
实验报告：pintos 解决忙等待

课程名称：操作系统

年级：2023 级

上机实践成绩：

指导教师：张民

姓名：张建夫

上机实践名称：pintos 解决忙等待

学号：

上机实践日期：

10235101477

11.4.2024

上机实践编号：

组号：

上机实践时间：14：

50~16：30

一、目的

明白忙等待的产生原因以及解决办法

二、内容与设计思想

操作系统中的忙等待指的是线程在应该休眠的时候并不休眠（例如该线程试图获取一个锁，而锁已经被获取了时，这里的例子与实验中的实际忙等待有一定区别，实验的忙等待是指定了睡眠时间，尽管意思一样），而是仅仅将 `cpu` 让出，但是，由于该操作只是让出 `cpu`，该线程在后面仍有被调度的可能，一旦该线程又被调度，如果此时锁还未释放，该线程只会浪费 `cpu` 时间以及切换线程的时间，这就是忙等待，该线程浪费 `cpu` 时间却不做事（只会让出 `cpu`）。

要解决忙等待问题，需要在该线程要休眠时把该线程拿出就绪队列，这样操作系统就不会调用该线程，因此，对于目前的 `pintos` 系统，`thread` 状态还没有 `thread_sleep` 的状态，我们需要加入这个状态并添加一个睡眠队列（这样操作系统知道什么时候唤醒队列中的人，以及唤醒谁），之后在 `timer_sleep` 函数和 `timer_interrupt` 函数中加入相关代码（这里睡眠的具体实现还要封装相应函数）即可。ps：这里之所以要改变 `timer_interrupt` 是因为 `timer_sleep` 是指定了睡眠时间的，每一次硬件的中断都代表时间过了 10ms，因此每次中断都要判断是否 `sleep` 足够时间了，而判断的代码就是在 `timer_interrupt` 函数中（实际上是在封装的函数里）。

三、使用环境

1. 主机 os: Windows-11
2. 虚拟机 os: Linux
3. 虚拟环境: docker 的 linux 镜像
4. 代码编辑器: vscode

四、实验过程

1. 展示忙等：

先展示一下忙等待的情况，我们在 `thread_yield` 函数里加了如下代码以展示忙等待：

```

/** 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 ();
    timer_print_stats ();
    printf ("线程%s放弃cpu\n", cur->name);
    if (cur != idle_thread) /*如果该线程不是idle空转线程*/
        if (!thread_mlfqs)
            priority_insert_threads (cur, &ready_list); /*调度线程时将线程加入就绪队列*/
    cur->status = THREAD_READY;
    schedule ();
    intr_set_level (old_level);
}

```

此处的 `printf` 语句和 `timer_print_stats` 函数（打印当前时间戳，是 `timer.c` 里面内置的函数，不是我自己写的）是为了满足实验和调试的要求（我 `github` 库里的则将这些 `printf` 加上注释了），运行 `pintos -v -- -q run alarm-multiple`:

[illegible]

```

Timer: 574 ticks
线程main放弃cpu
(alarm-multiple) thread 0: duration=10, iteration=1, product=10
(alarm-multiple) thread 0: duration=10, iteration=2, product=20
(alarm-multiple) thread 1: duration=20, iteration=1, product=20
(alarm-multiple) thread 0: duration=10, iteration=3, product=30
(alarm-multiple) thread 2: duration=30, iteration=1, product=30
(alarm-multiple) thread 3: duration=40, iteration=1, product=40
(alarm-multiple) thread 0: duration=10, iteration=4, product=40
(alarm-multiple) thread 1: duration=20, iteration=2, product=40
(alarm-multiple) thread 4: duration=50, iteration=1, product=50
(alarm-multiple) thread 0: duration=10, iteration=5, product=50
(alarm-multiple) thread 0: duration=10, iteration=6, product=60
Timer: 578 ticks
线程main放弃cpu
ct=60
(alarm-multiple) thread 1: duration=20, iteration=3, product=60
(alarm-multiple) thread 2: duration=30, iteration=2, product=60
(alarm-multiple) thread 0: duration=10, iteration=7, product=70
(alarm-multiple) thread 1: duration=20, iteration=4, product=80
(alarm-multiple) thread 3: duration=40, iteration=2, product=80
(alarm-multiple) thread 2: duration=30, iteration=3, product=90
(alarm-multiple) thread 4: duration=50, iteration=2, product=100
(alarm-multiple) thread 1: duration=20, iteration=5, product=100
(alarm-multiple) thread 3: duration=40, iteration=3, product=120
(alarm-multiple) thread 1: duration=20, iteration=6, product=120
(alarm-multiple) thread 2: duration=30, iteration=4, product=120
(alarm-multiple) thread 1: duration=20, iteration=7, product=140
(alarm-multiple) thread 2: duration=30, iteration=5, product=150
(alarm-Timer: 582 ticks
线程main放弃cpu
multiple) thread 4: duration=50, iteration=3, product=150
(alarm-multiple) thread 3: duration=40, iteration=4, product=160
(alarm-multiple) thread 2: duration=30, iteration=6, product=180
(alarm-multiple) thread 3: duration=40, iteration=5, product=200
(alarm-multiple) thread 4: duration=50, iteration=4, product=200
(alarm-multiple) thread 2: duration=30, iteration=7, product=210
(alarm-multiple) thread 3: duration=40, iteration=6, product=240
(alarm-multiple) thread 4: duration=50, iteration=5, product=250
(alarm-multiple) thread 3: duration=40, iteration=7, product=280
(alarm-multiple) thread 4: duration=50, iteration=6, product=300
(alarm-multiple) thread 4: duration=50, iteration=7, product=350
(alarm-multiple) end
Execution of 'alarm-multiple' complete.
Timer: 585 ticks
Thread: 0 idle ticks, 585 kernel ticks, 0 user ticks
Console: 154457 characters output
KeybTimer: 586 ticks
线程main放弃cpu
oard: 0 keys pressed
Powering off...
root@ec83d539f9b3:~/toolchain/pintos/src/threads#

```

可以从前面两张图看出来，该代码忙等待问题严重（上面直接将控制台的输出缓冲区占满了，pintos 初始化的部分都看不到了），线程 main 不断释放 cpu，又拿回 cpu，无效占用 cpu 时间，同时线程切换也是要时间的，这段时间也浪费了。

2.问题解决:

这个实验和 cs162 的解决办法一样，先修改 timer_sleep 函数:

```

87  /** Sleeps for approximately TICKS timer ticks. Interrupts must
88      be turned on. */
89  void timer_sleep(int64_t ticks) {
90      thread_sleep(ticks);
91  }
92

```

以下是 thread_sleep 函数的具体实现:

```
521  /*让当前进程休眠ticks时间*/
522  void thread_sleep(int64_t ticks)
523  {
524      if(ticks<=0)return;
525      struct thread*cur=thread_current();
526
527      enum intr_level old_level=intr_disable(); //关闭中断
528      if(cur!=idle_thread)
529      {
530          cur->status=THREAD_SLEEP;
531          cur->wake_time=timer_ticks()+ticks;
532          schedule();
533      }
534      intr_set_level(old_level); //恢复中断
535  }
```

注意此处要关闭中断，因为此处要读取硬件中断的总次数，如果此时中断了，读取的中断次数就不一样了。

接着，改变 timer_interrupt 函数，实现唤醒进程的功能：

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

以下是 wakeup_potential_sleep_thread 函数的具体实现：

```

537  /*检查是否有休眠进程*/
538  void wakeup_potential_sleep_thread()
539  {
540      struct list_elem*e;
541      int64_t cur_time=timer_ticks();
542      for(e=list_begin(&all_list);e!=list_end(&all_list);e=list_next(e))
543      {
544          struct thread*tmp=list_entry(e,struct thread,allelem);
545          enum intr_level old_level=intr_disable(); //关闭中断
546          if(tmp->status==THREAD_SLEEP&&tmp->wake_time<=cur_time)
547          {
548              /*唤醒进程*/
549              tmp->status=THREAD_READY;
550              timer_print_stats();
551              printf("线程%s醒来\n",tmp->name);
552              if(!thread_mlfqs)
553                  priority_insert_threads(tmp,&ready_list);
554          }
555          intr_set_level(old_level); //关闭中断
556      }
557  }

```

在执行唤醒线程的相关操作中，需要关闭中断是因为此处有线程加入到就绪队列的操作，该操作必须是原子的，否则在其他线程遍历就绪队列时会出现访问问题。

此处的 printf 语句和 timer_print_stats 函数作用和上面一样，是为了实验的调试要求，说明实现了忙等。此处有一个 if(!thread_mlfqs)，这个是方便后面实验多级队列实现的，暂且不谈。

实验结果：

```

root@83d539f9fb3:~/toolchain/pintos/src/threads# chmod +x test.sh
root@83d539f9fb3:~/toolchain/pintos/src/threads# ./test.sh alarm-multiple
cd build && make all
make[1]: Entering directory '/home/PKUS/toolchain/pintos/src/threads/build'
i386-elf-gcc -c ../threads/thread.c -o threads/thread.o -m32 -g -msoft-float -O0 -fno-stack-protector -nostdinc -I../.. -I../lib -I../lib/kernel -Wall -W -Wstrict-prototypes -Wmissing-prototypes -Wsystem-headers
-MD -MF threads/thread.d
../threads/thread.c: In function 'thread_unblock':
../threads/thread.c:241:3: warning: implicit declaration of function 'priority_insert_threads' [-Wimplicit-function-declaration]
priority_insert_threads(t,&ready_list); /*将线程加入就绪队列*/
^
../threads/thread.c: At top level:
../threads/thread.c:301:6: warning: no previous prototype for 'priority_insert_threads' [-Wmissing-prototypes]
void priority_insert_threads(struct thread *cur,struct list*list)
^
../threads/thread.c:301:6: warning: conflicting types for 'priority_insert_threads'
../threads/thread.c:241:3: note: previous implicit declaration of 'priority_insert_threads' was here
priority_insert_threads(t,&ready_list); /*将线程加入就绪队列*/
^
../threads/thread.c:522:6: warning: no previous prototype for 'thread_sleep' [-Wmissing-prototypes]
void thread_sleep(int64_t ticks)
^
../threads/thread.c: In function 'thread_sleep':
../threads/thread.c:531:20: warning: implicit declaration of function 'timer_ticks' [-Wimplicit-function-declaration]
cur->wake_time=timer_ticks()+ticks;
^
../threads/thread.c: At top level:
../threads/thread.c:538:6: warning: function declaration isn't a prototype [-Wstrict-prototypes]
void wakeup_potential_sleep_thread()
^
../threads/thread.c: In function 'wakeup_potential_sleep_thread':
../threads/thread.c:550:7: warning: implicit declaration of function 'timer_print_stats' [-Wimplicit-function-declaration]
timer_print_stats();
^
i386-elf-gcc -c ../devices/timer.c -o devices/timer.o -m32 -g -msoft-float -O0 -fno-stack-protector -nostdinc -I../.. -I../lib -I../lib/kernel -Wall -W -Wstrict-prototypes -Wmissing-prototypes -Wsystem-headers
-MD -MF devices/timer.d
../devices/timer.c: In function 'timer_sleep':
../devices/timer.c:94:3: warning: implicit declaration of function 'thread_sleep' [-Wimplicit-function-declaration]
thread_sleep(ticks);
^
../devices/timer.c: In function 'timer_interrupt':
../devices/timer.c:174:3: warning: implicit declaration of function 'wakeup_potential_sleep_thread' [-Wimplicit-function-declaration]
wakeup_potential_sleep_thread();
^
i386-elf-ld -melf_i386 -T threads/kernel.lds.s -o kernel.o threads/start.o threads/init.o threads/thread.o threads/switch.o threads/interrupt.o threads/intr-stubs.o threads/synch.o threads/palloc.o threads/malloc.o device
s/pit.o devices/timer.o devices/rtc.o devices/vga.o devices/serial.o devices/block.o devices/partition.o devices/ide.o devices/input.o devices/intq.o devices/rtc.o devices/shutdown.o devices/speaker.o lib/debug.o lib/rand
om.o lib/stdio.o lib/stdlib.o lib/string.o lib/strtime.o lib/strtime.o lib/kernel/debug.o lib/kernel/list.o lib/kernel/stdio.o lib/kernel/console.o tests/threads/tests.o tests/threads/alarm-wait.o test
s/threads/alarm-simultaneous.o tests/threads/alarm-priority.o tests/threads/alarm-zero.o tests/threads/alarm-negative.o tests/threads/priority-change.o tests/threads/priority-donate-one.o tests/threads/priority-donate-mul
tiple.o tests/threads/priority-donate-multiple2.o tests/threads/priority-donate-nest.o tests/threads/priority-donate-sema.o tests/threads/priority-donate-lower.o tests/threads/priority-fifo.o tests/threads/priority-preem
p.o tests/threads/priority-sema.o tests/threads/priority-condvar.o tests/threads/priority-donate-chain.o tests/threads/mlfqs-load-1.o tests/threads/mlfqs-load-60.o tests/threads/mlfqs-load-avg.o tests/threads/mlfqs-recent
-1.o tests/threads/mlfqs-fair.o tests/threads/mlfqs-block.o tests/threads/hello-world.o
i386-elf-objdump -S kernel.o > kernel.asm

```

运行命令（此处我将 make 命令和跑测试的命令写成一个脚本 test.sh，这样就不用分两

次敲了)

```
1386-elf-objdump -S kernel.o > kernel.asm
1386-elf-objcopy -O binary -n kernel.o > kernel.bin
1386-elf-objcopy -O binary -n .note -R .comment -S kernel.o kernel.bin
make[1]: Leaving directory '/home/PLUUS/toolchain/pintos/src/threads/build'
perl: warning: Setting locale failed.
perl: warning: Please check that your locale settings:
    LANGUAGE = (unset),
    LC_ALL = (unset),
    LANG = "en_US.UTF-8"
are supported and installed on your system.
perl: warning: Falling back to the standard locale ("C").
qemu-system-i386 -device isa-debug-exit -drive format=raw,media=disk,index=0,file=/tmp/dlzm_x3McI.dsk -m 4 -net none -nographic -monitor null
Pintos hdal
Loading.....
Kernel command line: q run alarm-multiple
Pintos booting with 3,068 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calling timer... 136,867,288 loops/s.
Boot complete.
Executing 'alarm-multiple':
(alarm-multiple) begin
(alarm-multiple) Creating 5 threads to sleep 7 times each.
(alarm-multiple) Thread 0 sleeps 10 ticks each time,
(alarm-multiple) thread 1 sleeps 20 ticks each time, and so on.
(alarm-multiple) If successful, product of iteration count and
(alarm-multiple) sleep duration will appear in nondescending order.
Timer: 136 ticks
线程thread 4醒来
Timer: 146 ticks
线程thread 0醒来
Timer: 146 ticks
线程thread 1醒来
Timer: 156 ticks
线程thread 0醒来
Timer: 156 ticks
线程thread 2醒来
Timer: 166 ticks
线程thread 0醒来
Timer: 166 ticks
线程thread 1醒来
Timer: 166 ticks
线程thread 2醒来
Timer: 176 ticks
线程thread 0醒来
Timer: 186 ticks
线程thread 0醒来
Timer: 186 ticks
线程thread 1醒来
线程thread 2醒来
Timer: 196 ticks
线程thread 0醒来
Timer: 206 ticks
线程thread 1醒来
Timer: 206 ticks
线程thread 3醒来
Timer: 216 ticks
线程thread 2醒来
Timer: 226 ticks
线程thread 1醒来
Timer: 226 ticks
线程thread 4醒来
Timer: 246 ticks
线程thread 1醒来
Timer: 246 ticks
线程thread 2醒来
Timer: 246 ticks
线程thread 3醒来
Timer: 266 ticks
线程thread 1醒来
Timer: 276 ticks
线程thread 2醒来
Timer: 276 ticks
线程thread 4醒来
Timer: 286 ticks
线程thread 3醒来
Timer: 306 ticks
线程thread 2醒来
Timer: 326 ticks
线程thread 3醒来
Timer: 326 ticks
线程thread 4醒来
Timer: 336 ticks
线程thread 2醒来
Timer: 366 ticks
线程thread 3醒来
Timer: 376 ticks
线程thread 4醒来
Timer: 406 ticks
线程thread 3醒来
Timer: 426 ticks
线程thread 4醒来
Timer: 476 ticks
线程thread 4醒来
Timer: 576 ticks
线程main醒来
(alarm-multiple) thread 0: duration=10, iteration=1, product=10
(alarm-multiple) thread 0: duration=10, iteration=2, product=20
(alarm-multiple) thread 1: duration=20, iteration=1, product=20
(alarm-multiple) thread 0: duration=10, iteration=3, product=30
(alarm-multiple) thread 2: duration=30, iteration=1, product=30
(alarm-multiple) thread 0: duration=10, iteration=4, product=40
(alarm-multiple) thread 1: duration=20, iteration=2, product=40
(alarm-multiple) thread 3: duration=40, iteration=1, product=40
(alarm-multiple) thread 0: duration=10, iteration=5, product=50
(alarm-multiple) thread 4: duration=50, iteration=1, product=50
(alarm-multiple) thread 0: duration=10, iteration=6, product=60
(alarm-multiple) thread 1: duration=20, iteration=3, product=60
(alarm-multiple) thread 2: duration=30, iteration=2, product=60
(alarm-multiple) thread 0: duration=10, iteration=7, product=70
(alarm-multiple) thread 1: duration=20, iteration=4, product=80
(alarm-multiple) thread 3: duration=40, iteration=2, product=80
(alarm-multiple) thread 2: duration=30, iteration=3, product=90
(alarm-multiple) thread 1: duration=20, iteration=5, product=100
(alarm-multiple) thread 4: duration=50, iteration=2, product=100
(alarm-multiple) thread 1: duration=20, iteration=6, product=120
(alarm-multiple) thread 2: duration=30, iteration=4, product=120
(alarm-multiple) thread 3: duration=40, iteration=3, product=120
(alarm-multiple) thread 1: duration=20, iteration=7, product=140
(alarm-multiple) thread 2: duration=30, iteration=5, product=150
(alarm-multiple) thread 4: duration=50, iteration=3, product=150
(alarm-multiple) thread 3: duration=40, iteration=4, product=160
(alarm-multiple) thread 2: duration=30, iteration=6, product=180
(alarm-multiple) thread 3: duration=40, iteration=5, product=200
(alarm-multiple) thread 4: duration=50, iteration=4, product=200
(alarm-multiple) thread 2: duration=30, iteration=7, product=210
(alarm-multiple) thread 3: duration=40, iteration=6, product=240
(alarm-multiple) thread 4: duration=50, iteration=5, product=250
(alarm-multiple) thread 3: duration=40, iteration=7, product=280
(alarm-multiple) thread 4: duration=50, iteration=6, product=300
(alarm-multiple) thread 4: duration=50, iteration=7, product=350
(alarm-multiple) end
Execution of 'alarm-multiple' complete.
Timer: 586 ticks
Thread: 550 idle ticks, 36 kernel ticks, 0 user ticks
Console: 4319 characters output
Keyboard: 0 keys pressed
Powering off...
root@cs3d539f9fb3:~/toolchain/pintos/src/threads#
```

可以看到线程 main 不再发生忙等待，只有一个“线程 main 醒来”（下图左下角），而没有线程 main 放弃进程的提示了。

个人 github 的代码库：https://github.com/saydontgo/school_OS_course

五、总结

本次实验很好的展现了忙等待的危害（未修改代码前，main 线程不断地释放和持有进程，cpu 时间白白浪费），同时让我明白了线程休眠状态的具体实现（虽然之前做过 cs162 的 project，但实际上对于一些实现细节，如关闭中断，并没有完全明白，这次再做一次的过程中，对于所有的实现细节就都明白了）。

