Announcements/Reminders

- Topics poll is open – please vote today or tomorrow
- HW1 will be out tonight. Due in a week. Please check class website. Please work on homework independently.
- Please remember to form an MP group
- Please fill out the MP group signup sheet (see post @29 on piazza) to let us know who is in your group
- TA office hours are now posted on the webpage
Implementing Multiple Threads (on One Processor)

- Threads provide an illusion of concurrency
  - Given one “real CPU”, give the illusion of multiple “virtual CPUs” each dedicated to a thread
  - The “virtual CPUs” transparently share the “real CPU”
- The abstraction is recursive
  - If the virtual CPU abstraction is perfect, a “virtual CPU” is indistinguishable from a “real CPU”
  - Therefore, a “virtual CPU” can play the role of a real CPU and can give the illusion of multiple (2nd-level) “virtual CPUs” transparently sharing the 1st-level virtual CPU.

What’s a “Real CPU”?

- Code
- Segment
- Offset
- Program
- Counter

Program instructions
What’s a “Real CPU”? 

Program Counter 
Offset 
Program Counter 
Offset 
OpCode 
Operand 
Current Instruction 
Offset 
Data Operand 
Heap 

Program instructions 

Stack Segment 
Offset 
Stack Pointer 
Stack 

Program instructions 

Heap 

Stack
What’s a “Real CPU”? 

What’s the *State* of a “Real CPU”?
What’s the *State* of a “Real CPU”? 

The Context Switch

Save State (Context)

Load State (Context)
Side Note
The Context Switch

In thread context switches, heap is not switched

Save State (Context)

Load State (Context)

Save State (Context)
Side Note
The Context Switch

Who does the context switch and when?

In thread context switches, heap is not switched

Solution 1: An Interrupt

Save State (Context)
Context Switch (Interrupt)

Interrupt
Save PC on thread stack
Jump to Interrupt handler
Context Switch (Interrupt)

Save PC on thread stack
Jump to Interrupt handler

Handler
- Save thread state in thread control block
  (SP, registers, segment pointers, ...)

- Choose next thread
Context Switch (Interrupt)

- Save thread state in thread control block (SP, registers, segment pointers, ...)
- Choose next thread
- Load thread state from control block

Thread Control Block
Context Switch (Interrupt)

Save PC on thread stack
Jump to Interrupt handler

Handler:
- Save thread state in thread control block (SP, registers, segment pointers, ...)
- Choose next thread
- Load thread state from control block
- Pop PC from thread stack (return from handler)

Where does it return?
In thread context switches, heap is not switched.

Solution 2: Voluntary yield()

Save State (Context)

Load State (Context)

Context Switch (Yield)
Context Switch (Yield)

yield()
- Save thread state in thread control block
  (SP, registers, segment pointers, ...)

Thread Control Block

Save PC on thread stack
Jump to yield() function
Context Switch (Yield)

**Thread Control Block**

Save PC on thread stack
Jump to yield() function

yield() - Save thread state in thread control block
(SP, registers, segment pointers, ...)
- Choose next thread

**Thread Control Block**

Save PC on thread stack
Jump to yield() function

yield() - Save thread state in thread control block
(SP, registers, segment pointers, ...)
- Choose next thread
- Load thread state from control block
Context Switch (Yield)

Save PC on thread stack
Jump to yield() function

yield()
- Save thread state in thread control block
  (SP, registers, segment pointers, ...)
- Choose next thread
- Load thread state from control block
- Pop PC from thread stack (return from handler)

Thread Control Block

Linux User Level Implementation

Save PC on thread stack
Jump to yield() function

yield()
- Choose next thread
- swapcontext()

Thread Control Block
Save PC on thread stack
Jump to yield() function

yield()
- Choose next thread
- swapcontext()

Thread Control Block

Where is the Scheduling Policy?

Maintains a sorted queue of ready threads
Issues

- Initialization?
- Interrupt context switch versus yield?

- Stack for each thread needs to be allocated a priori \(\rightarrow\) just `malloc()` the right sized chunk.
  - Thread control blocks must be created
- Interrupt context switch versus yield?
Issues

- Initialization?
  - Stack for each thread needs to be allocated a priori → just malloc() the right sized chunk.
  - Thread control blocks must be created
- Interrupt context switch versus yield?
  - Yield is voluntary. What if application did not yield?
  - Yield cannot do preemptive scheduling.

A User-level Thread Package

- Initialization?
  - Stack for each thread needs to be allocated a priori → just malloc() the right sized chunk.
  - Thread control blocks must be created
- Interrupt context switch versus yield?
  - Yield is voluntary. What if application did not yield?
  - Yield cannot do preemptive scheduling.
A User-level Thread Package

- Initialization?
  - Stack for each thread needs to be allocated a priori → just malloc() the right sized chunk.
  - Thread control blocks must be created
- Interrupt context switch versus yield?
  - Yield is voluntary. What if application did not yield?
  - Yield cannot do preemptive scheduling.

```
thread_create()
```

```
thread_yield()
```

```
scheduler()
```

Not seen by user

Review

- How do you implement a context switch?
  - In cooperative user-level threads?
  - In kernel threads?
- How do you implement a thread scheduler?
- Trade-offs between cooperative and non-cooperative thread models?