**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：** AlarmClock

**班 级：** 1903015

**姓 名：** 吕思勤

**学 号：** 19030130448

**一、实验内容**

源代码devices/timer.c中有一个timer\_sleep()函数。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。

本实验的要求：重新实现timer\_sleep( )函数，避免“忙等待”的发生。

**二、分析与设计**

1.分析timer\_sleep( )函数:

timer\_sleep(int64\_t ticks)

{

int64\_t start = timer\_ticks();

ASSERT (intr\_get\_level() == INTR\_ON);

while (timer\_elapsed(start) < ticks)

thread\_yield();

}

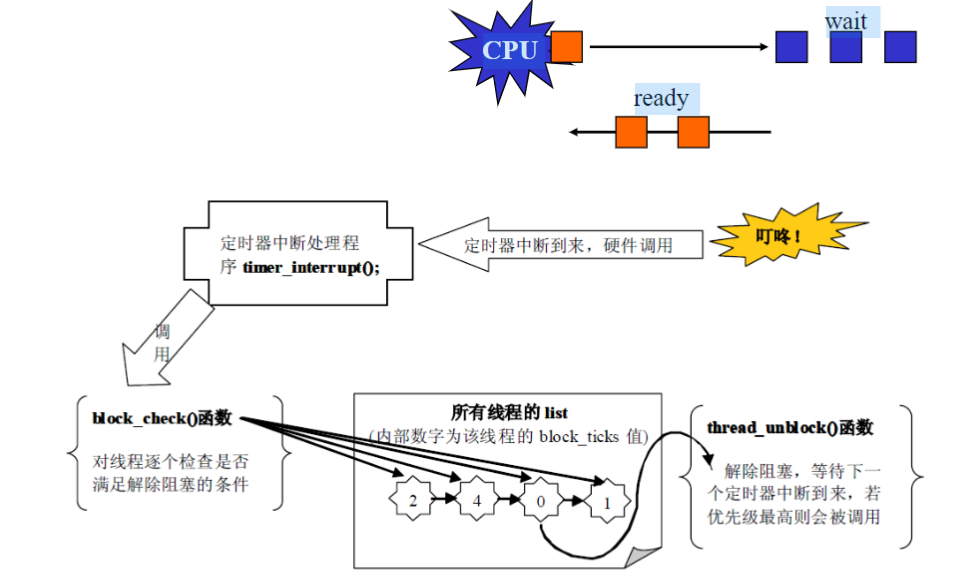
可以看出造成忙等待原因在于：

（1）它是通过一个while循环来不断检测当前经过的时间是否已经达到了预期时间来达到让线程等待ticks时间的目的。

（2）timer\_elapse( )函数只在一个定时器中断到达时才加1。由此我们知道在一个定时器中断时间内，while循环将会执行多次，从而形成“忙等待”。要避免“忙等待”，需要把timer\_sleep()函数中的while循环替换掉。

2.设计

考虑在timer\_sleep( )函数中让进程暂时阻塞(thread\_block())，然后等待ticks个定时器中断周期结束以后再去唤醒它(thread\_unblock())。这个“等”不再使用循环来实现，而是在每个定时器中断到来后去检查一下wait\_list中的线程，看是否已经到了唤醒的时机。如果时机已到，马上将其唤醒(进入ready\_list)；如反之则不做任何操作。没有了循环调用，其他线程就可以充分在CPU上运行。为了在每个定时器中断到来时去检查wait\_list，通过改进timer\_interrup()函数实现：加入遍历链表的操作，检查每个线程等待的时间。要知道一个线程被阻塞了多长时间，可加入一个整形变量来记录当前线程需要等待的时间，之后每次中断的时候检查一次，并自减1，当它等于0时，意味着等待结束，将其调入ready\_list。基本原理图如下：

****

**三、详细实现**

1.阻塞线程:在timer\_sleep函数中调用阻塞线程函数thread\_block()

timer\_sleep(int64\_t ticks)

{

if (ticks > 0)

{

enumintr\_levelold\_level;

old\_level= intr\_disable();

thread\_block(); //block thread

intr\_set\_level(old\_level);

}

}

2.计时

(1)改造thread结构体定义线程组塞时间 int block\_ticks; 并在timer\_sleep()中记录线程组塞时间timer\_sleep函数修改如下:

timer\_sleep (int64\_t ticks)

{

if (ticks > 0)

{

enum intr\_level old\_level;

struct thread \*t;

t=thread\_current ();

t->block\_ticks = ticks;//记录阻塞时间

old\_level = intr\_disable();

thread\_block(); //线程阻塞

intr\_set\_level(old\_level);

}

}

(2)定义block\_check()并在timer\_interrupt函数中调用，检查线程是否被阻塞

主要代码如下:

void block\_check(struct thread \*t, void \*aux UNUSED)

{

if (t->status == THREAD\_BLOCKED && t->block\_ticks>0)

{

t->block\_ticks--;

if (t->block\_ticks == 0)

{

thread\_unblock (t);

}

}

}

timer\_interrupt (struct intr\_frame \*args UNUSED)

{

ticks++;

enum intr\_level old\_level; //记录原来的中断状态

old\_level=intr\_disable(); //thread\_foreach函数要求关中断

thread\_foreach (block\_check,NULL);//检查是否阻塞

intr\_set\_level (old\_level); // 恢复中断

thread\_tick ();

}

**四、实验结果**

完成代码改写后，进入../printos/src/threads/目录，运行#make check命令，实验结果如下图：有5个相关检测通过(Alarm-single；Alarm-multiple；Alarm-simultaneous；Alarm-zero；Alarm-negative)

一張含有 文字 的圖片

自動產生的描述

**五、心得体会**

通过本次实验，回顾了上个学期在操作系统理论课程中所学习到的线程阻塞、中断与忙等待的基本概念，这次实验通过alarm clock实际例子，解决”忙等待”的问题，让我接触到操作系统的实际应用，虽然一开始对这看似困难且庞大的程序无从下手，但在老师清晰的讲解下，我逐渐理解程序中一些关键函数的调用方式与工作原理，从中分析并加以修改，过程中也因为在线程结构体上没有修改好报错了几次，但在不断地尝试与分析下，最终成功通过调试，完成了本次实验，在实验的过程中阅读代码让我对于操作系统的概念掌握的更加彻底，做完本次实验也让我获得满满的成就感。