

实验报告：pintos 修改 testcase 实验

课程名称：操作系统实践

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 修改 testcase

学号 10235101477

上机实践日期 2024/10/14

上机实践编号：

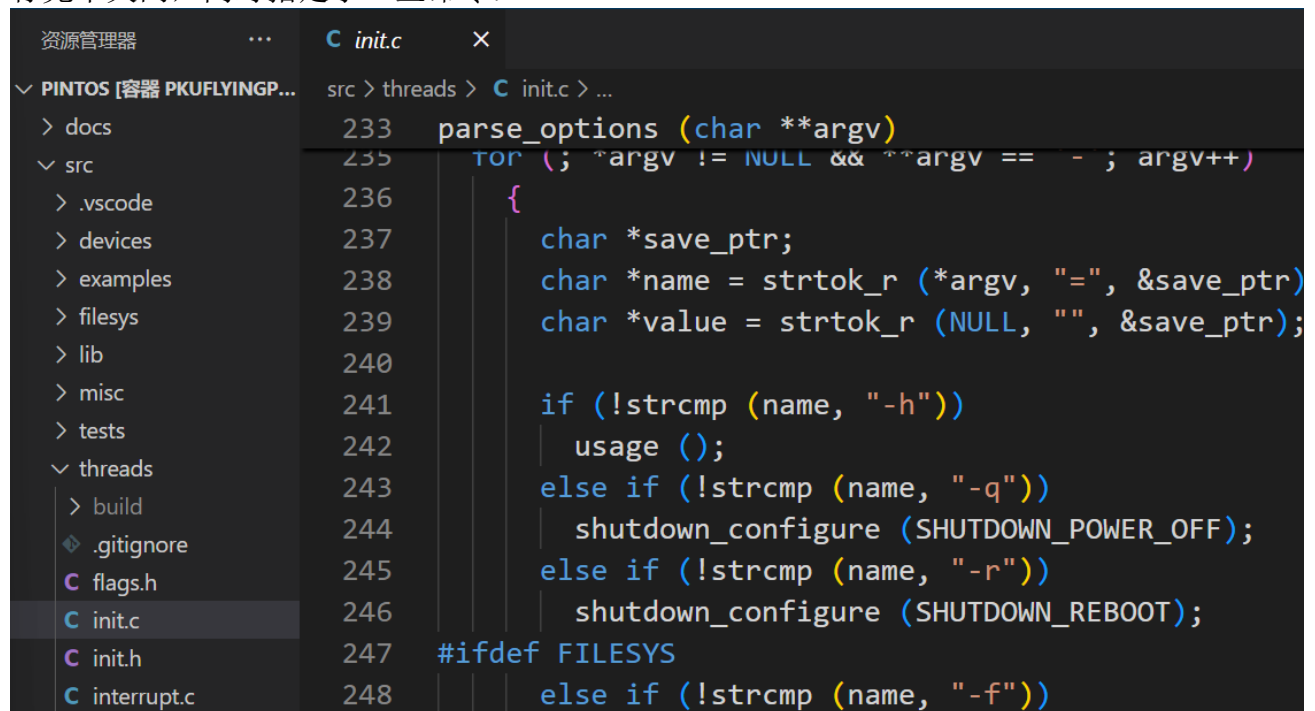
组号：

上机实践时间：

14:50~16:30 点

1. 前置知识：

pku 的 pintos 与 cs162 的 pintos 跑测试的方式略有不同，前者需要加上 -q 选项以防止运行完不关闭，同时指定了一些命令：



后面都需要加上被测试的名字进行测试。

以下是一个用户程序加载进 pintos 内存并运行的全过程：

在 pintos 完成加载后，如果命令行有参数传进（即传入被测试程序），pintos_init 函数会调用 run_actions，根据传入的命令（例如 run）执行对应操作，对于用户程序，一般是 run。对于 run，run_actions 会调用 run_task，又由于是用户程序，接着会调用：

```
#ifdef USERPROG
    process_wait (process_execute (task));
```


process_execute 会创建一个子线程（pintos 中并没有进程的概念，pintos 中任何进程都是由线程模拟的），创建完成后，这个新子线程会运行 start_process 函数，设置该子进程相应堆栈信息（例如设置栈指针，将命令行参数推入栈等），从子进程调用 start_process 开始，子进程和创建它的进程就开始并发进行（尽管这里调用了 process_wait，但实际上这个

函数尚未实现，要实现 wait 系统调用才有用），start_process 中通过模拟从中断返回来执行正式用户进程（即自己写的程序）。

在用户程序运行完之后，会调用 process_exit, 结束进程。

2. 完成过程:

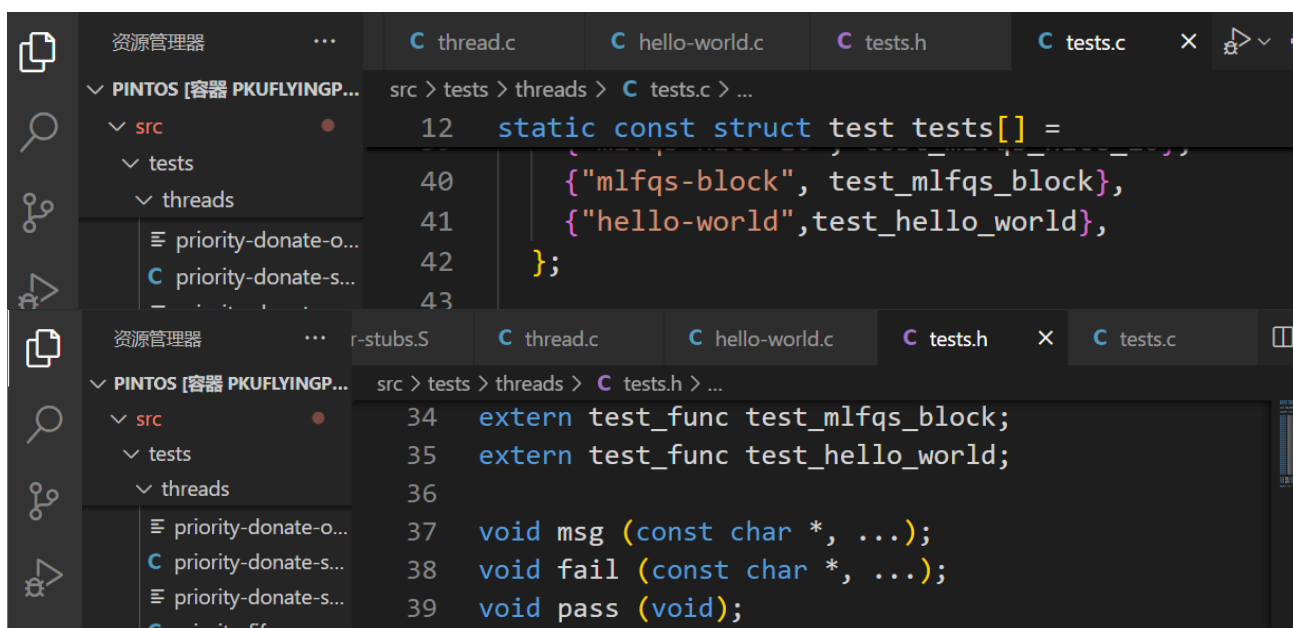
先在 src/tests/threads 中 touch 一个 hello-world.c, 在 vscode 中找到 c 源文件，编写如下代码：



```

src > tests > threads > C hello-world.c > test_hello_world(void)
1  #include<stdio.h>
2  #include "tests/threads/tests.h"
3  void test_hello_world(void){
4      printf("hello world!\n");
5  }
  
```

接着在 test.c 和 test.h 中加入源文件中编写的函数：

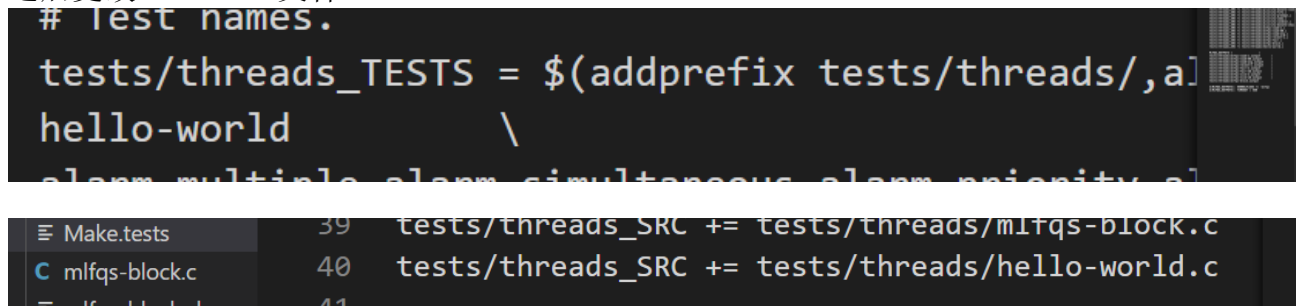


```

src > tests > threads > C tests.c > ...
12  static const struct test tests[] =
40      {"mlfqs-block", test_mlfqs_block},
41      {"hello-world", test_hello_world},
42  };
43

src > tests > threads > C tests.h > ...
34  extern test_func test_mlfqs_block;
35  extern test_func test_hello_world;
36
37  void msg (const char *, ...);
38  void fail (const char *, ...);
39  void pass (void);
  
```

之后更改 Make.test 文件：



```

# test names.
tests/threads_TESTS = $(addprefix tests/threads/,a
hello-world \
alarm-multiple-alarm-simultaneous-alarm-priority-a

39  tests/threads_SRC += tests/threads/mlfqs-block.c
40  tests/threads_SRC += tests/threads/hello-world.c
41
  
```

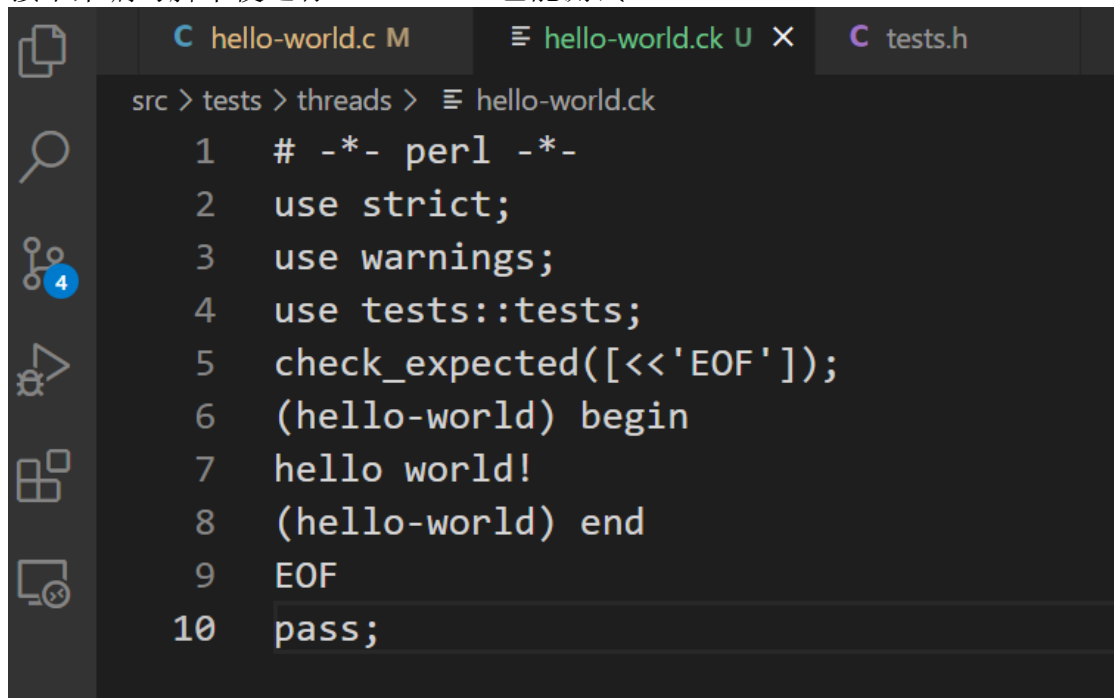
这样可以使 make check 访问到这个新测试文件，同时编译器也能知道源码在哪个文件里编译。

至此，新的 testcase 创建完成，回到/pintos/src/threads 里重新 make，执行 `pintos -- -q run hello-world`：

```
root@c83d539f9fb3:~/toolchain/pintos/src/threads# pintos -- -q run hello-world
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/8jq0x85_rK.d
sk -m 4 -net none -nographic -monitor null
Pintos hda1
Loading.....
Kernel command line: -q run hello-world
Pintos booting with 3,968 kB RAM...
367 pages available in kernel pool.
367 pages available in user pool.
Calibrating timer... 117,760,000 loops/s.
Boot complete.
Executing 'hello-world':
(hello-world) begin
hello world!
(hello-world) end
Execution of 'hello-world' complete.
Timer: 25 ticks
Thread: 0 idle ticks, 25 kernel ticks, 0 user ticks
Console: 385 characters output
Keyboard: 0 keys pressed
Powering off...
```

成功执行！

接下来编写脚本使运行 `make check` 也能测试 `hello-world.c`：



```
src > tests > threads > hello-world.ck
1  # -*- perl -*-
2  use strict;
3  use warnings;
4  use tests::tests;
5  check_expected([<<'EOF']);
6  (hello-world) begin
7  hello world!
8  (hello-world) end
9  EOF
10 pass;
```

(btw，这个脚本语言比 shell 脚本语言规范好多)

再运行 `make check`，成功通过该用例：

```
pintos -v -k -T 60 --bochs -- -q run hello-world < /dev/null 2> tests/threads/hello-world.errors
> tests/threads/hello-world.output
perl -I../.. ../../tests/threads/hello-world.ck tests/threads/hello-world tests/threads/hello-world
.result
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").
pass tests/threads/hello-world
```

3. 实验代码与结果:

个人 github 仓库 url: (实验结果的截图已在实验过程中包括)

https://github.com/saydontgo/school_OS_course

4. 报告总结:

本次实验收获颇丰。最大的收获是知道如何编写一个测试用例了(之前 cs162 实验就有一个小任务是编写两个测试用例,但我当时根本不知道怎么写,搜遍全网也没有答案,这次可以把这个小任务也一起完成了),其次就是了解了一门新的脚本语言,之前只会用 shell 语言写脚本,语法一言难尽,但这个脚本语言就很规范,且专门用于测试结果检查(最近正好在学正则,可以用这个脚本练正则了)。除此之外,还了解到了 perl 解释器,先前只知道 bash 解释器。最后一个特别的收获是这个镜像已内置 openssh(之前一直以为没有),就可以用 ssh 的方式将代码 push 到个人仓库了,同时也可以在本地的机子上 ssh 到容器内部,更为安全和便捷。