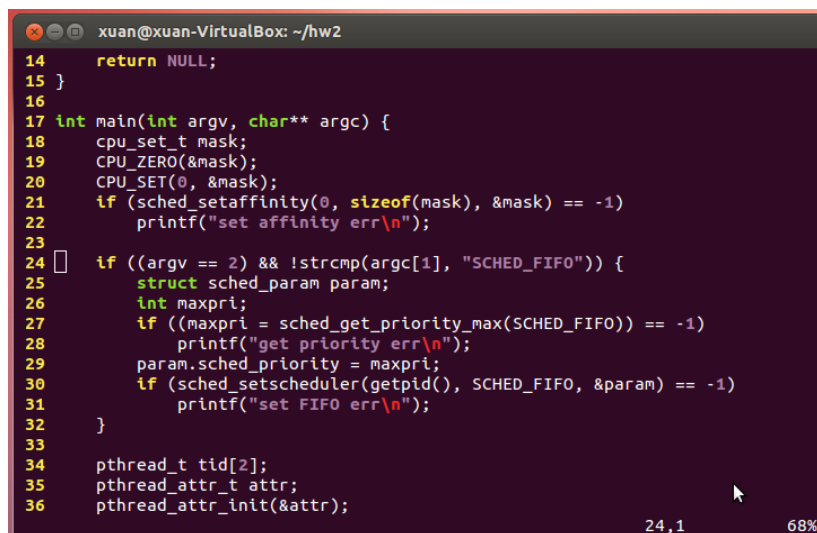


Part 1

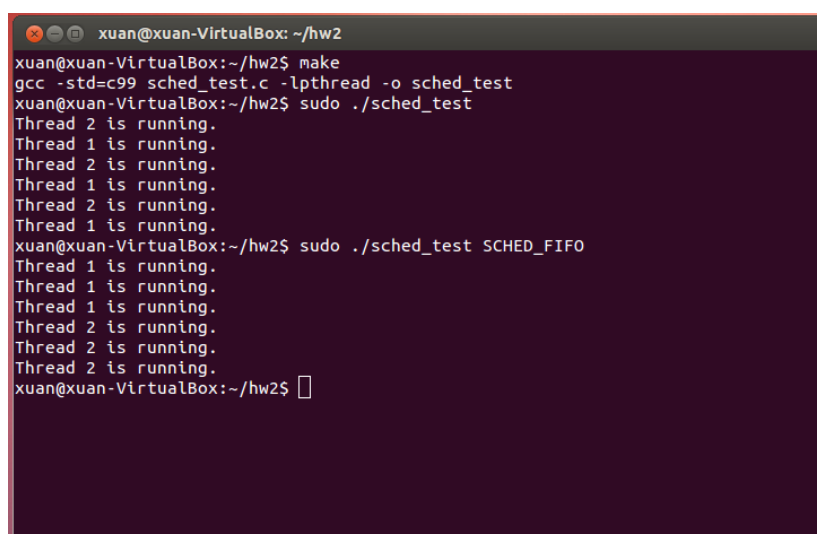
實作細節：

用 sched.h 中的 sched_setaffinity 設定 cpu affinity，再來用 sched.h 中的 sched_setscheduler 設定 sched_policy 為 FIFO，並且 create 出兩個 thread 觀察其表現。

結果如第二張圖，在不指定 policy 的情況下，Linux 預設是 RR 排程，會交互輸出結果，而在指定 FIFO 之後，thread1 完成執行後才會換到 thread2。



```
xuan@xuan-VirtualBox: ~/hw2
14     return NULL;
15 }
16
17 int main(int argv, char** argc) {
18     cpu_set_t mask;
19     CPU_ZERO(&mask);
20     CPU_SET(0, &mask);
21     if (sched_setaffinity(0, sizeof(mask), &mask) == -1)
22         printf("set affinity err\n");
23
24     if ((argv == 2) && !strcmp(argc[1], "SCHED_FIFO")) {
25         struct sched_param param;
26         int maxpri;
27         if ((maxpri = sched_get_priority_max(SCHED_FIFO)) == -1)
28             printf("get priority err\n");
29         param.sched_priority = maxpri;
30         if (sched_setscheduler(getpid(), SCHED_FIFO, &param) == -1)
31             printf("set FIFO err\n");
32     }
33
34     pthread_t tid[2];
35     pthread_attr_t attr;
36     pthread_attr_init(&attr);
```



```
xuan@xuan-VirtualBox:~/hw2$ make
gcc -std=c99 sched_test.c -lpthread -o sched_test
xuan@xuan-VirtualBox:~/hw2$ sudo ./sched_test
Thread 2 is running.
Thread 1 is running.
Thread 2 is running.
Thread 1 is running.
Thread 2 is running.
Thread 1 is running.
xuan@xuan-VirtualBox:~/hw2$ sudo ./sched_test SCHED_FIFO
Thread 1 is running.
Thread 1 is running.
Thread 1 is running.
Thread 2 is running.
Thread 2 is running.
Thread 2 is running.
xuan@xuan-VirtualBox:~/hw2$
```

討論：

1. 因為利用很大的 for 迴圈進行 busy waiting，所以在編譯的時候不能開啟

1. 如果在懶人包載下來之後直接編譯核心，重開機後直接執行測試程式，`process` 會一直卡在 `kernel mode`，用 `sudo kill -9 pid` 也無法強制結束。變成一隻殺不死也不會結束的 `process`。
2. 測試程式沒有設定 `cpu affinity`，如果直接跑測試程式，很可能因為兩個 `thread` 在不同的核心上執行產生 `race condition` 一起寫入 `buffer`，而得到像 `bcbcbcbcbcbcbcbcbc` 這樣的序列。

[illegible]