Xidian University

《操作系统课程设计》

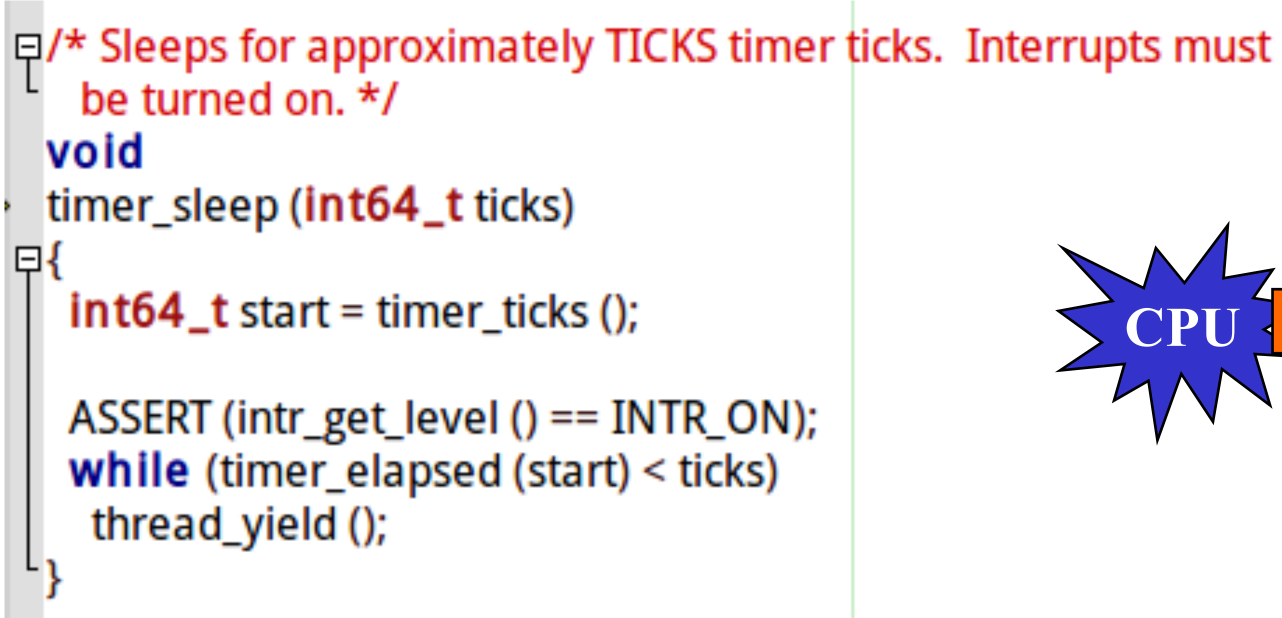
实验1：Alarm-Clock

黄伯虎

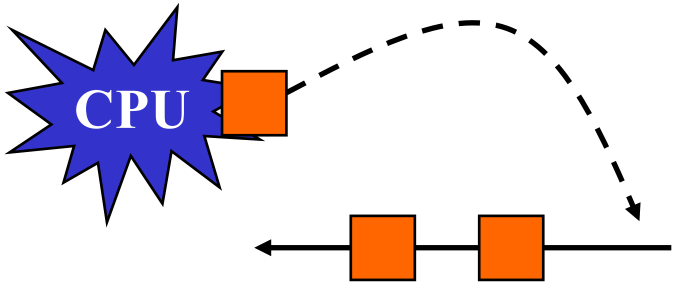
School of Computer Science & Technology

任务描述

- 源代码devices/timer.c中有一个timer_sleep()函数。定义如下：



```
/* 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 ();
}
```



•该函数的功能是让调用它的线程睡眠一段时间(ticks)，然后唤醒。

•事实上，Pintos已经实现该函数，只是使用的是“忙等待”的方法(见while循环)。

任务描述

本实验任务：

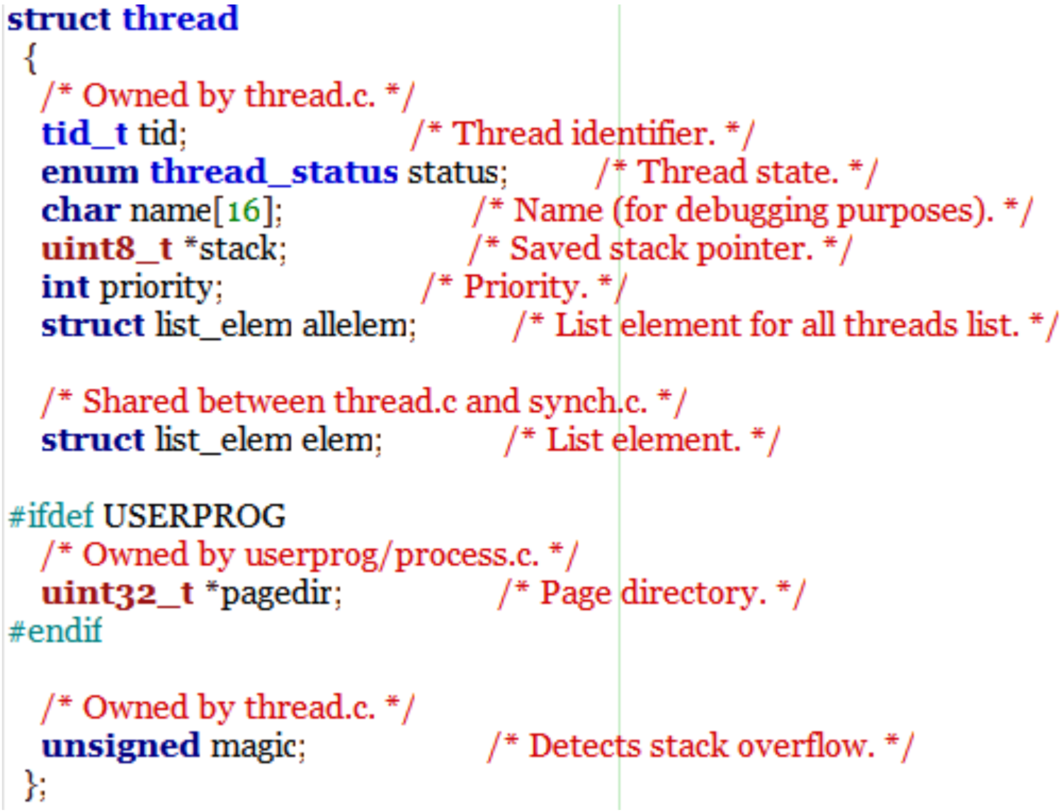
- 重新实现timer_sleep( )函数，避免“忙等待”的发生
- 策略有多种，请大家设计一种并实现即可

相关源代码

- **为顺利完成本实验，你至少需要阅读以下源代码文件(并非每一行都要读懂)，并了解其中关键数据结构和函数的含义，它们是：**
  - **../src/threads/**目录下：
    - **thread.h, thread.c**：有关线程初始化、阻塞、解除阻塞，线程调度等内容；
    - **interrupt.h, interrupt.c**：与中断有关的处理函数。
  - **../src/devices/**目录下：
    - **timer.h, timer.c**：本实验要修改的**time_sleep( )**函数就在其中，同时请注意理解定时器中断的处理过程。

数据结构

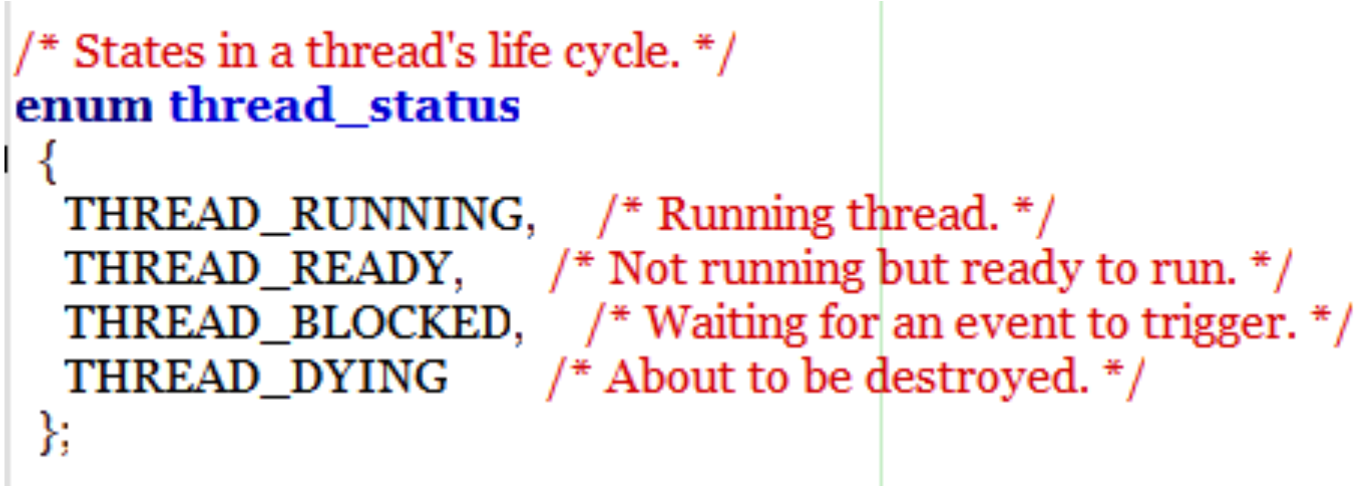
- **thread.h中定义了一个结构体struct thread，这个结构体用于存放线程的基本信息**



```
struct thread
 {
  /* Owned by thread.c. */
  tid_t tid;                    /* Thread identifier. */
  enum thread_status status;          /* Thread state. */
  char name[16];                   /* Name (for debugging purposes). */
  uint8_t *stack;                  /* Saved stack pointer. */
  int priority;                 /* Priority. */
  struct list_elem allelem;          /* List element for all threads list. */

  /* Shared between thread.c and synch.c. */
  struct list_elem elem;            /* List element. */

#ifdef USERPROG
  /* Owned by userprog/process.c. */
  uint32_t *pagedir;                /* Page directory. */
#endif

  /* Owned by thread.c. */
  unsigned magic;                  /* Detects stack overflow. */
 };
```

数据结构

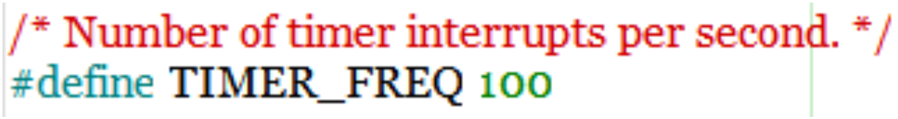
- **Pintos中线程的状态有四种，threads.h中定义如下：**



```
/* States in a thread's life cycle. */
enum thread_status
 {
   THREAD_RUNNING,    /* Running thread. */
   THREAD_READY,      /* Not running but ready to run. */
   THREAD_BLOCKED,    /* Waiting for an event to trigger. */
   THREAD_DYING       /* About to be destroyed. */
 };
```

系统的驱动

- **驱动力：定时器中断（timer interrupt）**
- **定时器中断频率(time.h)：**



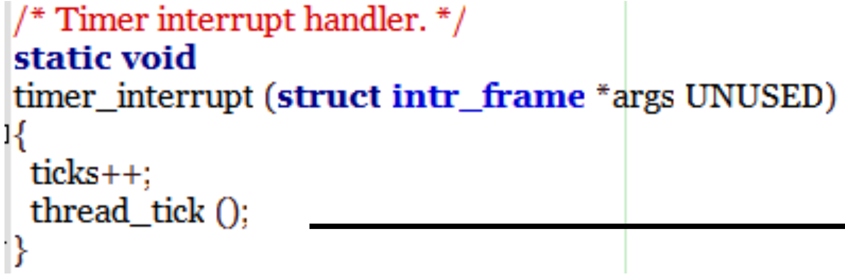
```
/* Number of timer interrupts per second. */
#define TIMER_FREQ 100
```

由此可知一个定时器中断的时长大约为10ms，这里称为一个ticks。

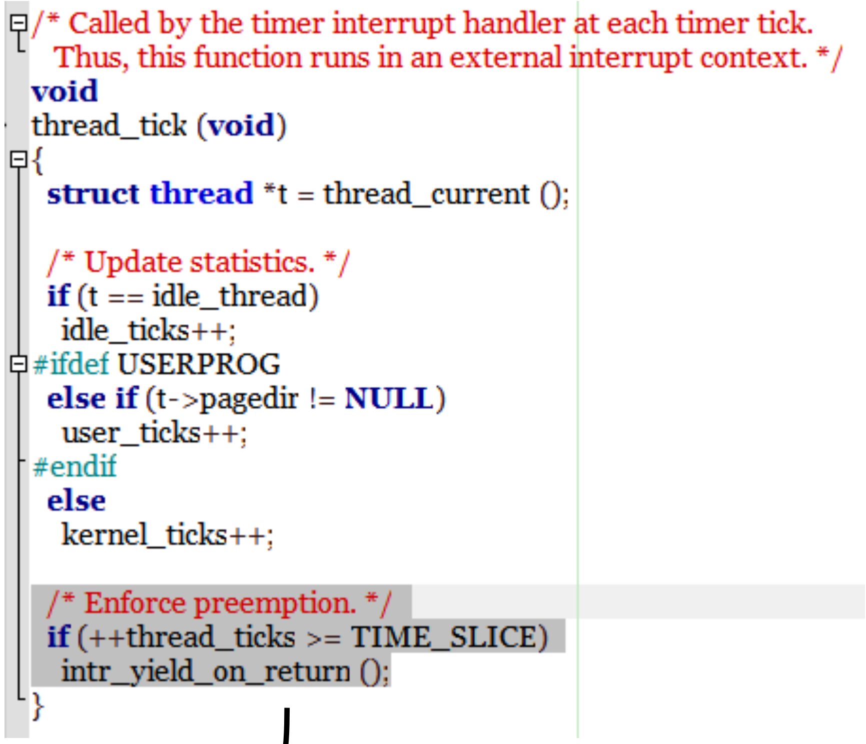
Pintos中一个时间片的长度 = 4*ticks ≈ 40ms。当一个线程运行了一个时间片后，它必须放弃处理器给其它的线程。那么系统是如何知道当前线程的运行时间，以及何时进行线程切换呢？

中断处理过程

Timer interrupt → 系统接受到后调用 intr_hannder() → 调用timer_interrupt()

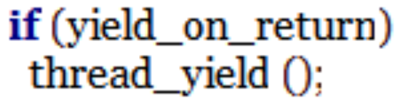


```
/* Timer interrupt handler. */
static void
timer_interrupt (struct intr_frame *args UNUSED)
{
  ticks++;
  thread_tick ();
}
```

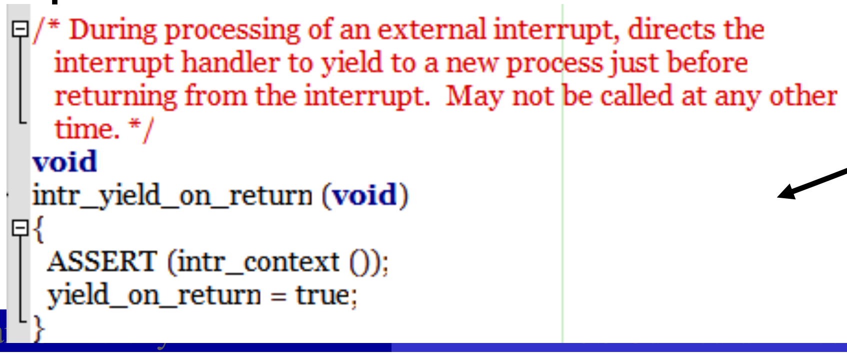


```
/* Called by the timer interrupt handler at each timer tick.
   Thus, this function runs in an external interrupt context. */
void
thread_tick (void)
{
  struct thread *t = thread_current ();

  /* Update statistics. */
  if (t == idle_thread)
    idle_ticks++;
#ifdef USERPROG
  else if (t->pagedir != NULL)
    user_ticks++;
#endif
  else
    kernel_ticks++;

  /* Enforce preemption. */
  if (++thread_ticks >= TIME_SLICE)
    intr_yield_on_return ();
}
```

intr_handler( )中

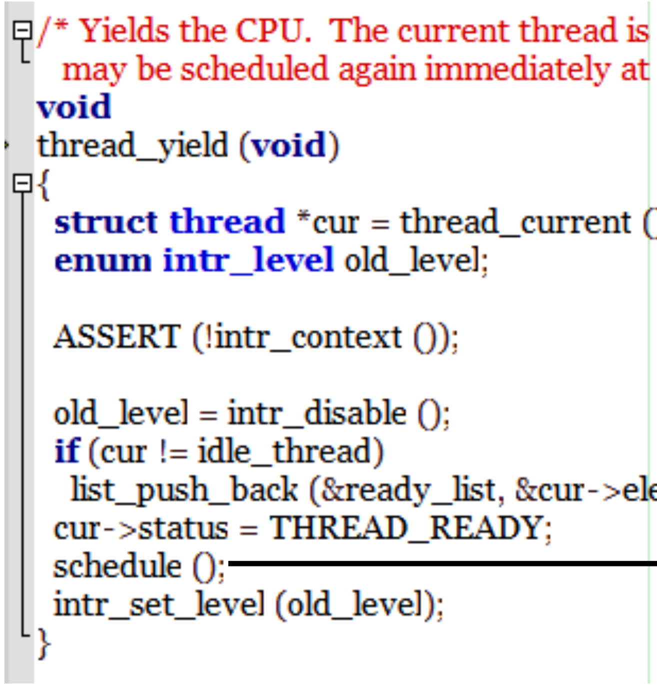


```
if (yield_on_return)
  thread_yield ();
```

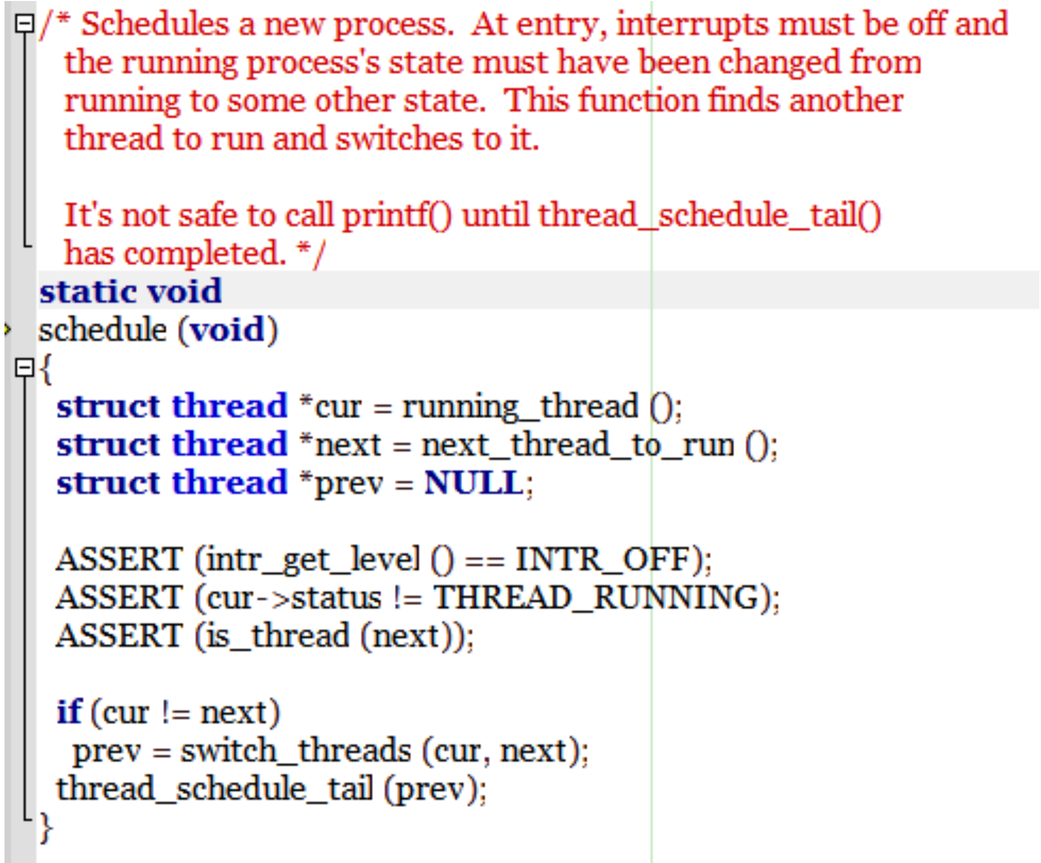


```
/* During processing of an external interrupt, directs the
   interrupt handler to yield to a new process just before
   returning from the interrupt.  May not be called at any other
   time. */
void
intr_yield_on_return (void)
{
  ASSERT (intr_context ());
  yield_on_return = true;
}
```

中断处理过程



```
/* 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);
}
```



```
/* Schedules a new process.  At entry, interrupts must be off and
   the running process's state must have been changed from
   running to some other state.  This function finds another
   thread to run and switches to it.

   It's not safe to call printf() until thread_schedule_tail()
   has completed. */
static void
schedule (void)
{
  struct thread *cur = running_thread ();
  struct thread *next = next_thread_to_run ();
  struct thread *prev = NULL;

  ASSERT (intr_get_level () == INTR_OFF);
  ASSERT (cur->status != THREAD_RUNNING);
  ASSERT (is_thread (next));

  if (cur != next)
    prev = switch_threads (cur, next);
  thread_schedule_tail (prev);
}
```

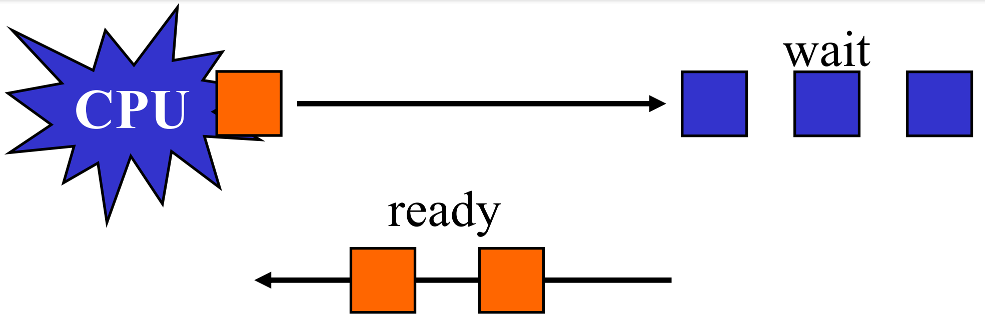
其它需关注函数

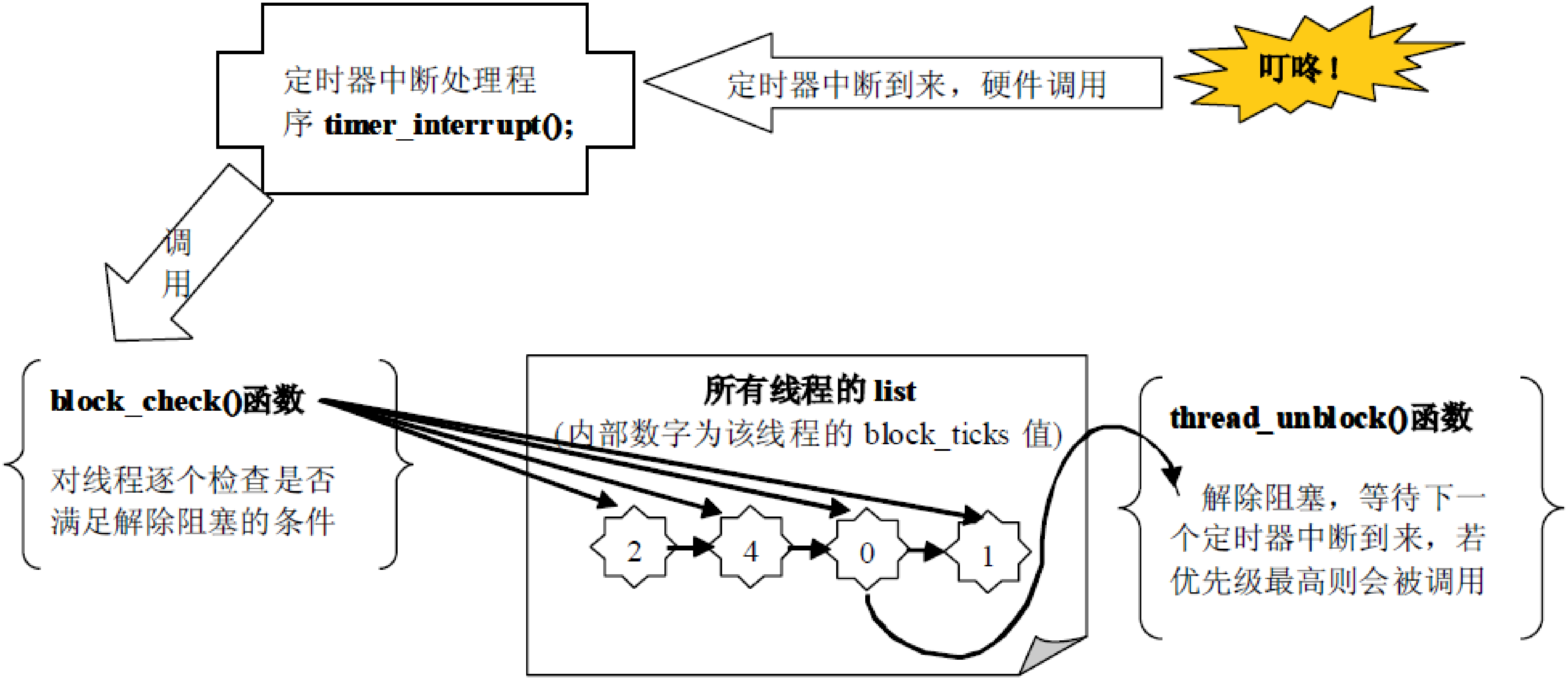
- **thread_current()**
  - 获取当前的线程的指针。
- **thread_foreach(thread_action_func *func, void *aux)**
  - 遍历当前**ready queue**中的所有线程，并且对于每一个线程执行一次**func**操作（注意到这里的**func**是一个任意给定函数的指针，参数**aux**则是你想要传给这个函数的参数）。实际上**Pintos**中所有**ready**的线程被保存在一个链表**(ready_list)**中，这个函数做的不过是遍历了一遍链表而已。注意这个函数只能在中断关闭的时候调用。

其它需关注函数

- **thread_block()和thread_unblock(thread *t)**
  - 这是一对函数，区别在于第一个函数的作用是把当前占用CPU的线程阻塞掉（放到waiting queue里面）；第二个函数作用是将已经被阻塞掉的进程t唤醒到ready队列中。
- **timer_ticks( )**
  - 返回自从系统启动经过的时间量(以ticks为单位)。
- **timer_elapsed( )**
  - 返回自某个时刻起经过的时间量（以ticks为单位）。

一种策略




叮咚！

定时器中断到来，硬件调用

定时器中断处理程序 **timer_interrupt();**

调用

block_check()函数

对线程逐个检查是否满足解除阻塞的条件

所有线程的 list

(内部数字为该线程的 block_ticks 值)

thread_unblock()函数

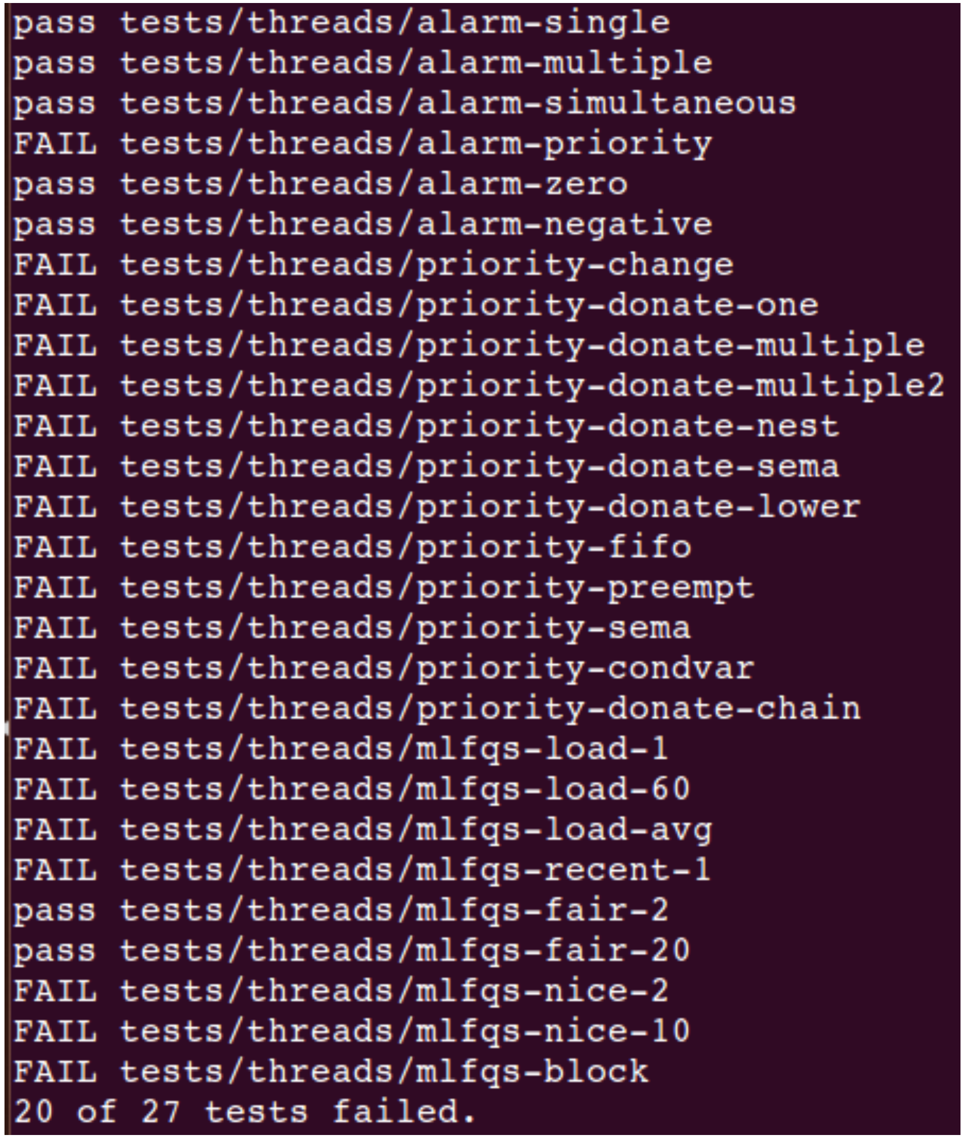
解除阻塞，等待下一个定时器中断到来，若优先级最高则会被调用

要求

- 基本要求：实现本讲义中策略
- 同时我们鼓励提出新的策略并尝试实现！

实验结果测试

- 进入../printos/src/threads/目录，运行#make check命令，它会自动检查你的任务有没有完成。



```
pass tests/threads/alarm-single
pass tests/threads/alarm-multiple
pass tests/threads/alarm-simultaneous
FAIL tests/threads/alarm-priority
pass tests/threads/alarm-zero
pass tests/threads/alarm-negative
FAIL tests/threads/priority-change
FAIL tests/threads/priority-donate-one
FAIL tests/threads/priority-donate-multiple
FAIL tests/threads/priority-donate-multiple2
FAIL tests/threads/priority-donate-nest
FAIL tests/threads/priority-donate-sema
FAIL tests/threads/priority-donate-lower
FAIL tests/threads/priority-fifo
FAIL tests/threads/priority-preempt
FAIL tests/threads/priority-sema
FAIL tests/threads/priority-condvar
FAIL tests/threads/priority-donate-chain
FAIL tests/threads/mlfqs-load-1
FAIL tests/threads/mlfqs-load-60
FAIL tests/threads/mlfqs-load-avg
FAIL tests/threads/mlfqs-recent-1
pass tests/threads/mlfqs-fair-2
pass tests/threads/mlfqs-fair-20
FAIL tests/threads/mlfqs-nice-2
FAIL tests/threads/mlfqs-nice-10
FAIL tests/threads/mlfqs-block
20 of 27 tests failed.
```