CS 251: Thread Implementation

Sam King

Administrative

- Emphasis for the next quiz is on condition variables and monitors

Writing an intro

- Many different ways to do this
  - Usually do intro and related work at the same time
  - Amount of related work at beginning is a tradeoff

Four to five main sections of an intro

- (1-2 par) What problem are you solving and why is it important. You need to have **citations to backup claims of importance**!
- (1-2 par) How other people solve this problem and why they fall short
- (1 par, optional) What is hard about this problem
- (1-2 par) How you solve it and why your approach is better
- (1 par) Summarize results (these will be *anticipated* results for the proposal)

Proposal == intro + a plan

- Intro -- the bulk of your grade and time
- Want you to start thinking about what this research will take
  - Timeline of what you plan to accomplish, decompose project into smaller tasks
  - Anticipated results
  - Evaluation plan
- **Your proposal must have: intro, timeline, anticipated results, and eval plan**.

Proposal grading: expect scrutiny

- I’m not going to regrade or allow resubmissions, please spend time on getting this right the first time

Proposal

- Similar to the intro of a paper, but some key differences
  - Intro -- usually write after you know the results. Proposal, write ahead of time
  - Can sometimes write ahead of time
- Proposal is more like the intro for a grant proposal
  - Have to show that you have a good problem and that you understand the area
  - Speculating on the work that needs to be done, judged on having a reasonable guess
- Must use LaTeX template that we provide!!!

Proposal hints

- Think big, propose something bold
  - Have a well thought out contingency plan
- Most of your time will be spent on the first 2-4 paragraphs
  - Understand and motivate the problem
    - Citations!!!!
  - Understand the problem space and where you fit
    - Classify other approaches
  - Related work that you read but doesn’t fit in the intro forms your related work section

Writing hints

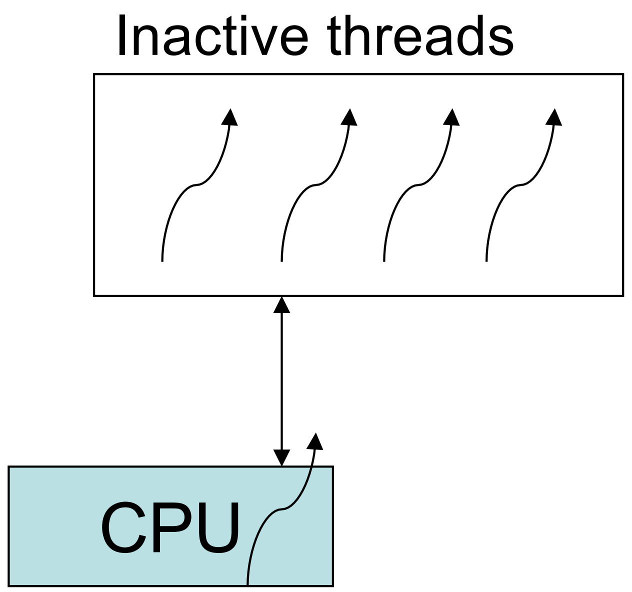
- Good topic sentences
- Careful use of emphasis (e.g., lists, italics)
- Don’t cite anything from Wikipedia!!!!
  - You will get an automatic 0 if you do
  - Do use wikipedia to help find the primary source

Thread impl. on uni-proc.

- So far, we’ve been assuming that we have enough physical process to run each thread on its own processor
  – But threads are useful also for running when you have more threads than CPUs (web server example)
  – How to give the illusion of infinite physical processors on a finite set of processors?

Ready threads

- What to do with thread while it's not running
  - Must save private state somewhere
  - What constitutes private data for a thread?



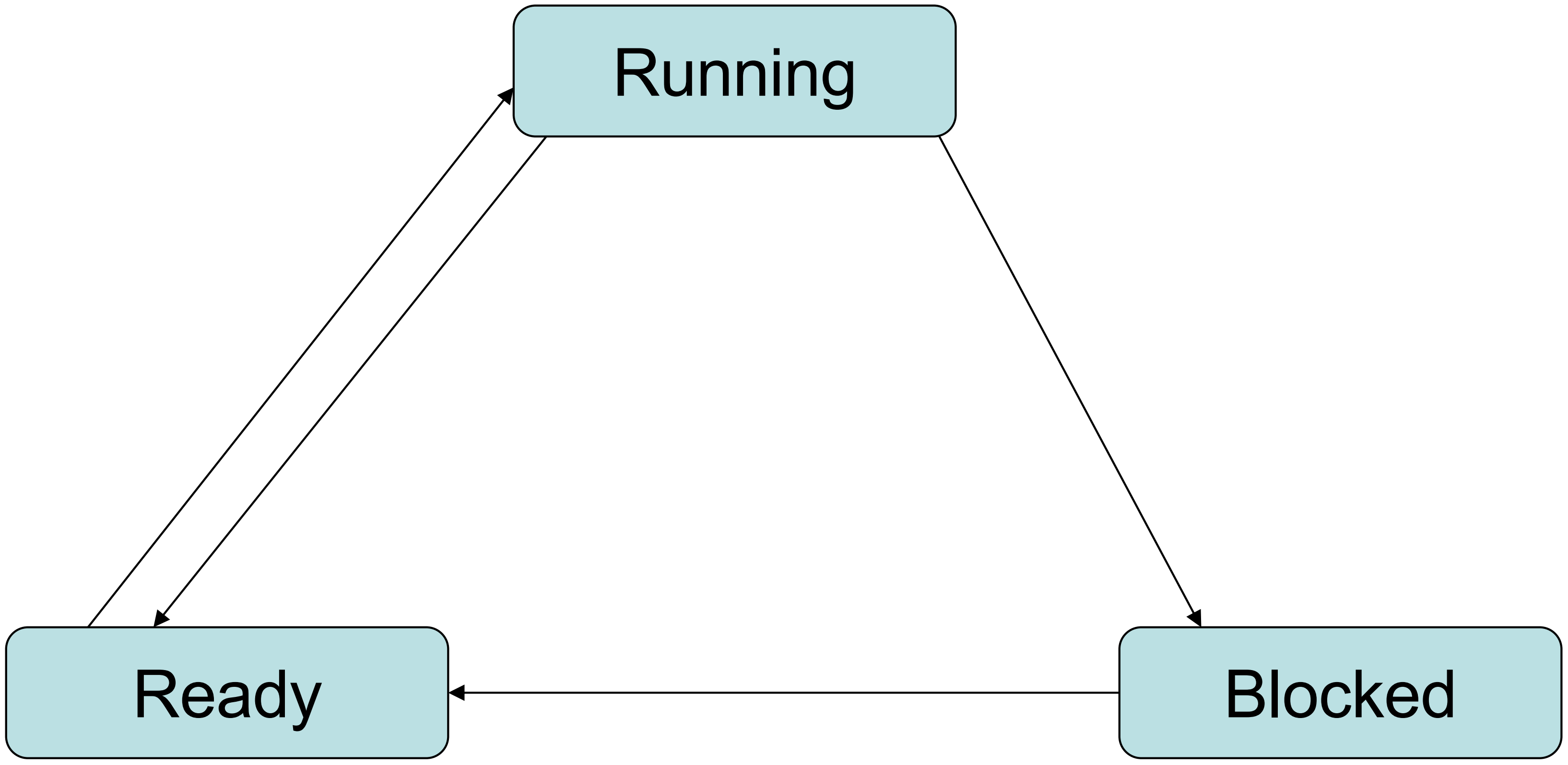
Thread context

- This information is called the thread “context” and is stored in a “thread control block” when the thread isn’t running
  - To save space, share code among all threads
  - To save space, don’t copy stack to the thread control block.
    - Multiple stacks in same address space, copy stack pointer in thread control block

Thread context

- Keep thread control blocks for threads that aren’t running on a queue of **ready** threads
  - Thread state can now be running, ready, or blocked

Thread states



Switching threads

- Steps to switch to another thread
  - Thread returns control to the OS
  - Choose new thread to run
  - Save state of current thread
  - Load context of the next thread
  - Run thread

Returning control to the OS

- Come up with a list of ways for a thread to switch to the OS

Returning control to OS

- How does thread return control back to the OS (so system can save state of current thread and run new one)?

Returning control to OS

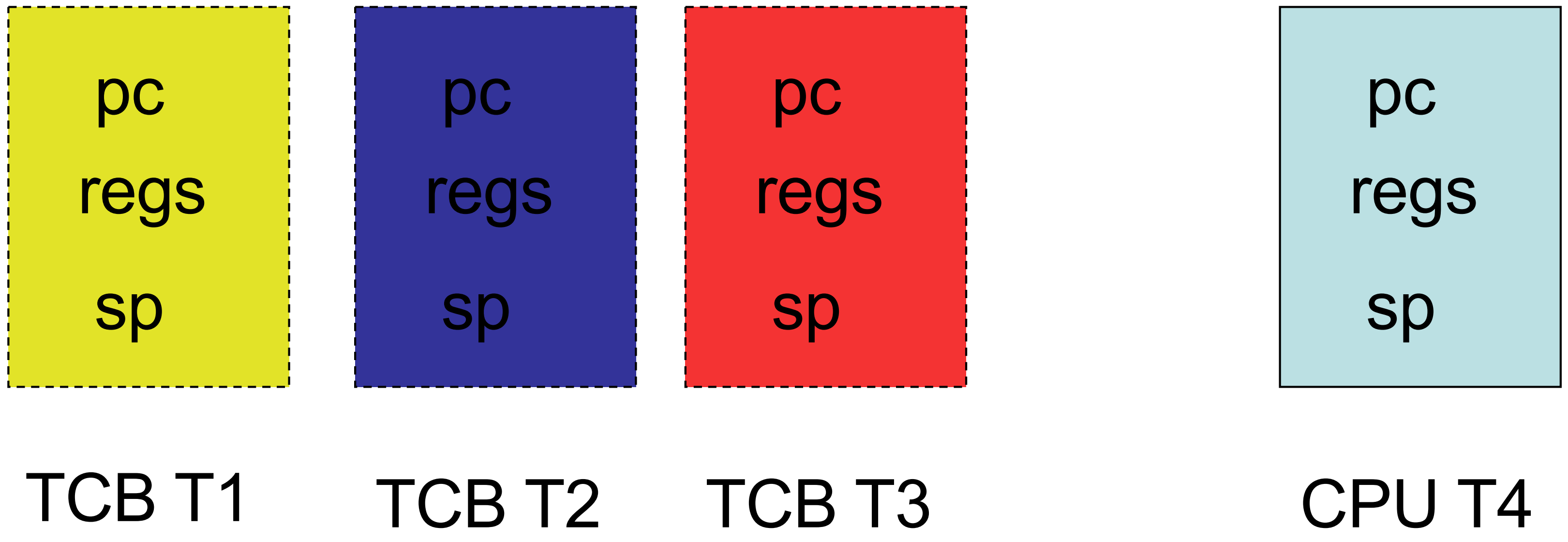
- Is it enough to depend on internal events?

Choosing the next thread to run

- If no ready thread, just loop idly
  - Loop switches to a thread when one is ready
- If 1 ready thread, run it
- If more than 1 ready, choose one
  - FIFO
  - Priority queue

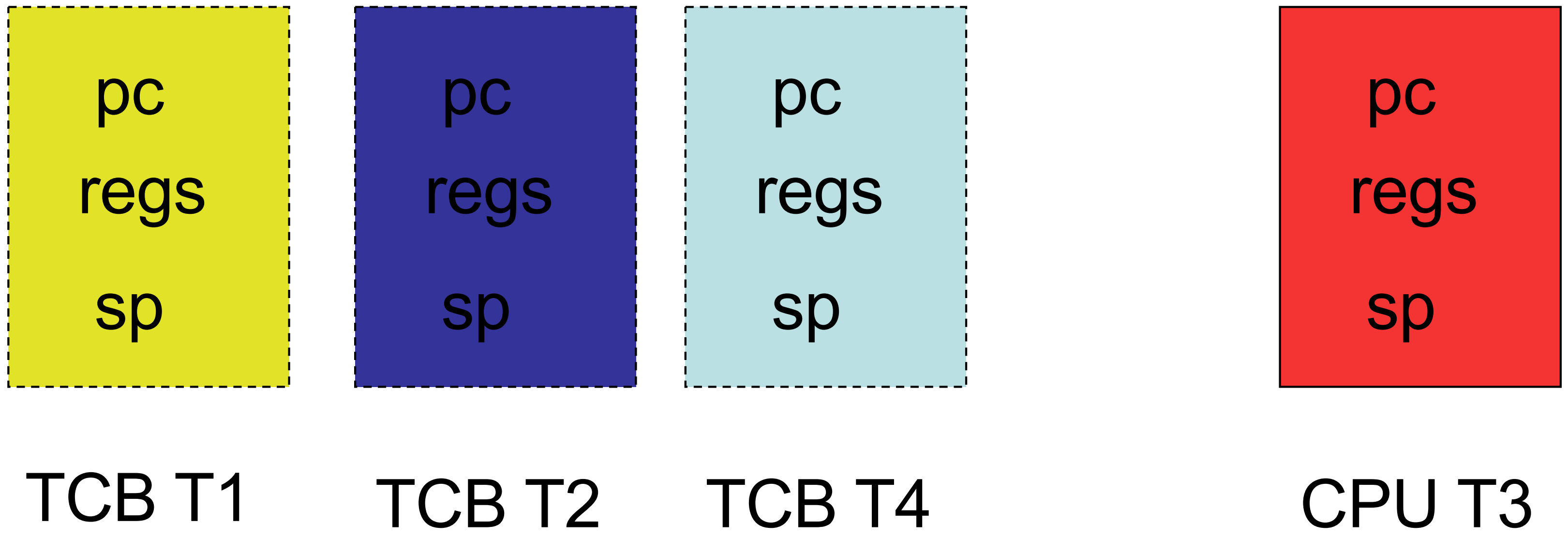
Context switching

- A thread is a sequence of instructions
- What do you do with a thread when it is not running?



Context switching

- A thread is a sequence of instructions
- What do you do with a thread when it is not running?



Saving state of current thread

- How to save state of the current thread?
  - Save registers, PC, stack pointer
  - Very tricky assembly-language code
  - Why won’t the following code work?

```
100 save PC (I.e. value 100)
101 switch to next thread
```

Loading context of new thread

- How to load the context of the next thread to run and run it?
  - Registers?
  - Stack?
  - Resume execution?
- Who is running these steps?
- How does the thread that just gave up control run again?

Example of thread switching

```
Thread 1
  print “start thread 1”
  yield()
  print “end thread 1”

Thread 2
  print “start thread 2”
  yield()
  print “end thread 2”

Yield
  print “start yield (thread %d)”
  switch to next thread
  print “end yield (current thread %d)
```

Thread switching in Linux

- PCB == TCB conceptually
- Thread switching is the same as Process switching except that the address space stays the same
- Details of switching function, any thread that switches **must** do so through this function

Thread switching in Linux

- When executing in kernel, executing on behalf of a thread
  - Kernel stack key to this abstraction on x86
    - Contains local state (stack) and process struct
    - E.g., current pointer
  - Other architectures use different techniques

```
switch_to(task_struct *prev_p,
          task_struct *next_p)
```

x86 assembly overview (32 bit)

- Movl src, dst
- Pushl reg
- Pushfl – push eflags
- Jump imm
- Popl reg
- Popfl – pop eflags

- Eax – gp reg
- Ebx – gp reg
- Ecx – gp reg
- Edx – gp reg
- …
- Ebp – frame pointer
- Eip -- PC
- Esp – stack pointer

Thread switching in xv6

```
# void swtch(struct context **old, struct context *new);
#
# Save current register context in old
# and then load register context from new
swtch:
# Save old registers
movl 4(%esp), %eax # put old ptr into eax
popl 0(%eax) # save the old IP
movl %esp, 4(%eax) # and stack
movl %ebx, 8(%eax) # and other registers
movl %ecx, 12(%eax)
movl %edx, 16(%eax)
movl %esi, 20(%eax)
movl %edi, 24(%eax)
movl %ebp, 28(%eax)
```

Thread switching in xv6

```
# Load new registers
movl 4(%esp), %eax # put new ptr into eax
movl 28(%eax), %ebp # restore other registers
movl 24(%eax), %edi
movl 20(%eax), %esi
movl 16(%eax), %edx
movl 12(%eax), %ecx
movl 8(%eax), %ebx
movl 4(%eax), %esp # stack is switched here
pushl 0(%eax) # return addr put in place
ret # finally return into new ctxt
```