簡上博 · Follow
Last edited by 簡上博 on Nov 7, 2024
Contributed by

作業說明：https://hackmd.io/@fLANt9b6TbWx5I3lYKkBow/rJKfzGY1Jx

報告網址：https://hackmd.io/@iTkWej8WRj2Xrnb85FtlPQ/BJX49OOZkx

I. Describe how you implemented the program in detail. (10%)

1. main 接收命令列參數

```
int main(int argc, char* argv[])
```

2. 解析使用者命令

- 用 `getopt` 來 parse，設置 `"n:t:s:p:"` 讀取各參數後的值
- 把 parse 完的各個 thread 的資料存到 `num_threads`、`time_wait`、`policies`、`priorities`

```
int opt;
int num_threads = 0;
float time_wait = 0;
vector<string> policies;
vector<int> priorities;
while ((opt = getopt(argc, argv, "n:t:s:p:")) != -1) {
        switch (opt) {
            case 'n':{
                num_threads = atoi(optarg);
                break;
            }
            case 't':{
                time_wait = atof(optarg);
                break;
            }
            case 's':{
                char *policies_str = optarg;
                char *policy;
                policy = strtok(policies_str, ",");
                int i = 0;
                while (policy != NULL && i < num_threads) {
                    policies.push_back(string(policy));
                    policy = strtok(NULL, ",");
                    i++;
                }
                break;
            }
            case 'p':{
                char* priorities_str = optarg;
                char *priority;
                priority = strtok(priorities_str, ",");
                int i = 0;
                while (priority != NULL && i < num_threads) {
                    priorities.push_back(stoi(string(priority)));
                    priority = strtok(NULL, ",");
                    i++;
                }
                break;
            }
            default:
                cout << "Please follow the pattern below:" << "\n";
                cout << argv[0] << " -n <num_threads> -t <time_wait> -s <policies>
                return -1;
        }
    }
```

3. 宣告 pthread_barrier 和 thread_info

- 建立 struct 存 thread 資料

```
pthread_barrier_t barrier; // global variable
```

```
typedef struct {
    pthread_t thread_id; //typedef unsigned long pthread_t
    int thread_num;
    float time_wait;
    int thread_sched_policy;
    int thread_sched_priority;
} thread_info_t;
```

```
pthread_barrier_init(&barrier, NULL, num_threads + 1); // the created threads and m
```

```
thread_info_t thread_info[num_threads];
```

4. Set CPU affinity

- 設置執行 threads 的 CPU，讓所有 thread 在相同 CPU 下執行

```
#define _GNU_SOURCE
```

```
int cpu_id = 0;
cpu_set_t mask;
CPU_ZERO(&mask);
CPU_SET(cpu_id, &mask);
if (sched_setaffinity(0, sizeof(mask), &mask) != 0) {
    cout << "Error: sched_setscheduler\n";
}
```

5. Set the attributes to each thread and create the threads

- 設置各個 thread 的資料（`thread_num`、`time_wait`、`thread_sched_policy`、`thread_sched_priority`）
  - 用 `get_policy_num()` 把 policies 轉為數字
- 依據各個 thread 的資料設置 policy
- 不繼承 parent 的 schduling policy 才能設置自己的 policy
  `pthread_attr_setinheritsched` 設 `PTHREAD_EXPLICIT_SCHED`
- scheduling policy 若為 NORMAL，則不需設置，直接建立 thread
- 建立 thread
  `pthread_create(&thread_id, &thread_attr, thread_func, &arg)`
  - 3rd variable (thread_func) 是 thread 要執行的 function
  - 4th variable (arg) 是該 function 的 arguments
- 最後設置 pthread_barrier ，擋住 main thread
  `pthread_barrier_wait(&barrier);`

```
#define SCHED_NORMAL 0
#define SCHED_FIFO 1
int get_policy_num(string policy){
    if(policy == "NORMAL"){
        return SCHED_NORMAL;
    }
    else{
        return SCHED_FIFO;
    }
}
```

```
for (int i = 0; i < num_threads; i++) {
        thread_info[i].thread_num = i;
        thread_info[i].time_wait = time_wait;
        thread_info[i].thread_sched_policy = get_policy_num(policies[i]);
        thread_info[i].thread_sched_priority = priorities[i];

        if (thread_info[i].thread_sched_policy == SCHED_FIFO) {
            pthread_attr_t pthread_attr;
            struct sched_param param;

            pthread_attr_init(&pthread_attr);
            // do not inherit the parent's scheduling policy to set the thread's
policy.
            pthread_attr_setinheritsched(&pthread_attr, PTHREAD_EXPLICIT_SCHED);

            // set the scheduling policy
            if (pthread_attr_setschedpolicy(&pthread_attr, SCHED_FIFO) != 0) {
                cout << "Error: SCHED_FIFO pthread_attr_setschedpolicy\n";
            }

            param.sched_priority = thread_info[i].thread_sched_priority; // set
priority

            // set the scheduling paramters
            if (pthread_attr_setschedparam(&pthread_attr, &param) != 0) {
                cout << "Error: SCHED_FIFO pthread_attr_setschedparam\n";
            }

            if (pthread_create(&thread_info[i].thread_id, &pthread_attr,
thread_func, &thread_info[i]) != 0) {
                cout << "Error: Thread " << thread_info[i].thread_num <<
"SCHED_FIFO pthread_create\n";
            }

        }
        else if (thread_info[i].thread_sched_policy == SCHED_NORMAL) {
            if (pthread_create(&thread_info[i].thread_id, NULL, thread_func,
&thread_info[i]) != 0){
                cout << "Error: Thread " << thread_info[i].thread_num <<
"SCHED_NORMAL pthread_create\n";
            }
        }
        else {
            cout << "The policy supports only NORMAL and FIFO!\n";
        }
    }
pthread_barrier_wait(&barrier);
```

6 . 設置 void *thread_func(void *arg)

- 設置 pthread_barrier 讓 thread 一起執行
  `pthread_barrier_wait(&barrier);`
- 設置 busy waiting，讓 thread 持續占用 CPU

```
void *thread_func(void *arg)
{
    thread_info_t *thread_info = (thread_info_t *)arg;

    /* 1. Wait until all threads are ready */
    // printf("Thread %d sched_getcpu = %d\n", thread_info->thread_num,
sched_getcpu());
    pthread_barrier_wait(&barrier);

    /* 2. Do the task */
    for (int i = 0; i < 3; i++) {
        printf("Thread %d is starting\n", thread_info->thread_num);
        /* Busy for <time_wait> seconds */
        auto start_time = chrono::high_resolution_clock::now();
        auto end_time = start_time + chrono::milliseconds(int(thread_info-
>time_wait*1000));
        auto time = chrono::high_resolution_clock::now();
        while (time < end_time) {
            time = chrono::high_resolution_clock::now();
            // printf("Thread %d is running\n", thread_info->thread_num);
            // auto now = chrono::high_resolution_clock::to_time_t(time);
            // cout << ctime(&now);  // This will print a readable time format
        }
    }

    /* 3. Exit the function  */
    pthread_exit(NULL);
}
```

7. 回收資源

- 等待所有 thread 執行完，回收 thread 資源
- 回收 pthread_barrier

```
for (int i = 0; i < num_threads; i++) {
    pthread_join(thread_info[i].thread_id, NULL);
}
pthread_barrier_destroy(&barrier);
```

8. 製作 Makefile

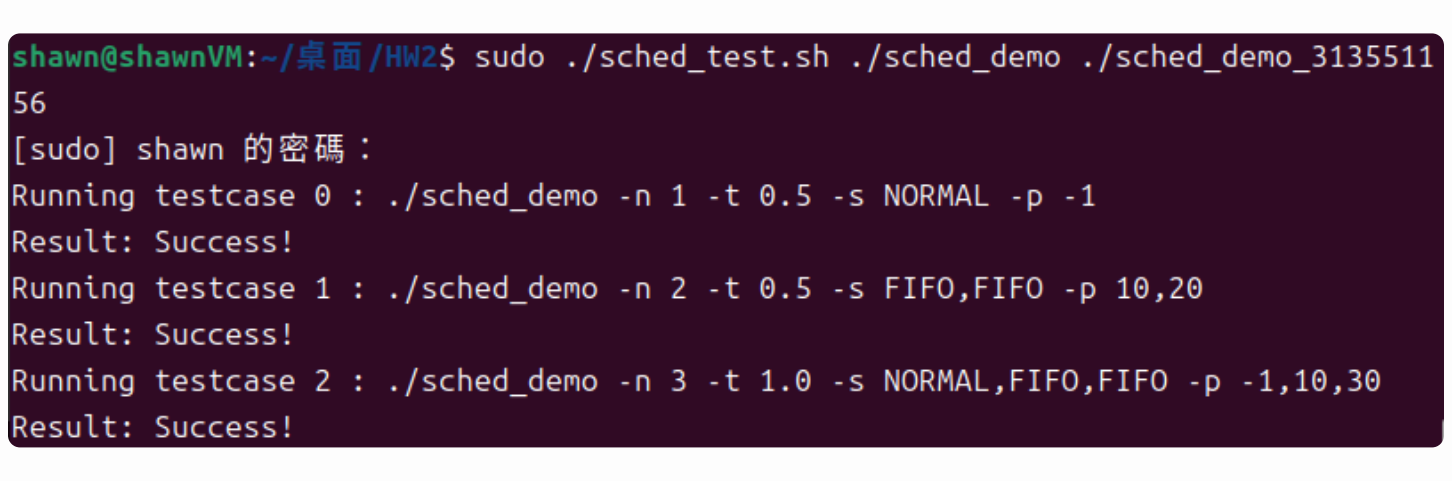
- 輸出執行檔

```
# indicating that target "all" and "clean" are not files
.PHONY: all clean

# set some variables
CC= g++
CFLAGS= -Wall -Wextra -O3 -Wpedantic
LDFLAGS = -lpthread
OUTPUT_OPTION= -MMD -MP -o $@

SOURCE= sched_demo_313551156.cpp
OBJS= $(SOURCE:.cpp=.o)
DEPS= $(SOURCE:.cpp=.d)
TARGET= sched_demo_313551156

# first command of make
all: $(TARGET)

-include $(DEPS)

# implicit targets
# %.o: %.c
#        $(CC) $^ -o $@ -c $(CFLAGS)

$(OBJS): $(SOURCE)
        $(CC) $(CFLAGS) -c $< -o $@

$(TARGET): $(OBJS)
        $(CC) $(OBJS) $(LDFLAGS) -o $@

clean:
        @rm -f $(TARGET) $(OBJS) $(DEPS)
```

9. 成果展現

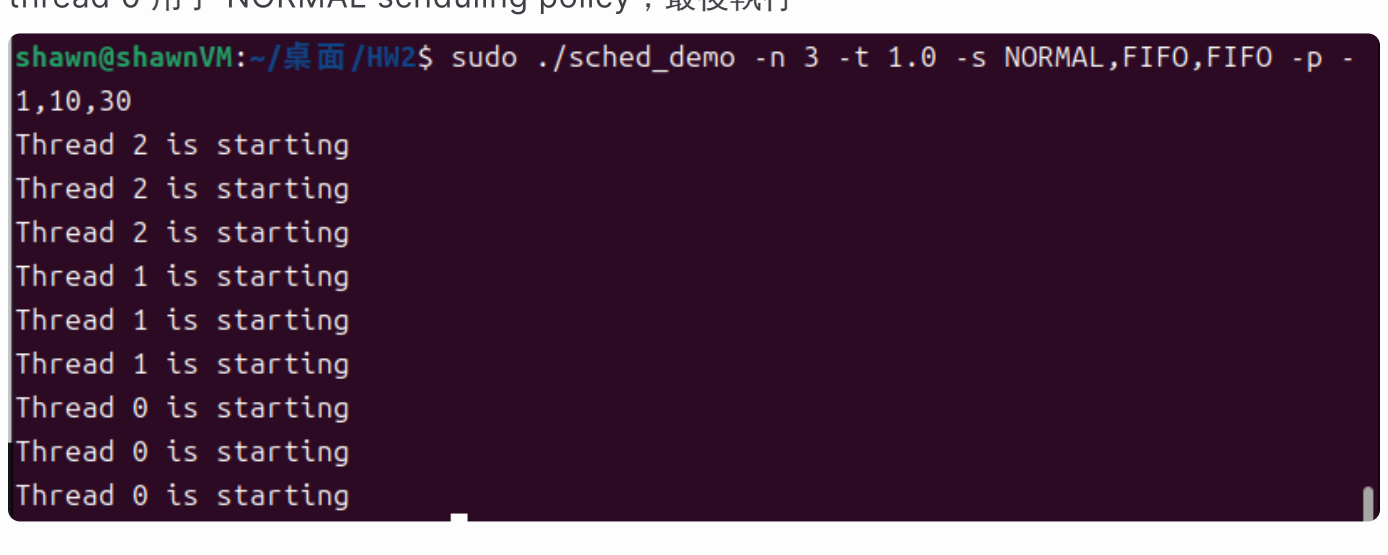
- 修改 kernel 參數，讓 CPU 不會因為 timeout 被 switch
  在 terminal 中輸入 `sudo sysctl kernel.sched_rt_runtime_us=1000000`
- run the test cases

```
make
sudo ./sched_test.sh ./sched_demo ./sched_demo_313551156
```


shawn@shawnVM:~/桌面/HW2$ sudo ./sched_test.sh ./sched_demo ./sched_demo_3135511
56
[sudo] shawn 的密碼：
Running testcase 0 : ./sched_demo -n 1 -t 0.5 -s NORMAL -p -1
Result: Success!
Running testcase 1 : ./sched_demo -n 2 -t 0.5 -s FIFO,FIFO -p 10,20
Result: Success!
Running testcase 2 : ./sched_demo -n 3 -t 1.0 -s NORMAL,FIFO,FIFO -p -1,10,30
Result: Success!

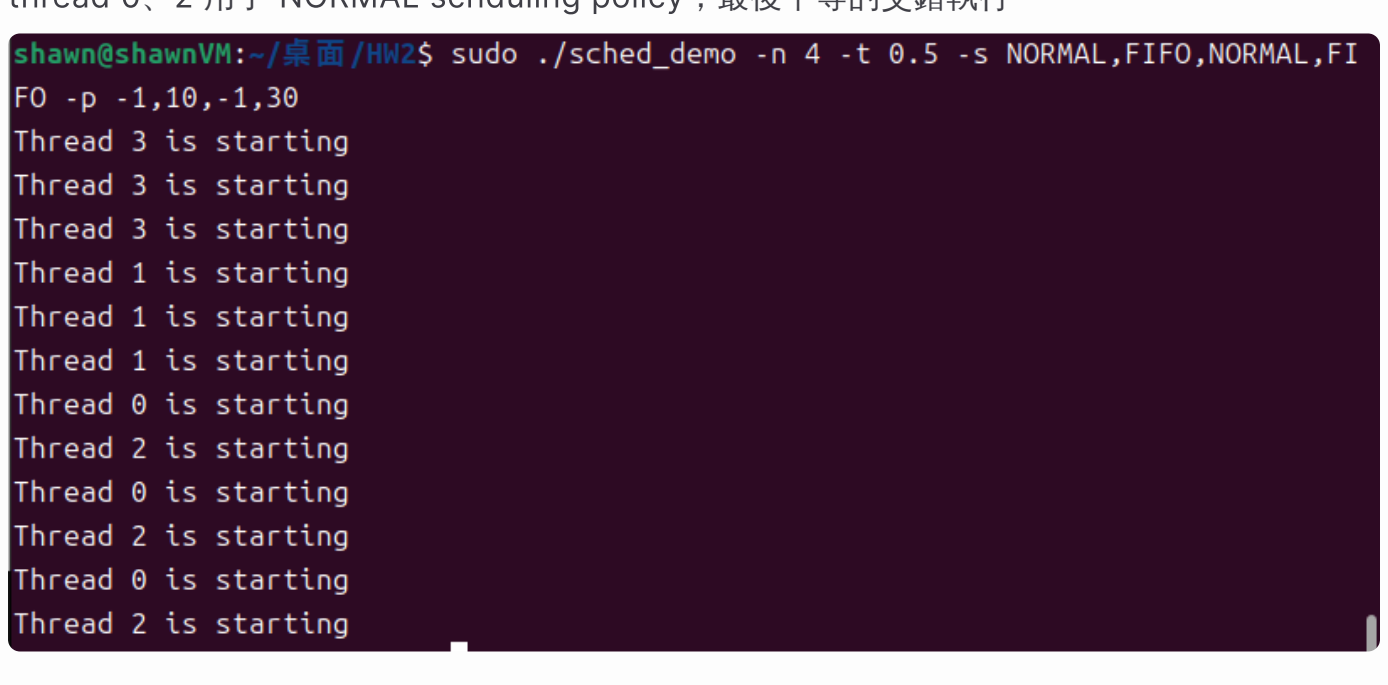
II. Describe the results of sudo ./sched_demo -n 3 -t 1.0 -s NORMAL,FIFO,FIFO -p -1,10,30 and what causes that. (10%)

- thread 1、2 都用了 FIFO schduling policy，而 thread 2 的 priority (30) 大於 thread 1 的 priority (10)，所以 thread 2 先執行
- thread 0 用了 NORMAL schduling policy，最後執行


shawn@shawnVM:~/桌面/HW2$ sudo ./sched_demo -n 3 -t 1.0 -s NORMAL,FIFO,FIFO -p -
1,10,30
Thread 2 is starting
Thread 2 is starting
Thread 2 is starting
Thread 1 is starting
Thread 1 is starting
Thread 1 is starting
Thread 0 is starting
Thread 0 is starting
Thread 0 is starting

III. Describe the results of sudo ./sched_demo -n 4 -t 0.5 -s NORMAL,FIFO,NORMAL,FIFO -p -1,10,-1,30, and what causes that. (10%)

- thread 1、3 都用了 FIFO schduling policy，而 thread 3 的 priority (30) 大於 thread 1 的 priority (10)，所以 thread 3 先執行
- thread 0、2 用了 NORMAL schduling policy，最後平等的交錯執行


shawn@shawnVM:~/桌面/HW2$ sudo ./sched_demo -n 4 -t 0.5 -s NORMAL,FIFO,NORMAL,FI
FO -p -1,10,-1,30
Thread 3 is starting
Thread 3 is starting
Thread 3 is starting
Thread 1 is starting
Thread 1 is starting
Thread 1 is starting
Thread 0 is starting
Thread 2 is starting
Thread 0 is starting
Thread 2 is starting
Thread 0 is starting
Thread 2 is starting

IV. Describe how did you implement n-second-busy-waiting? (10%)

- 用了 `<chrono>`
- 設置 start time、end time 以及 time(現在時間) 來判斷 thread 執行時間

```
auto start_time = chrono::high_resolution_clock::now();
auto end_time = start_time + chrono::milliseconds(int(thread_info->time_wait*1000))
auto time = chrono::high_resolution_clock::now();
while (time < end_time) {
    time = chrono::high_resolution_clock::now();
}
```

V. What does the kernel.sched_rt_runtime_us effect? If this setting is changed, what will happen?(10%)

- 控制 runtime task 的 time quota
- 給足夠的執行時間，讓 CPU 不會因為 timeout 被 context switch，影響程式的排程