ILLINOIS TECH

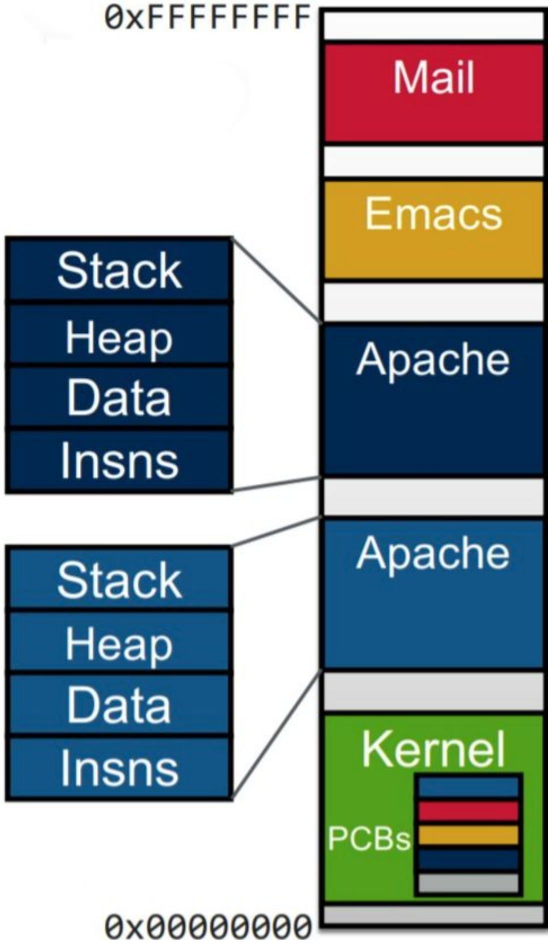
College of Computing

CS 450 Operating Systems Threads

Yue Duan

Consider..

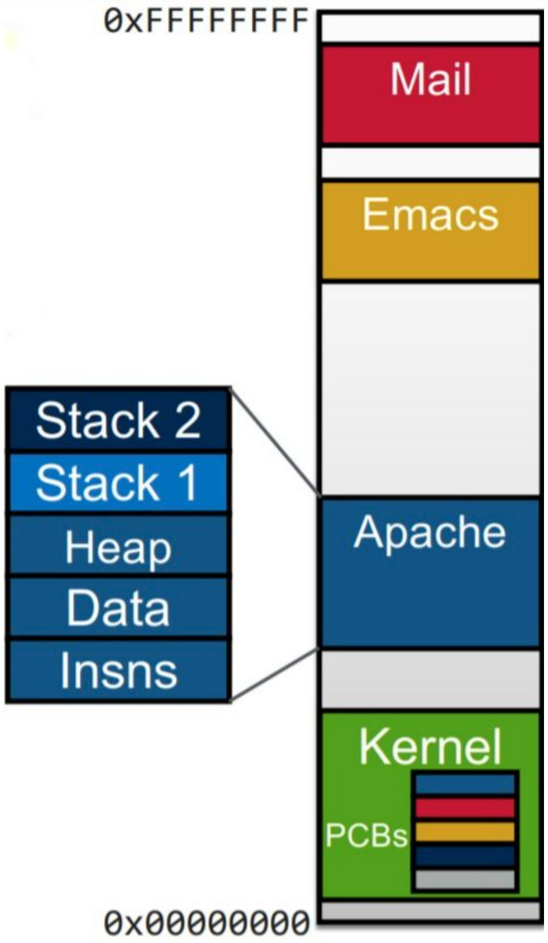
- Apache wants to run multiple concurrent computations
- Two heavyweight address spaces for two concurrent computations
- Hard to share cache, etc.



- *Physical address space*
- *Each process' address space by color (shown contiguous to look nicer)*

Improvement idea

- Place concurrent computations in the same address space!



Thread

- Also known as
  - Lightweight Process
  - Thread of Control
  - Task
- Major differences between thread and process:
  - lightweight vs. heavyweight
  - share data vs. isolation

Process vs. Thread

- A process is an abstraction of a computer
  - CPU, memory, devices
- A thread is an abstraction of a core
  - registers (incl. PC and SP)

Unbounded #computers, each with unbounded #cores

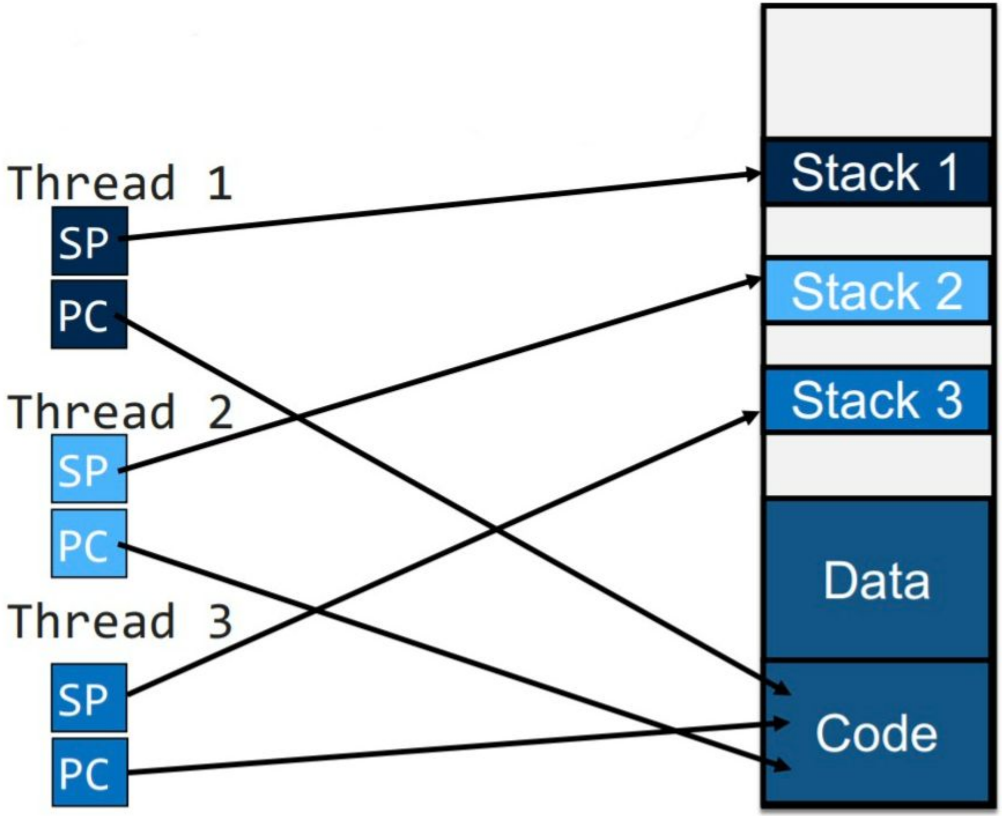
- Different processes typically have their own (virtual) memory, but different threads share virtual memory.
- Different processes tend to be mutually distrusting, but threads must be mutually trusting. **Why?**

Process vs. Thread

- **Multiple threads** within a single process **share**:
  - Address space
    - Code (instructions)
    - Most data (heap)
  - Open file descriptors
  - Current working directory
  - User and group id
- Each thread has its own
  - Thread ID (TID)
  - Set of registers, including Program counter and Stack pointer
  - Stack for local variables and return addresses (in same address space)

Virtual Memory Layout

- Thread stacks are allocated on the heap!



Why Threads?

- Concurrency
  - exploiting multiple CPUs/cores
- Mask long latency of I/O
  - doing useful work while waiting
- Responsiveness
  - high priority GUI threads / low priority work threads
- Encourages natural program structure
  - expressing logically concurrent tasks
  - update screen, fetching data, receive user input

Example

- Web server:
  - 1. get network message (URL) from client
  - 2. get URL data from disk
  - 3. compose response
  - 4. send response

```
for (k = 0; k < n; k++) {
    a[k] = b[k] × c[k] + d[k] × e[k]
}
```

Simple Thread API

`void thread_create (func,arg)`	Creates a new thread that will execute function **func** with the arguments **arg**
`void thread_yield()`	Calling thread gives up processor. Scheduler can resume running this thread at any point.
`void thread_exit()`	Finish caller

Preemption

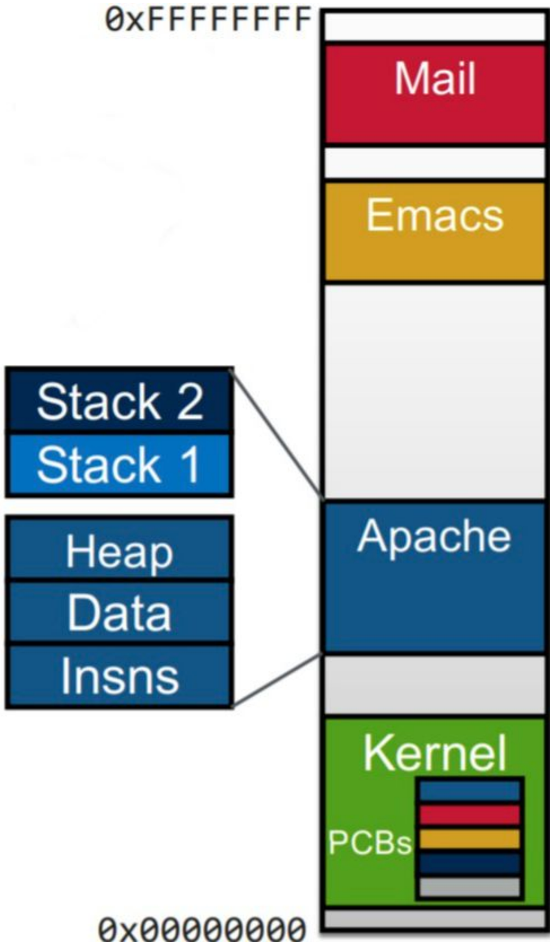
- Two kinds of threads:
  - Non-preemptive:
    - explicitly yield to other threads
  - Preemptive:
    - yield automatically upon clock interrupts
- Most modern threading systems are preemptive

Implementation of Threads

- One abstraction, two implementations:
- 1. “kernel threads”:
  - each thread has its own PCB in the kernel, but the PCBs point to the same physical memory
- 2. “user threads”:
  - one PCB for the process; threads implemented entirely in user space.
  - each thread has its own Thread Control Block (TCB)

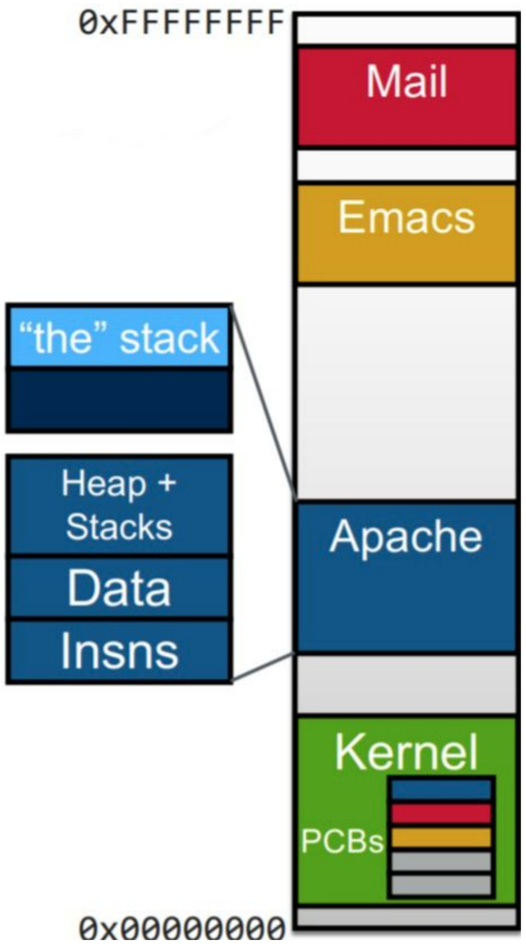
Kernel-Level Threads

- Kernel knows about, schedules threads (just like processes)
- Separate PCB for each thread
- PCBs have:
  - **same**: page table base register
  - **different**: PC, SP, registers, interrupt stack



User-Level Threads

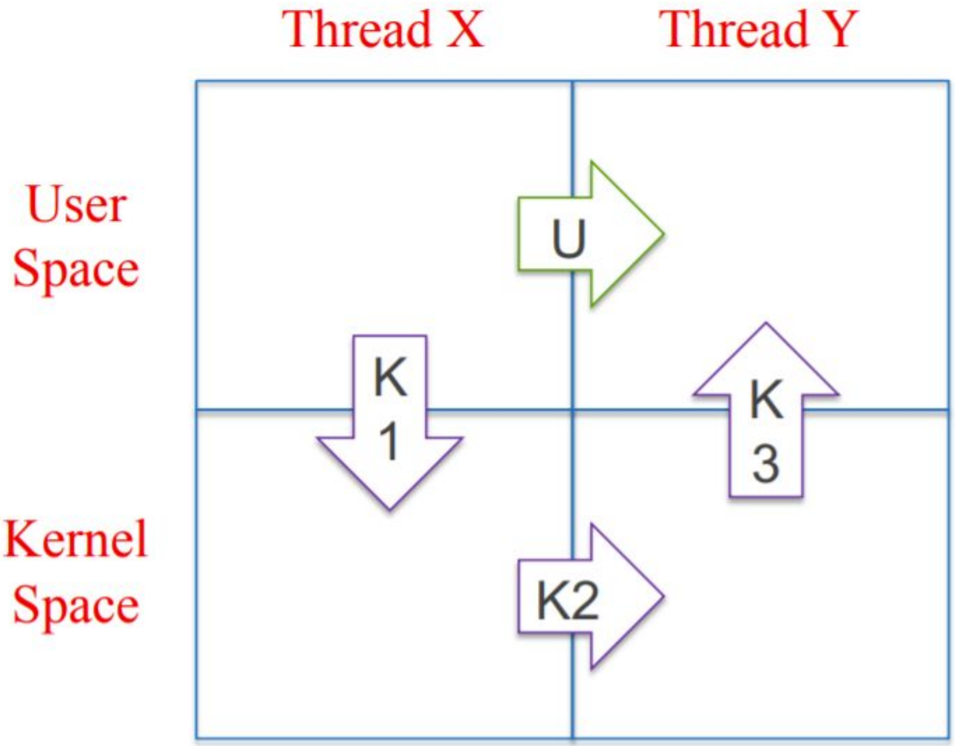
- Run mini-OS in user space
  - Real OS unaware of threads
  - Single PCB
  - Thread Control Block (TCB) for each thread
- Generally more efficient than kernel-level threads (**Why**?)
- But kernel-level threads simplify system call handling and scheduling (**Why**?)



Kernel- vs User-level Threads

Kernel-Level Threads	User-level Threads
• Easy to implement: just processes with shared page table	• Requires user-level context switches, scheduler
• Threads can run blocking system calls concurrently	• Blocking system call blocks all threads: needs O.S. support for non-blocking system calls
• Thread switch requires three context switches	• Thread switch efficiently implemented in user space

Kernel vs User Thread Switch



Thread Schedule 1

- balance = balance + 1; //balance at 0x9cd4

State:
0x9cd4: 100
%eax: ?
%rip = 0x195

process control blocks:

Thread 1	Thread 2
%eax: ? %rip: 0x195	%eax: ? %rip: 0x195

T1 →

```
0x195  mov 0x9cd4, %eax
0x19a  add $0x1, %eax
0x19d  mov %eax, 0x9cd4
```

Thread Schedule 1

- balance = balance + 1; //balance at 0x9cd4

State:
0x9cd4: 100
%eax: 100
%rip = 0x19a

process control blocks:

Thread 1	Thread 2
%eax: ?	%eax: ?
%rip: 0x195	%rip: 0x195

```
      0x195  mov 0x9cd4, %eax
T1 →  0x19a  add $0x1, %eax
      0x19d  mov %eax, 0x9cd4
```

Thread Schedule 1

- balance = balance + 1; //balance at 0x9cd4

State:
0x9cd4: 100
%eax: 101
%rip = 0x19d

process control blocks:

	Thread 1	Thread 2
%eax	?	?
%rip	0x195	0x195

```
   0x195  mov 0x9cd4, %eax
   0x19a  add $0x1, %eax
T1 0x19d  mov %eax, 0x9cd4
```

Thread Schedule 1

- balance = balance + 1; //balance at 0x9cd4

State:
0x9cd4: 101
%eax: 101
%rip = 0x1a2

process control blocks:

	Thread 1	Thread 2
	%eax: ?	%eax: ?
	%rip: 0x195	%rip: 0x195

```
0x195  mov 0x9cd4, %eax
0x19a  add $0x1, %eax
0x19d  mov %eax, 0x9cd4
```

T1

Thread Context Switch

Thread Schedule 1

- balance = balance + 1; //balance at 0x9cd4

State:
0x9cd4: 101
%eax: ?
%rip = 0x195

process control blocks:

Thread 1	Thread 2
%eax: 101 %rip: 0x1a2	%eax: ? %rip: 0x195

T2

```
0x195  mov 0x9cd4, %eax
0x19a  add $0x1, %eax
0x19d  mov %eax, 0x9cd4
```

Thread Schedule 1

- balance = balance + 1; //balance at 0x9cd4

State:
0x9cd4: 101
%eax: 101
%rip = 0x19a

process control blocks:

Thread 1	Thread 2
%eax: 101 %rip: 0x1a2	%eax: ? %rip: 0x195

```
     0x195  mov 0x9cd4, %eax
T2 → 0x19a  add $0x1, %eax
     0x19d  mov %eax, 0x9cd4
```

Thread Schedule 1

- balance = balance + 1; //balance at 0x9cd4

State:
0x9cd4: 101
%eax: 102
%rip = 0x19d

process control blocks:

Thread 1	Thread 2
%eax: 101 %rip: 0x1a2	%eax: ? %rip: 0x195

```
      0x195  mov 0x9cd4, %eax
      0x19a  add $0x1, %eax
T2 →  0x19d  mov %eax, 0x9cd4
```

Thread Schedule 1

- balance = balance + 1; //balance at 0x9cd4

State:
0x9cd4: **102**
%eax: 102
%rip = 0x1a2

process control blocks:

Thread 1	Thread 2
%eax: 101 %rip: 0x1a2	%eax: ? %rip: 0x195

```
0x195  mov 0x9cd4, %eax
0x19a  add $0x1, %eax
0x19d  mov %eax, 0x9cd4
```

T2

Desired result!

Thread Schedule 2

- balance = balance + 1; //balance at 0x9cd4

State:
0x9cd4: 100
%eax: ?
%rip = 0x195

process control blocks:

Thread 1	Thread 2
%eax: ? %rip: 0x195	%eax: ? %rip: 0x195

T1

```
0x195  mov 0x9cd4, %eax
0x19a  add $0x1, %eax
0x19d  mov %eax, 0x9cd4
```

Thread Schedule 2

- balance = balance + 1; //balance at 0x9cd4

State:
0x9cd4: 100
%eax: 100
%rip = 0x19a

process control blocks:

Thread 1	Thread 2
%eax: ?	%eax: ?
%rip: 0x195	%rip: 0x195

```
     0x195  mov 0x9cd4, %eax
T1   0x19a  add $0x1, %eax
     0x19d  mov %eax, 0x9cd4
```

Thread Schedule 2

- balance = balance + 1; //balance at 0x9cd4

State:
0x9cd4: 100
%eax: 101
%rip = 0x19d

process control blocks:

Thread 1	Thread 2
%eax: ?	%eax: ?
%rip: 0x195	%rip: 0x195

```
     0x195  mov 0x9cd4, %eax
     0x19a  add $0x1, %eax
T1 → 0x19d  mov %eax, 0x9cd4
```

Thread Context Switch

Thread Schedule 2

- balance = balance + 1; //balance at 0x9cd4

State:
0x9cd4: 100
%eax: ?
%rip = 0x195

process control blocks:

	Thread 1	Thread 2
%eax	101	?
%rip	0x19d	0x195

```
T2 →  0x195  mov 0x9cd4, %eax
      0x19a  add $0x1, %eax
      0x19d  mov %eax, 0x9cd4
```

Thread Schedule 2

- balance = balance + 1; //balance at 0x9cd4

State:
0x9cd4: 100
%eax: 100
%rip = 0x19a

process control blocks:

Thread 1	Thread 2
%eax: 101 %rip: 0x19d	%eax: ? %rip: 0x195

T2

```
0x195  mov 0x9cd4, %eax
0x19a  add $0x1, %eax
0x19d  mov %eax, 0x9cd4
```

Thread Schedule 2

- balance = balance + 1; //balance at 0x9cd4

State:
0x9cd4: 100
%eax: 101
%rip = 0x19d

process control blocks:

Thread 1	Thread 2
%eax: 101	%eax: ?
%rip: 0x19d	%rip: 0x195

```
     0x195  mov 0x9cd4, %eax
     0x19a  add $0x1, %eax
T2   0x19d  mov %eax, 0x9cd4
```

Thread Schedule 2

- balance = balance + 1; //balance at 0x9cd4

State:
0x9cd4: 101
%eax: 101
%rip = 0x1a2

process control blocks:

Thread 1	Thread 2
%eax: 101 %rip: 0x19d	%eax: ? %rip: 0x195

```
0x195  mov 0x9cd4, %eax
0x19a  add $0x1, %eax
0x19d  mov %eax, 0x9cd4
```

T2

Thread Context Switch

Thread Schedule 2

- balance = balance + 1; //balance at 0x9cd4

State:
0x9cd4: 101
%eax: 101
%rip = 0x19d

process control blocks:

Thread 1	Thread 2
%eax: 101 %rip: 0x19d	%eax: 101 %rip: 0x1a2

```
   0x195  mov 0x9cd4, %eax
   0x19a  add $0x1, %eax
T1 0x19d  mov %eax, 0x9cd4
```

Thread Schedule 2

- balance = balance + 1; //balance at 0x9cd4

State:
0x9cd4: 101
%eax: 101
%rip = 0x1a2

process control blocks:

Thread 1	Thread 2
%eax: 101 %rip: 0x1a2	%eax: 101 %rip: 0x1a2

```
0x195  mov 0x9cd4, %eax
0x19a  add $0x1, %eax
0x19d  mov %eax, 0x9cd4
```

T1

WRONG RESULT! Final balance value is 101

Timeline View: Interleaving 1

Thread 1

```
mov 0x123, %eax
add %0x1, %eax
mov %eax, 0x123
```

Thread 2

```
mov 0x123, %eax
add %0x2  %eax
mov %eax, 0x123
```

time

correct!

Timeline View: Interleaving 2

Thread 1	Thread 2
`mov 0x123, %eax` `add %0x1, %eax`	
	`mov 0x123, %eax`
`mov %eax, 0x123`	
	`add %0x2, %eax` `mov %eax, 0x123`

time

incorrect!

Timeline View: Interleaving 3

Thread 1	Thread 2
	`mov 0x123, %eax`
`mov 0x123, %eax`	
	`add %0x2, %eax`
`add %0x1, %eax`	
	`mov %eax, 0x123`
`mov %eax, 0x123`	

time

incorrect!

Timeline View: Interleaving 4

Thread 1	Thread 2
	`mov 0x123, %eax`
	`add %0x2, %eax`
	`mov %eax, 0x123`
`mov 0x123, %eax`	
`add %0x1, %eax`	
`mov %eax, 0x123`	

time

correct!

Timeline View: Interleaving 5

Thread 1	Thread 2
	`mov 0x123, %eax`
	`add %0x2, %eax`
`mov 0x123, %eax`	
`add %0x1, %eax`	
`mov %eax, 0x123`	
	`mov %eax, 0x123`

time

incorrect!

Non-Determinism

- Concurrency leads to non-deterministic results
  - Not deterministic result: different results even with same inputs
  - **race conditions**

What do we want?

- Want 3 instructions to execute as an uninterruptible group
- That is, we want them to be an **atomic** unit

```
mov 0x123, %eax
add %0x1, %eax
mov %eax, 0x123
```

critical section

- More general:
  - Need mutual exclusion for critical sections
  - if process A is in critical section C, process B can't be (okay if other processes do unrelated work)

THANK YOU!