systems today
  - with some optimizations and variations between CPUs

Muddiest Points

- **Also back in the day when keyboards and all of this were new was there a limit on how fast you can type in concern to crashing the OS?**
- Possibly!
- If interrupts happen too fast, the CPU will miss some of them.

Muddiest Points

- **Where is the data from the program counter register saved so we can return to what was occurring before the interrupt took place?**
- To the kernel stack in memory

Muddiest Points

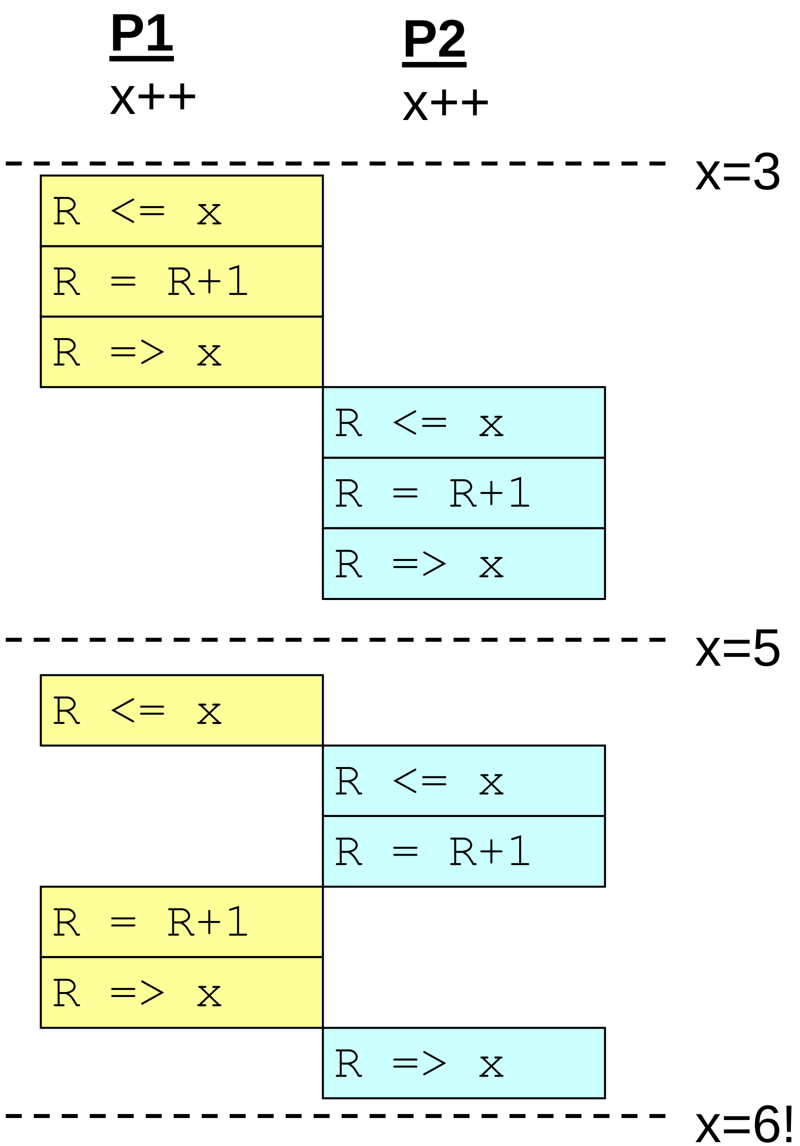
- **how the interrupt vector comes in?**
- The bootup code fills in that table in memory
- We will see that in the XV6 code walkthrough today

Xv6 Code Walkthrough

- IDT table initialization
- Syscall table
- How a syscall is invoked
- Syscall implementation
- Parameter passing into a syscall
- In Lab 1 you will add a system call to Xv6

Problem: race conditions

- R is a CPU register
- X is a variable stored in memory

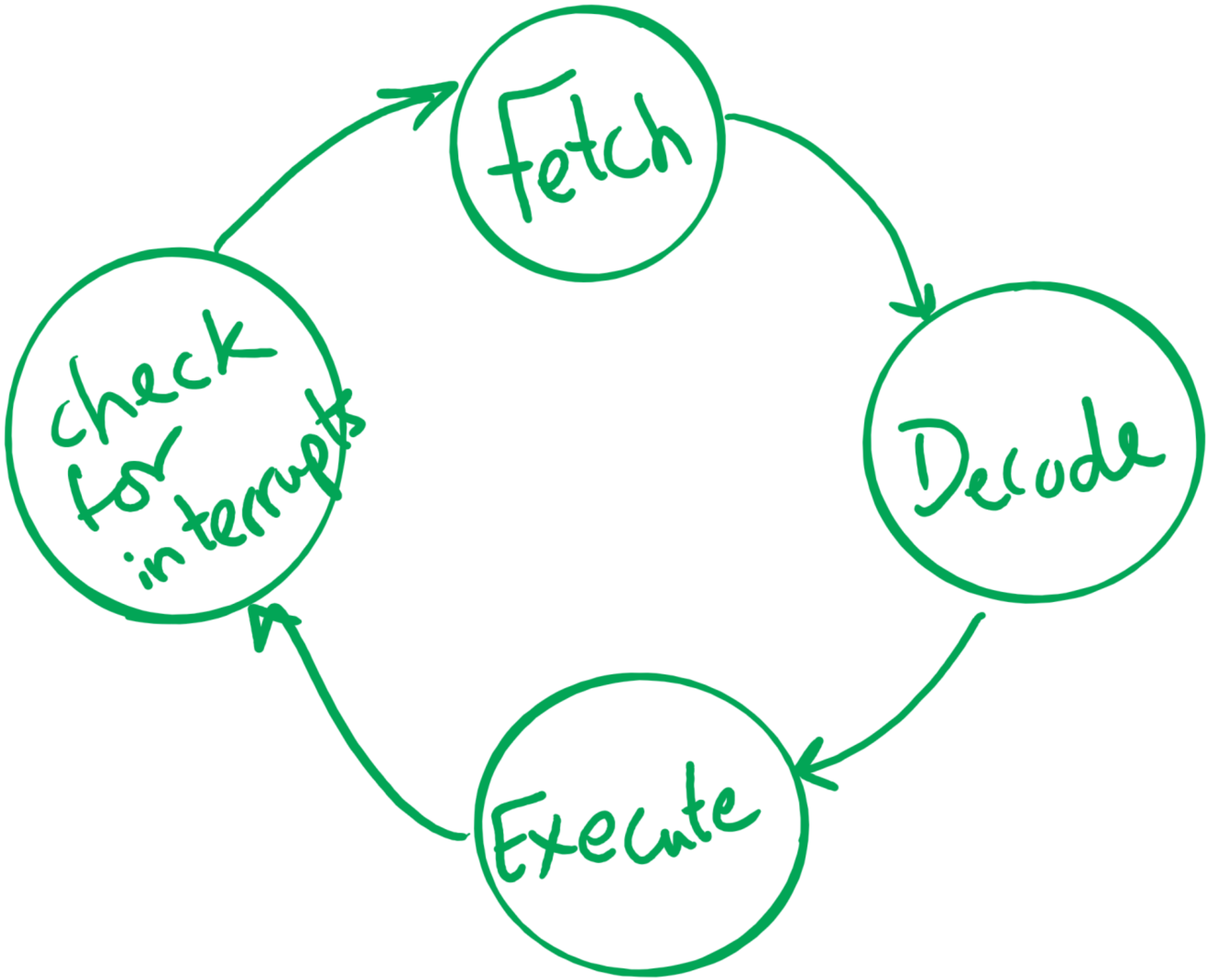


Race conditions

- Cooperating processes share storage (memory)
- Both may read and write the shared memory
- Problem: can't guarantee that read followed by write is **atomic**
  - Atomic means uninterruptible
  - Ordering matters!
- This can result in erroneous results!
- We need to eliminate race conditions…

Atomic operations

- If done in one instruction, then not interruptible



Context Switching

How did the CPU switch from P1 to P2 then to P1 then to P2 again …?

Process Control Block

``Active entities are data structures when viewed from a lower level.''

Raphael Finkel, University of Kentucky

Process Control Block (PCB)

Information associated with each process

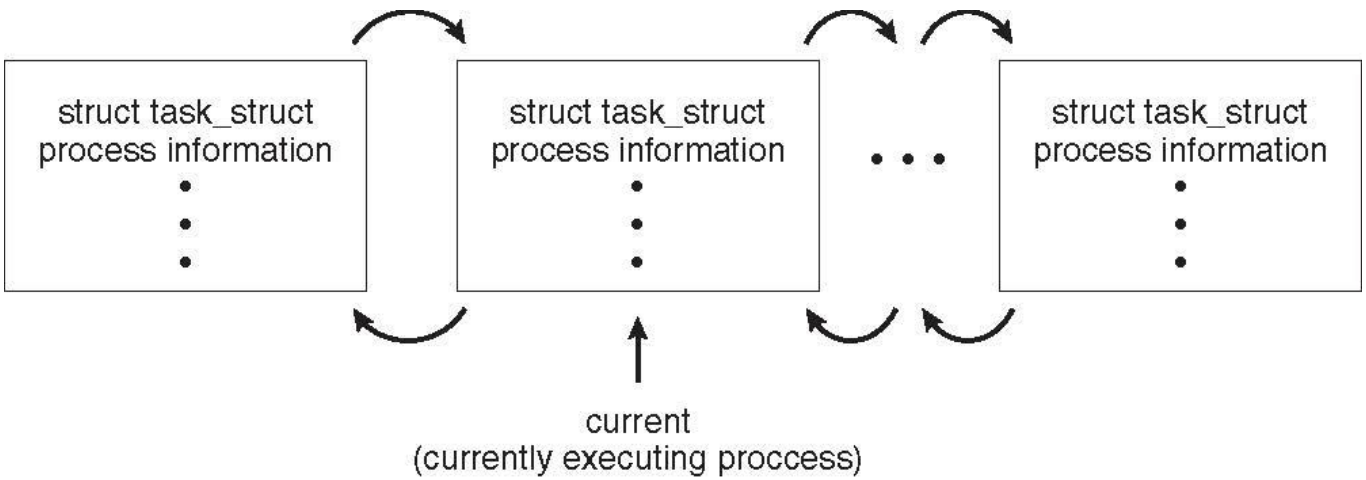
(also called **task control block**)

- Process state – running, waiting, etc
- Program counter – location of instruction to execute next
- CPU registers – contents of all process-centric registers
- CPU scheduling information- priorities, scheduling queue pointers
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files

Process Representation in Linux

Represented by the C structure `task_struct`

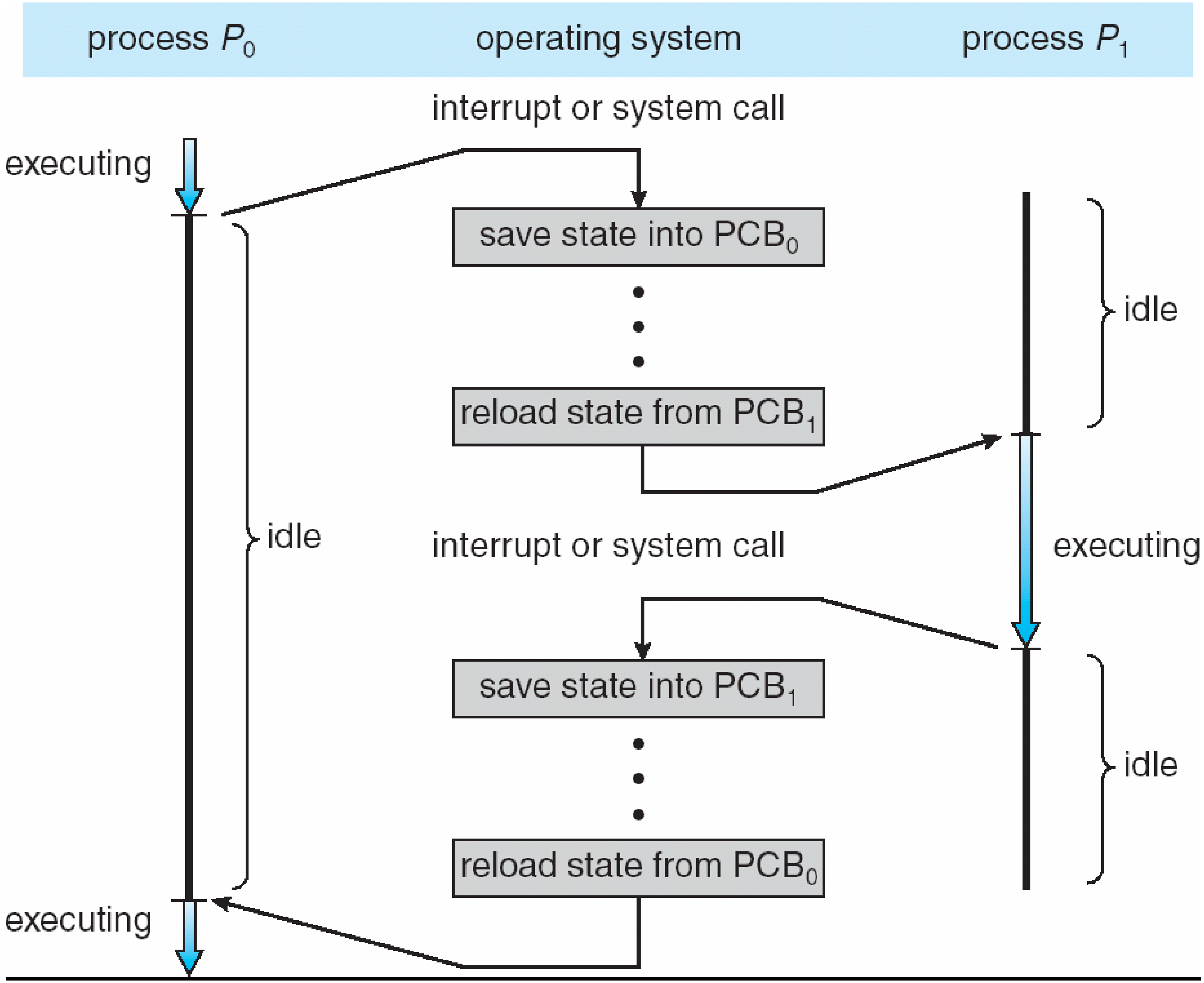
```
pid_t pid; /* process identifier */
long state; /* state of the process */
unsigned int time_slice /* scheduling information */
struct task_struct *parent; /* this process's parent */
struct list_head children; /* this process's children */
struct files_struct *files; /* list of open files */
struct mm_struct *mm; /* address space of this process */
```



Context Switch

- When CPU switches to another process, the system must **save the state** of the old process and load the **saved state** for the new process via a **context switch**
- **Context** of a process represented in the PCB
- Context-switch time is overhead; the system does no useful work while switching
  - The more complex the OS and the PCB ➔ the longer the context switch
- Time dependent on hardware support
  - Some hardware provides multiple sets of registers per CPU ➔ multiple contexts loaded at once

Context Switching



Xv6 Code Walkthrough

- PCB and process table
- Context switching
- Calling of the swtch routine