COMP312005 Operating System

Project1 : Threads

Prof. Seokin Hong

Kyungpook National University

Spring 2019

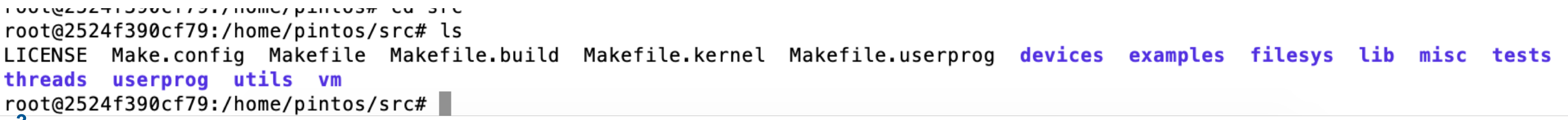
Slides from USC CS350 (http://bits.usc.edu/cs350/assignments/project1.pdf)

What is Pintos

- A 80x86 operating systems framework developed by Ben Pfaff for Stanford
- You will need to only code in C
- Makefiles are provided to you
- Each project consists of two parts
  - Programming
  - Design Document

How does pintos work

- Entry point of Pintos is threads/start.s
- start.s will initialize operating system resources, and call main() in threads/init.c
- main() will parse command line arguments, setup kernel memory, initialize the interrupt system, and call thread_start() in threads/thread.c for the main thread
- For the purposes of your project, the main thread are all the test files we will run



```
root@2524f390cf79:/home/pintos/src# ls
LICENSE  Make.config  Makefile  Makefile.build  Makefile.kernel  Makefile.userprog  devices  examples  filesys  lib  misc  tests
threads  userprog  utils  vm
root@2524f390cf79:/home/pintos/src#
```

Development Environment

- We will use Docker.
  - Setup tutorial is available on LMS

Working with Pintos

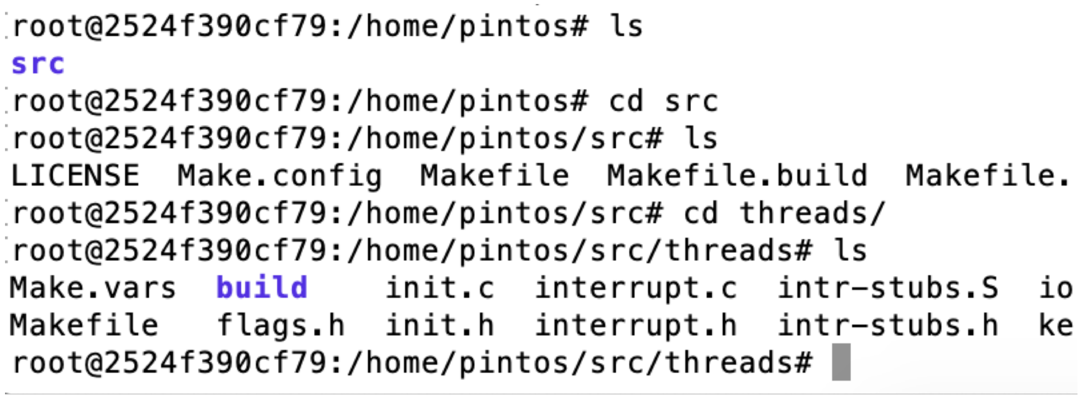
- Each of the three projects has its own main directory:
  - Project1: src/threads
  - Project2: src/userprog
  - Project3: src/vm

  - For each project, type $make in the project's main directory to compile your project
    - **$ cd src/threads**
    - **$ make**
  - This will create a new directory build/
  - Type **$make** check to run all tests **in the build directory**
    - You can check how many tests you are passing or failing

  - You can run **$make clean** to clean your project

Working with Pintos

- For each project, type $make in the project's main directory to compile your project
  - **$ cd src/threads**
  - **$ make**



```
root@2524f390cf79:/home/pintos# ls
src
root@2524f390cf79:/home/pintos# cd src
root@2524f390cf79:/home/pintos/src# ls
LICENSE  Make.config  Makefile  Makefile.build  Makefile.
root@2524f390cf79:/home/pintos/src# cd threads/
root@2524f390cf79:/home/pintos/src/threads# ls
Make.vars  build    init.c  interrupt.c  intr-stubs.S  io
Makefile   flags.h  init.h  interrupt.h  intr-stubs.h  ke
root@2524f390cf79:/home/pintos/src/threads# 
```

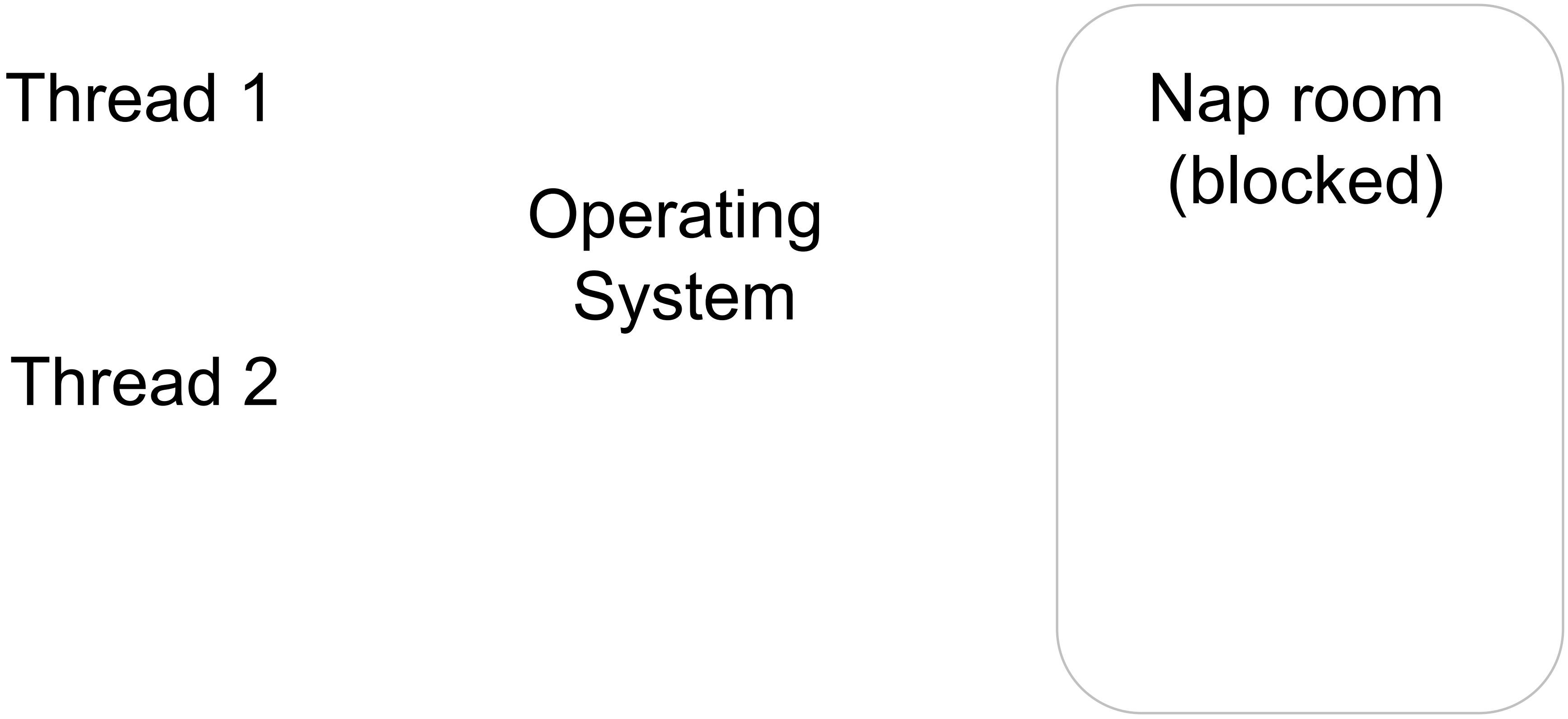
Working with Pintos

- Type **$make check** to run all tests **in the build directory**
  - You can check how many tests you are passing or failing
    - $cd build/
    - $make check

Working with Pintos

- For each project, type $make in the project’s main directory to compile your project
    - **$ cd src/threads**
    - **$ make**
  - This will create a new directory build/
  - Type **$make** check to run all tests **in the build directory**
    - You can check how many tests you are passing or failing
  - You can run **$make clean** to clean your project

Project1: Alarm clock



Project1: Alarm clock

Thread 1

Hey I want to take a nap for 10 seconds

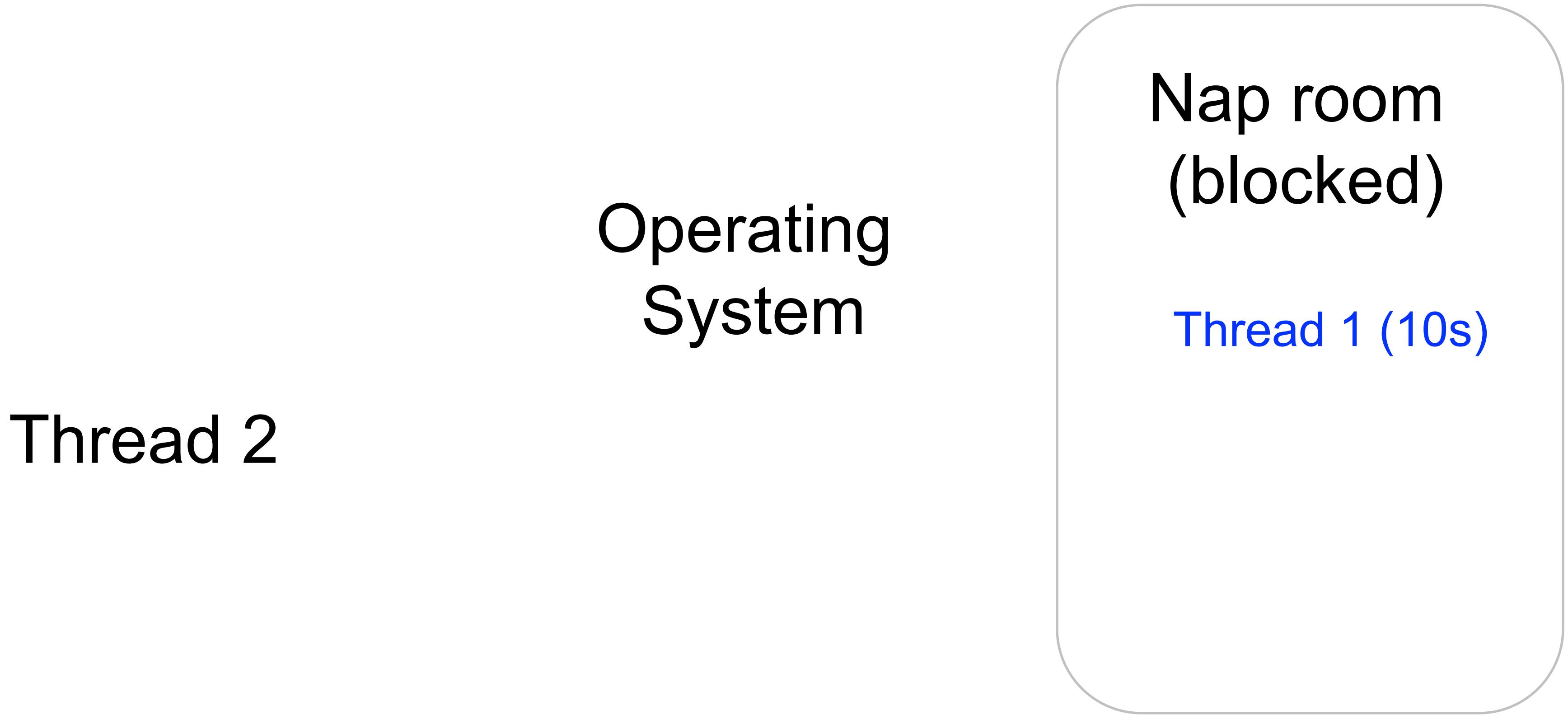
Sure!

Operating System

Nap room (blocked)

Thread 2

Project1: Alarm clock



Project1: Alarm clock

(5 seconds later)

Operating System

Nap room (blocked)

Thread 1 (10s, 5 seconds left)

Thread 2

Hey I want to take a nap for 5 seconds

Project1: Alarm clock

(5 seconds later)

Sure!

Operating System

Nap room (blocked)

Thread 1 (10s, 5 seconds left)

Thread 2

Hey I want to take a nap for 5 seconds

Project1: Alarm clock

(5 seconds later)

Thread 2

Operating System

Nap room (blocked)

Thread 1 (10s, 5 seconds left)

Thread 2 (5s)

Project1: Alarm clock

(10 seconds later)

Time to wake up thread 1 and 2

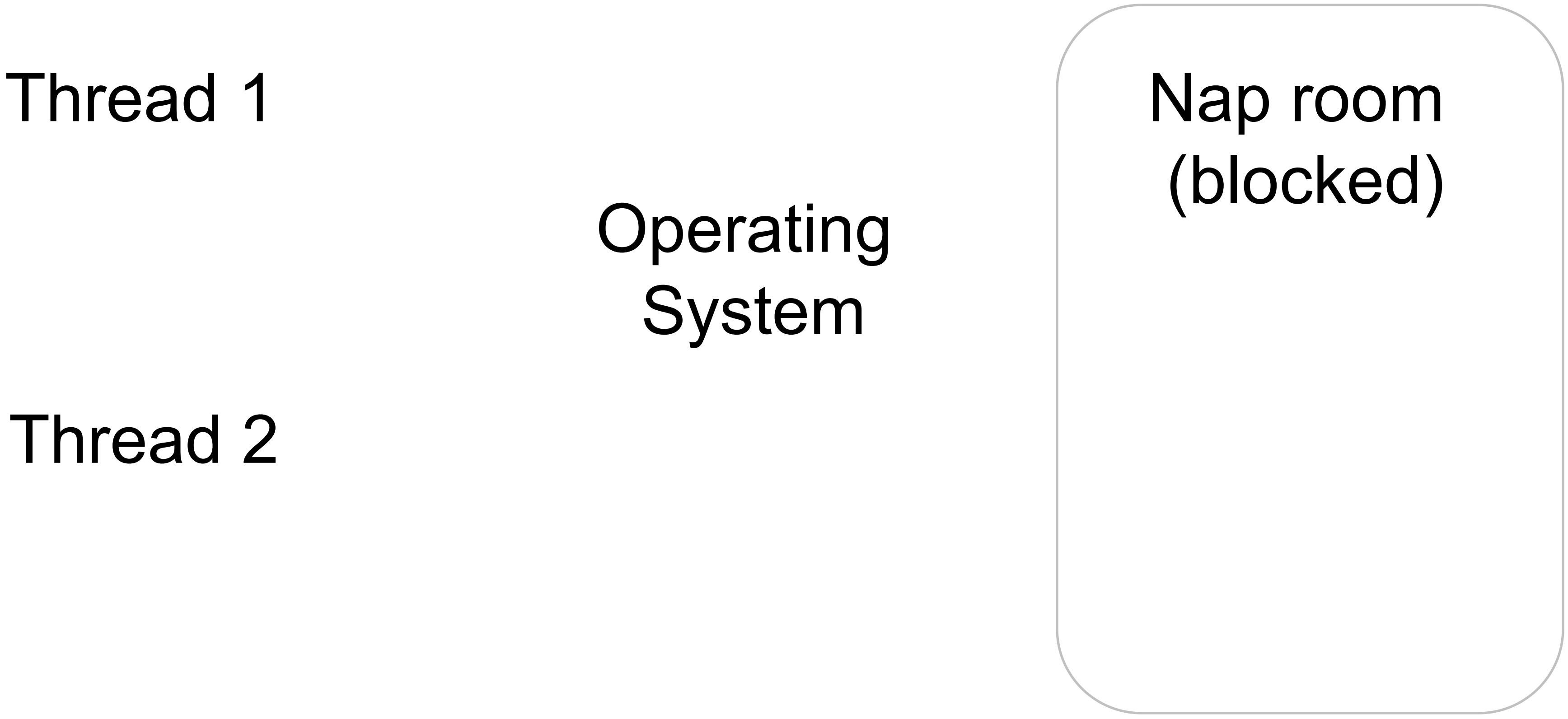
Operating System

Nap room (blocked)

Thread 1 (10s, 0 seconds left)

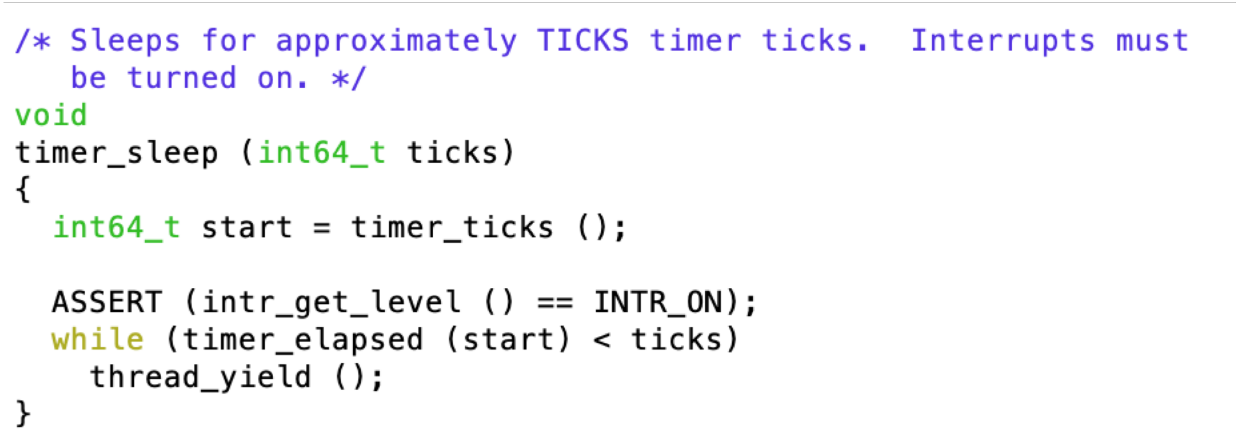
Thread 2 (5s, 0 seconds left)

Project1: Alarm clock



Project1: Alarm clock

- By default timer_sleep in pontos/src/devices/timer.c simply busy waits
- This is bad because you cannot guarantee the sleeping thread won’t get scheduled back to the processor → wasting resources for non-sleeping threads
- You will have to block the thread to **sleep** instead of **busy-waiting**!!



```
/* Sleeps for approximately TICKS timer ticks.  Interrupts must
   be turned on. */
void
timer_sleep (int64_t ticks)
{
  int64_t start = timer_ticks ();

  ASSERT (intr_get_level () == INTR_ON);
  while (timer_elapsed (start) < ticks)
    thread_yield ();
}
```

Project1: Alarm clock

- We will evaluate your code with the following steps
  - **$ cd src/threads**
  - **$ make**
  - $ cd build
  - $ make check

- Test results should be..
  - alarm-single PASSED
  - alarm-multiple PASSED
  - alarm-simultaneous PASSED
  - alarm-zero PASSED
  - alarm-negative PASSED

What you need to submit

- **Design document**
  - You need to copy the design document in “src/threads” directory of the PintOS source code when you submit it.
- **PintOS source code.**
  - Submit a compressed file to LMS!!

Documentation template

---- GROUP ----

>> Fill in the names and email addresses of your group members.

FirstName LastName <email@domain.example>

FirstName LastName <email@domain.example>

FirstName LastName <email@domain.example>

---- PRELIMINARIES ----

>> Describe briefly which parts of the assignment were implemented by each member of your team and specify the contribution between your member, say 3:3:4, or 1:3:6.

FirstName LastName: contribution

FirstName LastName: contribution

FirstName LastName: contribution

ALARM CLOCK

===========

---- DATA STRUCTURES ----

>> A1: Copy here the declaration of each new or changed `struct’ or `struct' member, global or static variable, `typedef', or enumeration. Identify the purpose of each in 25 words or less.

---- ALGORITHMS ----

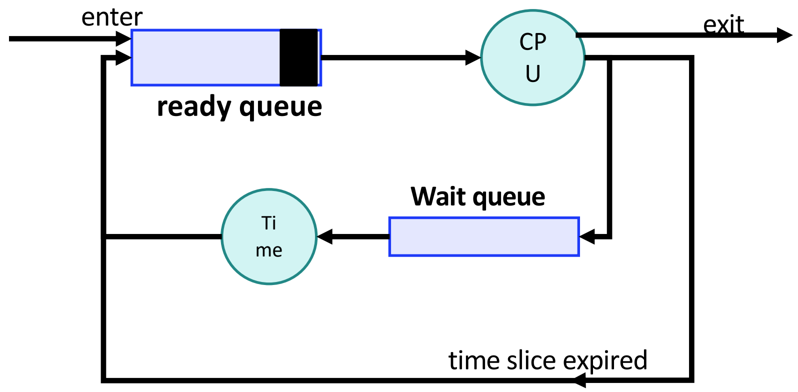
>> A2: Briefly describe what happens in a call to timer_sleep(), including the effects of the timer interrupt handler.

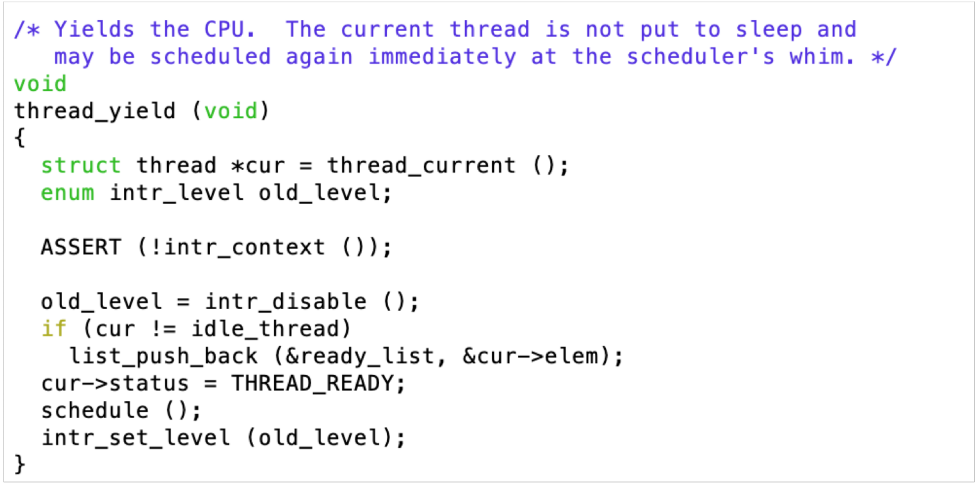
>> A3: What steps are taken to minimize the amount of time spent in the timer interrupt handler?

Appendix: Some hints

```
/* pintos/src/device/timer.c */
void timer_sleep(int64_t ticks){
      int64_t start = timer_ticks();
      while(timer_elapsed(start) < ticks)
                  thread_yield();
}
```

- **Problem**
  - **thread_yield()**
    - Current thread will be placed at the back of the queue placed at the back of the queue
    - Scheduler will keep dispatch the current thread!! → busy waiting!!



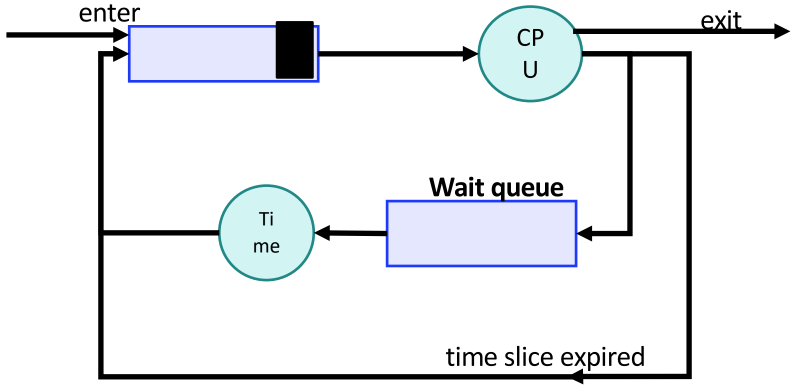


```
/* Yields the CPU.  The current thread is not put to sleep and
   may be scheduled again immediately at the scheduler's whim. */
void
thread_yield (void)
{
  struct thread *cur = thread_current ();
  enum intr_level old_level;

  ASSERT (!intr_context ());

  old_level = intr_disable ();
  if (cur != idle_thread)
    list_push_back (&ready_list, &cur->elem);
  cur->status = THREAD_READY;
  schedule ();
  intr_set_level (old_level);
}
```

Appendix: Some hints

- **One solution**
  - Block the current thread with **thread_block()**
  - Put the current thread to a **waiting list (queue)** and wake it up after the sleep time (ticks)
  - store a target wake time (ticks) in the thread structure
  - **On timer interrupt,**
    - check if there is a thread that should wake up (→ this should be very fast)

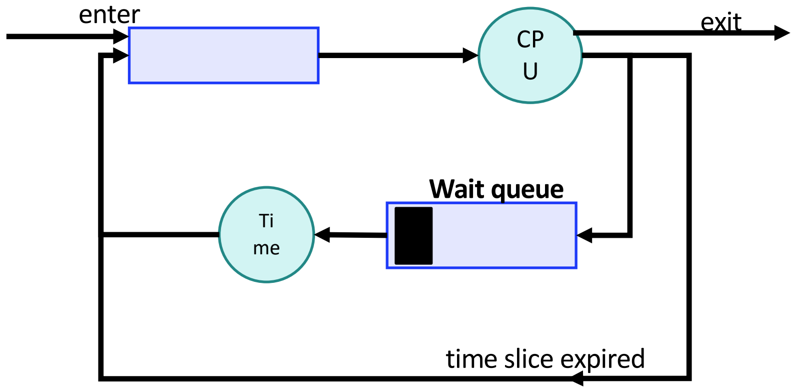
schedule()



Appendix: Some hints

- **One solution**
  - Block the current thread with **thread_block()**
  - Put the current thread to a **waiting list (queue)** and wake it up after the sleep time (ticks)
  - store a target wake time (ticks) in the thread structure
  - **On timer interrupt,**
    - check if there is a thread that should wake up (→ this should be very fast)

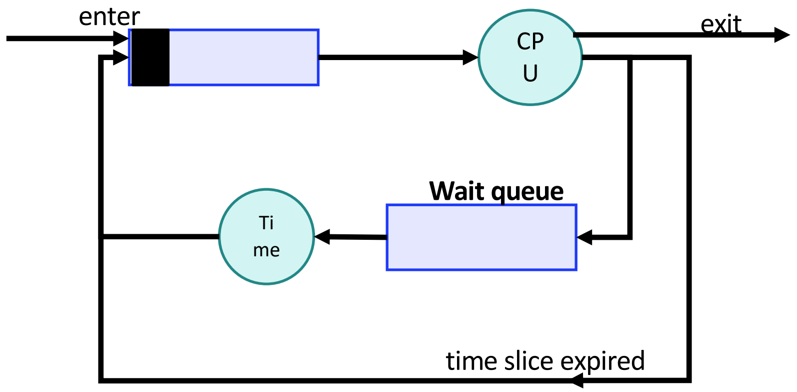
thread_block()



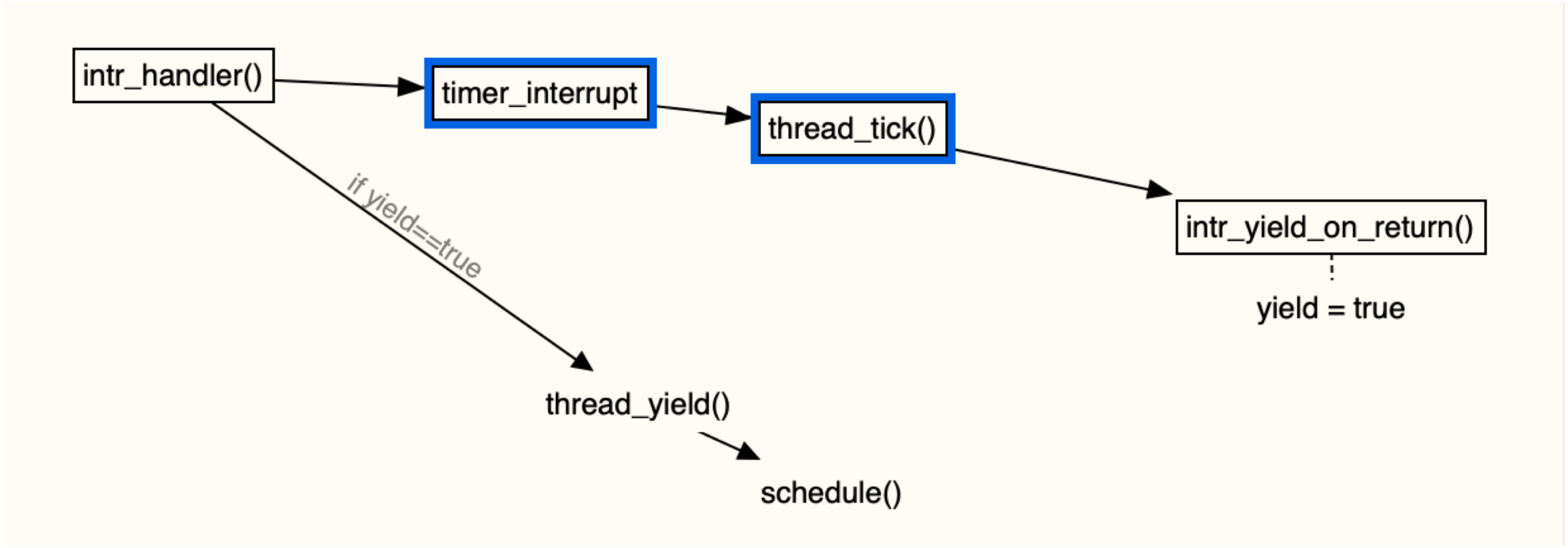
Appendix: Some hints

- **One solution**
  - Block the current thread with **thread_block()**
  - Put the current thread to a **waiting list (queue)** and wake it up after the sleep time (ticks)
  - store a target wake time (ticks) in the thread structure
  - **On timer interrupt,**
    - check if there is a thread that should wake up (→ this should be very fast)

???

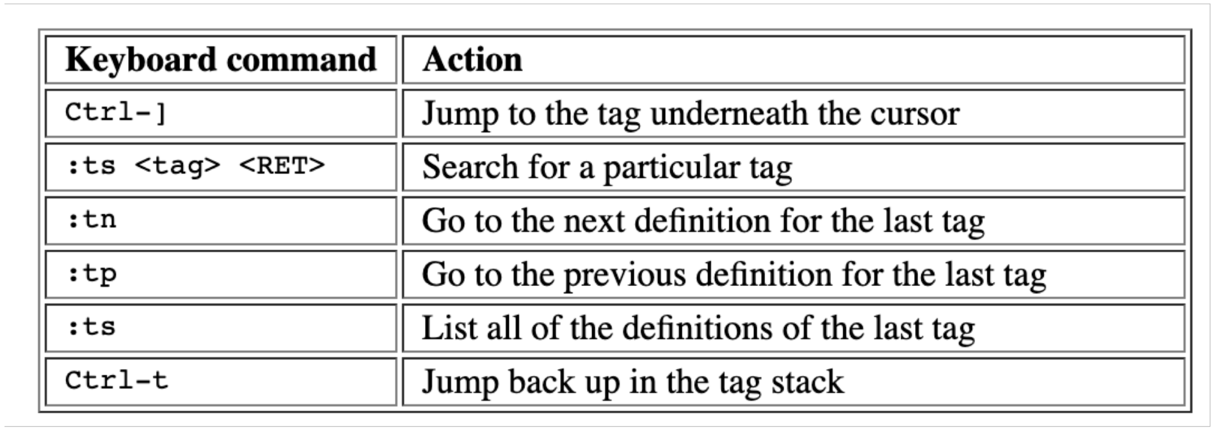


Appendix: functions associated with timer interrupt



Ctags ← Tool for hacking open source project

- Install
  - $ apt-get install ctags
- Generate “tags” file
  - $ ctags -R *
- Run VIM with “tags” file
  - $ vim -t tags



Keyboard command	Action
`Ctrl-]`	Jump to the tag underneath the cursor
`:ts <tag> <RET>`	Search for a particular tag
`:tn`	Go to the next definition for the last tag
`:tp`	Go to the previous definition for the last tag
`:ts`	List all of the definitions of the last tag
`Ctrl-t`	Jump back up in the tag stack