**西安电子科技大学**

**操作系统课程设计**

**(2021年度)**

**实**

**验**

**报**

**告**

**实验名称：** Alarm-Clock

**班 级：** 1903015

**姓 名：** 贾舜宇

**学 号：** 19030100383

**一、实验内容**

Pintos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 ();  
}

该函数功能是让调用它的线程睡眠一段时间（ticks），然后唤醒。

Pintos中原始实现方式使用的是“忙等待”，现在要求重新实现timer\_sleep函数，避免“忙等待”发生。

**二、分析与设计**

timer\_sleep()函数中大概分为三个部分int64\_t start = timer\_ticks () ;ASSERT（）函数；与while循环。

其中timer-ticks()函数是用来获取ticks的当前值返回；intr\_get\_level()返回了intr\_level的值；intr\_disable()获取了当前的中断状态， 然后将当前中断状态改为不能被中断， 然后返回执行之前的中断状态。

/\* Returns the number of timer ticks since the OS booted. \*/  
int64\_t  
timer\_ticks (void)   
{  
 enum intr\_level old\_level = intr\_disable ();  
 int64\_t t = ticks;  
 intr\_set\_level (old\_level);  
 return t;  
}

ASSERT（）中用来判断timer-ticks()中的中断状态是否改变成功。

while()循环则用来判断什么时候停止执行 thread\_yield ()，而thread\_yield ()函数是将当前线程放到就绪队列中，令其“睡眠”。

而Pintos中这个原始的timer\_sleep最大的缺点就是，在它工作的时候线程不断地在就绪队列和running队列中来回切换，占用了CPU资源。

按照课件中的思路，考虑在timer\_sleep()函数中让进程暂时阻塞(thread\_block())，然后等待ticks个定时器终端周期结束后再去唤醒它（thread\_unblock()）。

实现思路：调用timer\_sleep（）的时候令线程阻塞，然后给线程结构体加一个成员ticks\_blocked来记录这个线程“睡眠”了多少时间， 然后利用操作系统自身的时钟中断（每个tick会执行一次）加入对线程状态的检测， 每次检测将ticks\_blocked减1, 如果减到0就唤醒这个线程。

**三、详细实现**

**devices/time.c**中

主体函数改变，timer\_sleep()

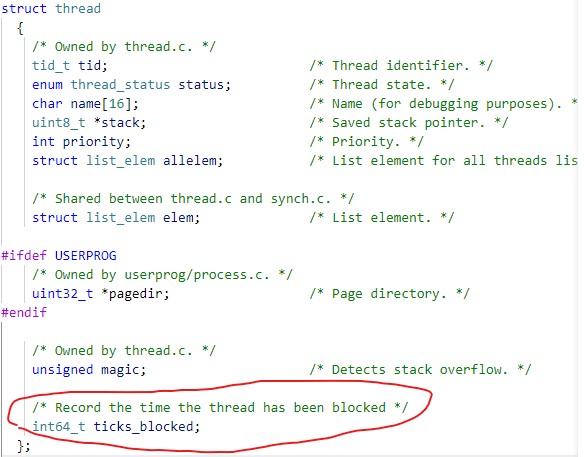
/\* Sleeps for approximately TICKS timer ticks. Interrupts must  
  be turned on. \*/  
void  
timer\_sleep (int64\_t ticks)   
{  
 if(ticks <= 0)  
{  
   return ;  
}  
 ASSERT (intr\_get\_level () == INTR\_ON); //判断当前线程的中断状态  
 enum intr\_level old\_level = intr\_disable();  
 struct thread \*current\_thread = thread\_current(); //为线程创建blocked成员  
 current\_thread->ticks\_blocked = ticks;  
 thread\_block();  
 intr\_set\_level(old\_level);  
}

在timer\_interrupt()中加入线程“睡眠”时间的检测

/\* Timer interrupt handler. \*/  
static void  
timer\_interrupt (struct intr\_frame \*args UNUSED)  
{  
 ticks++;  
 thread\_tick ();  
 thread\_foreach(blocked\_thread\_check, NULL);  
}

**threads/thread.h**中

给线程的结构体加入ticks\_blocked对象。



添加新的声明

void blocked\_thread\_check(struct thread \*t, void \*aux UNUSED);

**threads/thread.c**中

thread\_create()函数内初始化ticks\_blocked为0

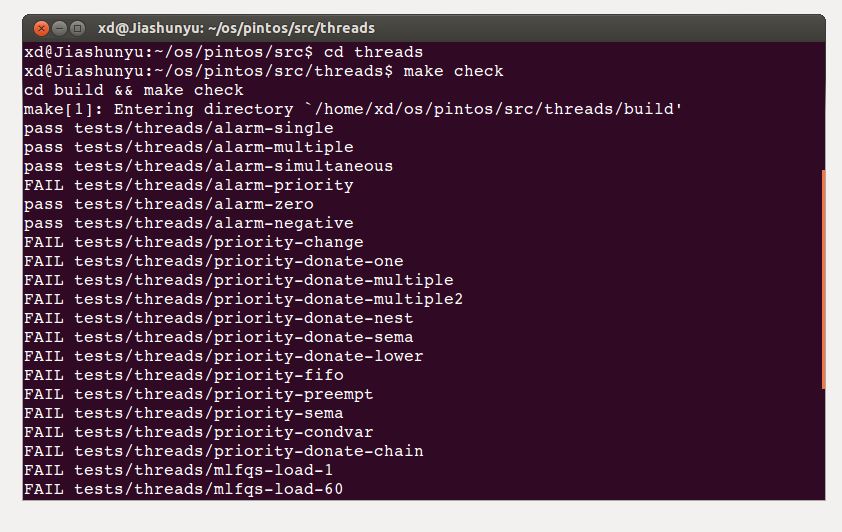
/\* Init ticks\_blocked \*/  
 t->ticks\_blocked = 0;

添加一个方法 blocked\_thread\_check ()

/\* Check the blocked thread \*/  
void  
blocked\_thread\_check(struct thread \*t, void \*aux UNUSED)  
{  
 if(t->status==THREAD\_BLOCKED && t->ticks\_blocked>0 )  
{  
   t->ticks\_blocked--;  
   if(t->ticks\_blocked == 0)  
  {  
     thread\_unblock(t);  
  }  
}  
}

**四、实验结果**

**在 os/pintos/src/threads目录下执行make check命令运行检测程序，结果如下：**

****

alarm序列中除去优先权测试为FAIL，其余皆通过

**五、心得体会**

本次Pintos相关实验是非常具有挑战性的，对我个人来说进行本次实验的第一个收获就是进一步学习到了阅读源码的技巧方法，首先要明白了解代码的执行目的和需要阅读的目标。比如在src/devices/timer.c中我要了解timer\_ticks（）的用法，先要明白他在timer.c中是否与其上下部分有联系，它的目的是要获取一个在while语句中用于判断的参数start，明白了它的目的之后再进行溯源的阅读会更快更有效。

第二点收获就是加深了对操作系统中进程与线程调度的理解，对代码层面操作系统功能执行的流程有了一个初步的认识。

第三点是在进行实验的准备阶段中，尝试在ubuntu 20.04系统中安装pintos，遇到了不少的问题，但是在查阅资料的过程中也学习到了一些关于Linux系统的知识。