PintOS实验指南

实验准备

编译PintOS

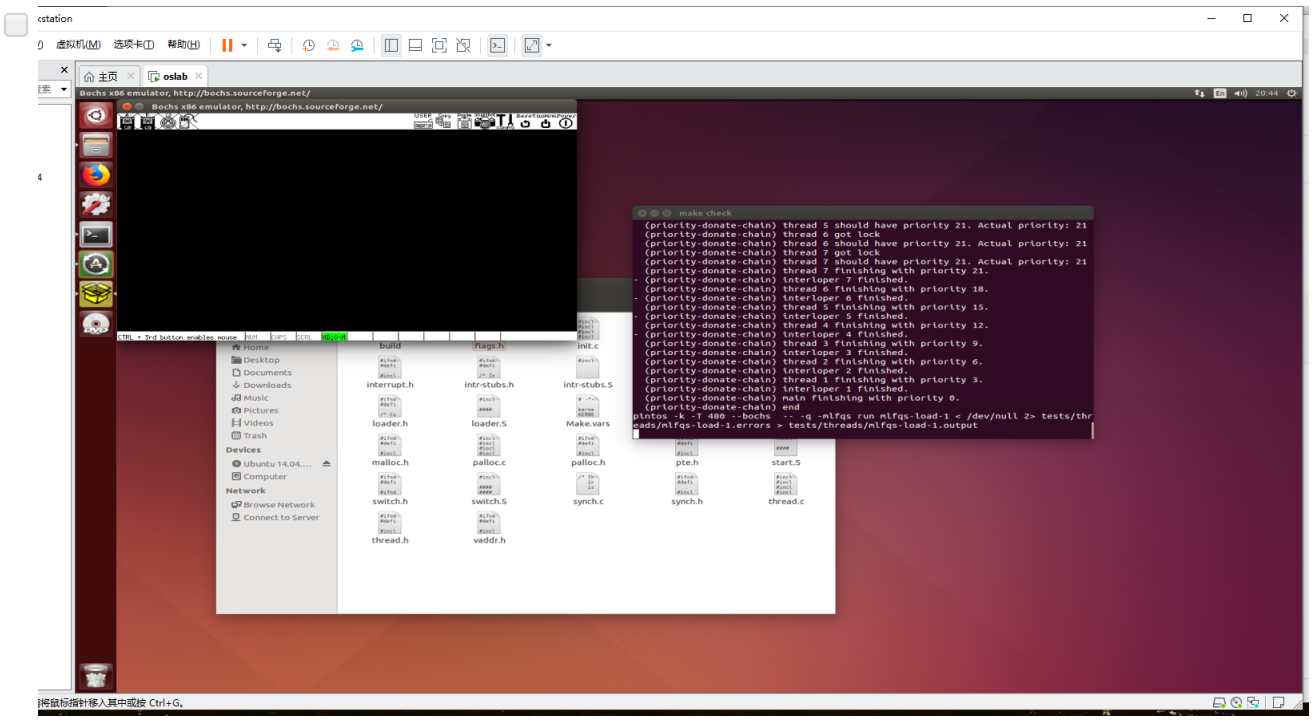
1. 前期准备工作见 [实验环境搭建]
2. 在`PintOS`根目录下执行以下操作

```
cd src
cd threads
make all
```

可编译本次线程实验的代码

测试你的代码

在`threads`目录下执行命令`make check`会执行自动测试程序



最后会给出你所有的测试点的通过情况

1.
```
pass tests/threads/alarm-single
pass tests/threads/alarm-multiple
pass tests/threads/alarm-simultaneous
pass tests/threads/alarm-priority
pass tests/threads/alarm-zero
pass tests/threads/alarm-negative
pass tests/threads/priority-change
pass tests/threads/priority-donate-one
pass tests/threads/priority-donate-multiple
pass tests/threads/priority-donate-multiple2
pass tests/threads/priority-donate-nest
pass tests/threads/priority-donate-sema
pass tests/threads/priority-donate-lower
pass tests/threads/priority-fifo
pass tests/threads/priority-preempt
pass tests/threads/priority-sema
pass tests/threads/priority-condvar
```

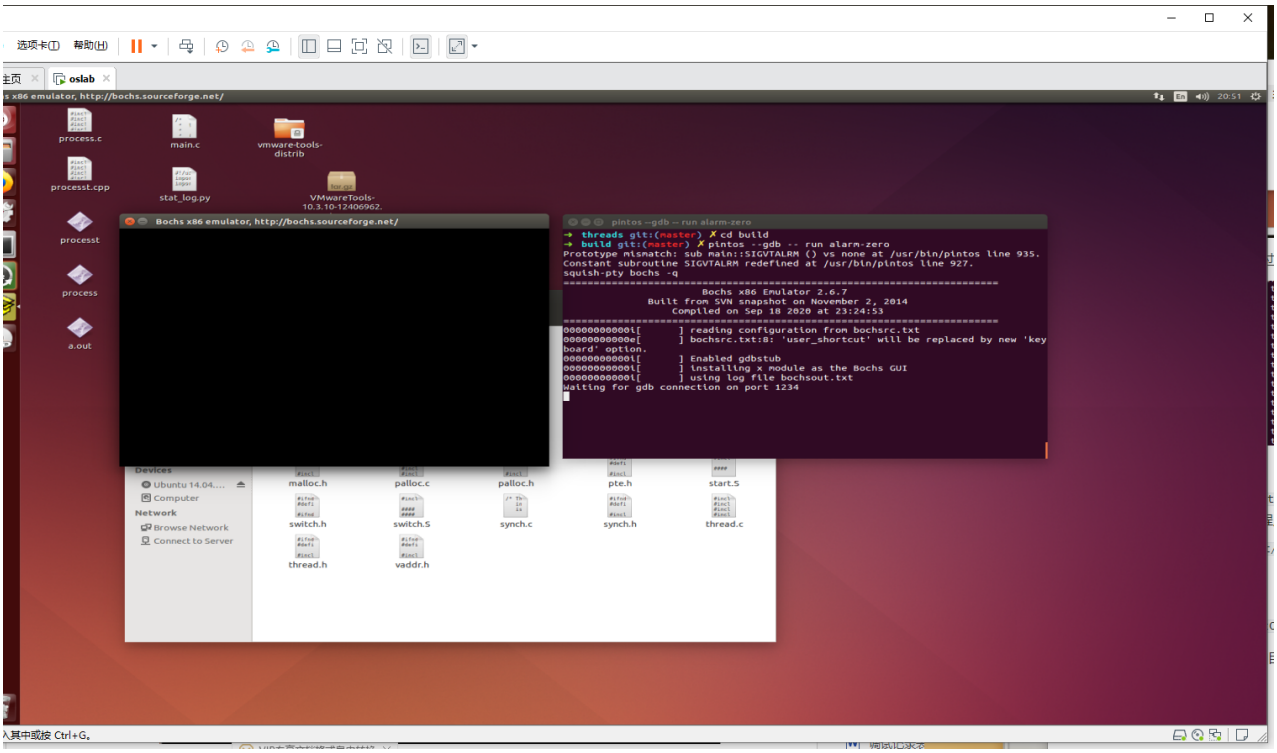
HINT

- 在`src/threads/build/tests/threads`里会存放类似`xxxx.result`后缀的文件，里面会存放此测试点期望的输出和实际的程序输出
- 本次实验的测试代码放在`src/tests/threads`中

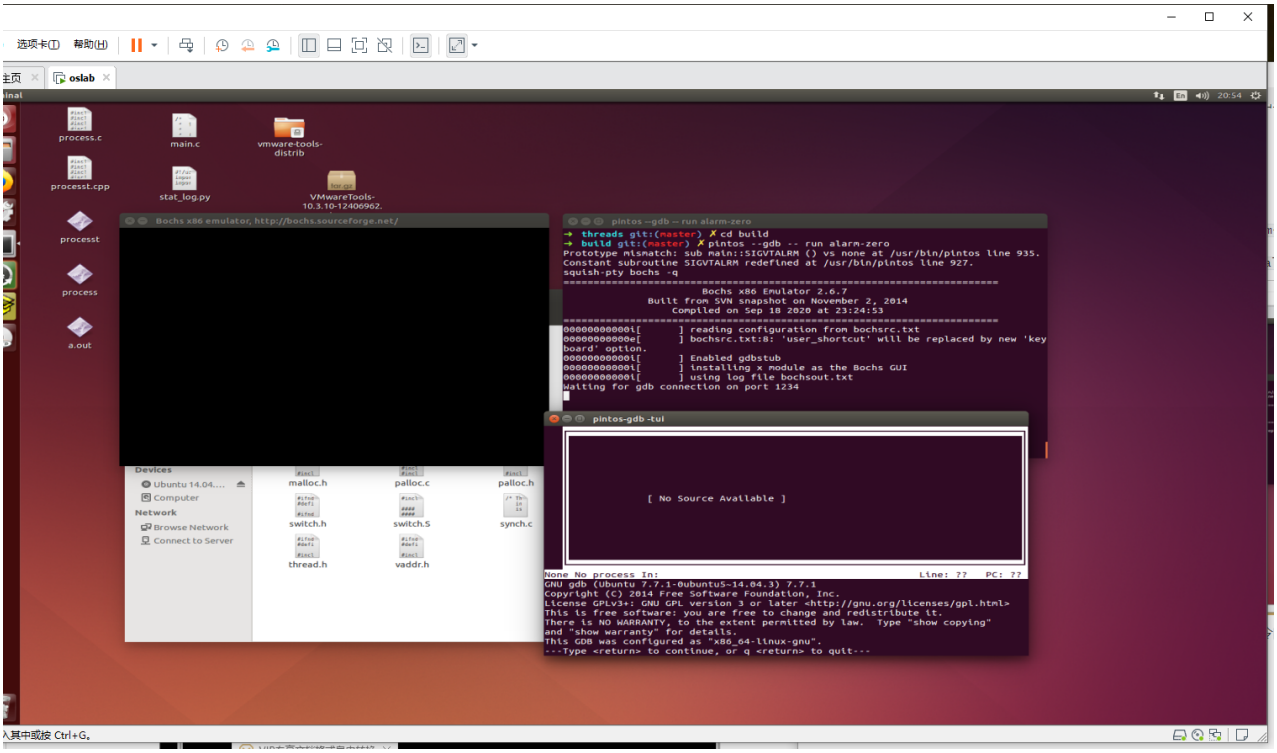
Debug方法

这里推荐使用`gdb`进行调试,`PintOS`使用`gdb`调试的方法如下(以调试)`alarm-zero`测试用例为例

1. 切换到`src/threads/build`目录下,输入命令`pintos --gdb -- run alarm-zero`



2. 新打开一个`terminal`，同样进入`src/threads/build`目录下,输入命令`pintos-gdb -tui`



3. 按下回车键,之后`gdb`连接`1234`端口(以第一个终端中显示的`port`号为准),`target remote localhost:1234`

```
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) target remote localhost:1234
Remote debugging using localhost:1234
Error while running hook_stop:
No symbol table is loaded.  Use the "file" command.
0x0000fff0 in ?? ()
(gdb)
```

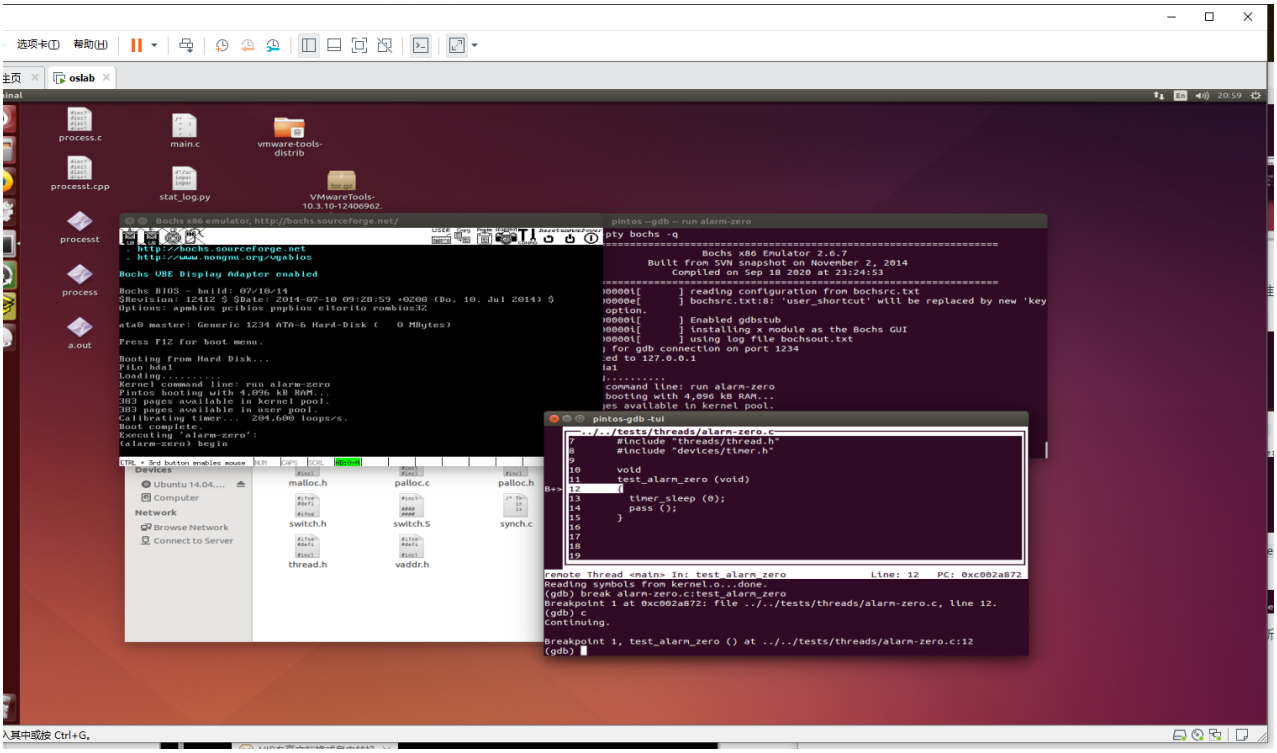
4. 之后选择要调试的文件,我们调试`kernel.o`，输入命令`file kernel.o`

```
remote Thread <main> In:                                  Line: ??    PC: 0xfff0
Error while running hook_stop:
No symbol table is loaded.  Use the "file" command.
0x0000fff0 in ?? ()
(gdb) file kernel.o
A program is being debugged already.
Are you sure you want to change the file? (y or n)
Reading symbols from kernel.o...done.
(gdb)
```

5. 之后我们设置断点，假如我们调试`test_alarm_zero`，则输入命令:`break alarm-zero.c:test_alarm_zero`

```
(gdb) break alarm-zero.c:test_alarm_zero
Breakpoint 1 at 0xc002a872: file ../../tests/threads/alarm-zero.c, line 12.
```

6. 之后可以进行调试了，我们按下`c`，代表`continue`，会跳到我们设置断点的那一行



7. 之后可以按`n`跳到下一行，或者`step`进入函数里，调试过程中可以使用`print 变量`显示变量的内容，详细的`gdb`使用方法请自行百度,这里只介绍几个最常用的

实验内容

第一部分

要求实现`devices/timer.c`中的`timer_sleep`函数，此函数在原来的`PintOS`中有实现，但是使用的是忙等待的机制，需要我们修改此函数使其功能不变，但是不是忙等待。

要想不轮询，很自然的想到了像`wait()`这种阻塞线程的操作，同理，这里我们可以将线程阻塞，之后等待睡眠时间足够了再将线程唤醒。

如何等待睡眠时间足够呢？答案是**时间中断**，我们可以在时间中断的中断处理函数中稍作修改，就可以轻松实现第一题的要求

第二部分

第二部分是这个实验比较麻烦的一部分，主要是为了解决优先级翻转的问题。什么是优先级翻转呢，比如有三个优先级分别为高，中，低的线程，中等优先级的线程就绪，高优先级的线程正阻塞在低优先级线程所持有的锁上。那么下次调度的时候若按优先级来调度，则将选择就绪态的中优先级线程。这样看来，中优先级的线程反而比高优先级的线程获得了更多的执行机会。

怎么解决这个问题呢，我们需要让优先级更高的线程获得更高的执行机会。一种可行的办法是优先级捐赠，也就是在高优先级线程阻塞在低优先级线程上的时候，暂时把低优先级的线程的优先级提高，这样下次调度执行的就是这个持有锁的原本优先级很低的线程，此线程把锁释放掉以后高优先级的线程就可以得以执行了，注意此时被暂时提高优先级的线程的优先级需要被重置。

除了这种最基本的情况，我们需要考虑复杂一些的情况

- 如果一个线程持有了好几个锁，则需要被多个线程捐赠，那么此时应该怎么处理？

- 在一个线程获取一个锁的时候， 如果拥有这个锁的线程优先级比自己低就提高它的优先级，并且如果这个锁还被别的锁锁着，那么此时该怎么处理？
- 在对一个线程进行优先级设置的时候， 如果这个线程处于被捐赠状态，则此线程释放锁以后处于未被捐赠状态时的优先级应该设置为什么？有哪几种情况？
- 线程优先级改变的时候会发生抢占吗？
- ……

这些问题都可以通过之前提到的测例的逻辑来推断，**通过比对测例的期望输出与测例中的输出可以得出多个线程的执行顺序**，

第三部分

第三部分主要实现多级反馈队列调度算法，这里就不用做优先级捐赠了，这里就根据BSD Scheduler一步步来就可以了，没有涉及太多操作系统的内容，算是大半个算法实现题。

1. 首先`PintOS`内核中不支持浮点运算，我们首先要实现一系列浮点运算，实现以下操作

Convert n to fixed point:	n * f
Convert x to integer (rounding toward zero):	x / f
Convert x to integer (rounding to nearest):	(x + f / 2) / f if x >= 0, (x - f / 2) / f if x <= 0.
Add x and y:	x + y
Subtract y from x:	x - y
Add x and n:	x + n * f
Subtract n from x:	x - n * f
Multiply x by y:	((int64_t) x) * y / f
Multiply x by n:	x * n
Divide x by y:	((int64_t) x) * f / y
Divide x by n:	x / n

2. 之后我们实现这个算法，这个算法主要通过动态修改`niceness`，`priority`，`recent_cpu`，`load_avg`这几个变量来动态修改每个线程的优先级，更新触发的时机和时钟的`ticks`有关，所以这里我们又需要在`timer_interrupt`时钟中断处理函数做一些修改了