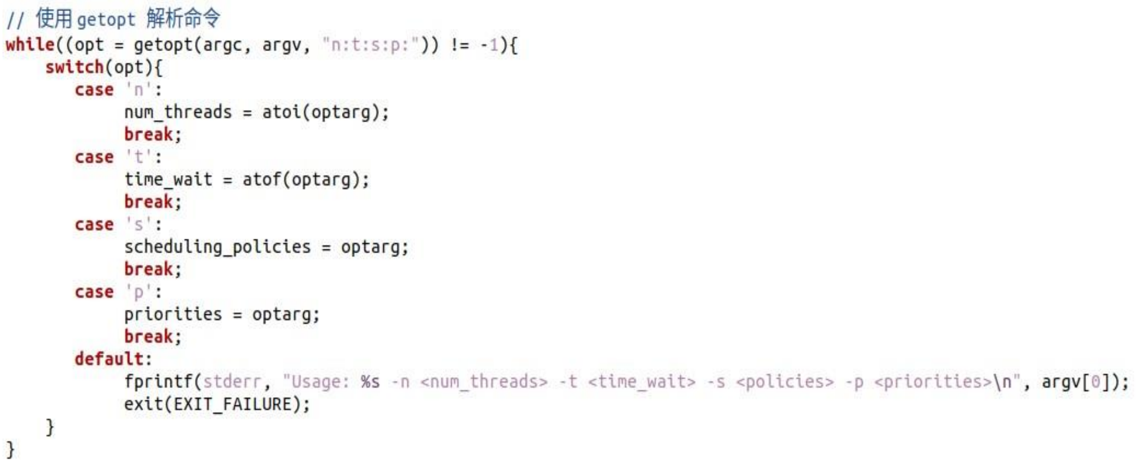
# Assignment 2: Scheduling Policy Demonstration Program

網工所碩一 林宏頤 312552047

1. Describe how you implemented the program in detail.

Step1:

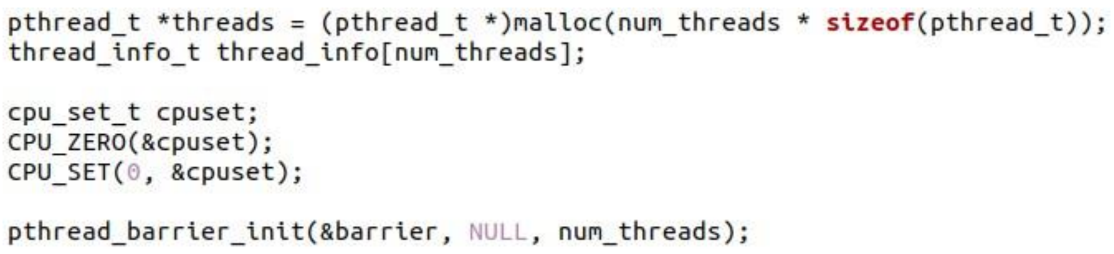
In the main function, I use getopt to parse command-line option.



```
// 使用getopt 解析命令
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':
            scheduling_policies = optarg;
            break;
        case 'p':
            priorities = optarg;
            break;
        default:
            fprintf(stderr, "Usage: %s -n <num_threads> -t <time_wait> -s <policies> -p <priorities>\n", argv[0]);
            exit(EXIT_FAILURE);
    }
}
```

Step2:

I initiate the relating data structure and parameters.



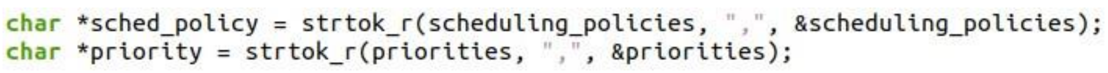
```
pthread_t *threads = (pthread_t *)malloc(num_threads * sizeof(pthread_t));
thread_info_t thread_info[num_threads];

cpu_set_t cpuset;
CPU_ZERO(&cpuset);
CPU_SET(0, &cpuset);

pthread_barrier_init(&barrier, NULL, num_threads);
```

Step3:

I use strtok_r to separate the scheduling policies and priorities.



```
char *sched_policy = strtok_r(scheduling_policies, ",", &scheduling_policies);
char *priority = strtok_r(priorities, ",", &priorities);
```

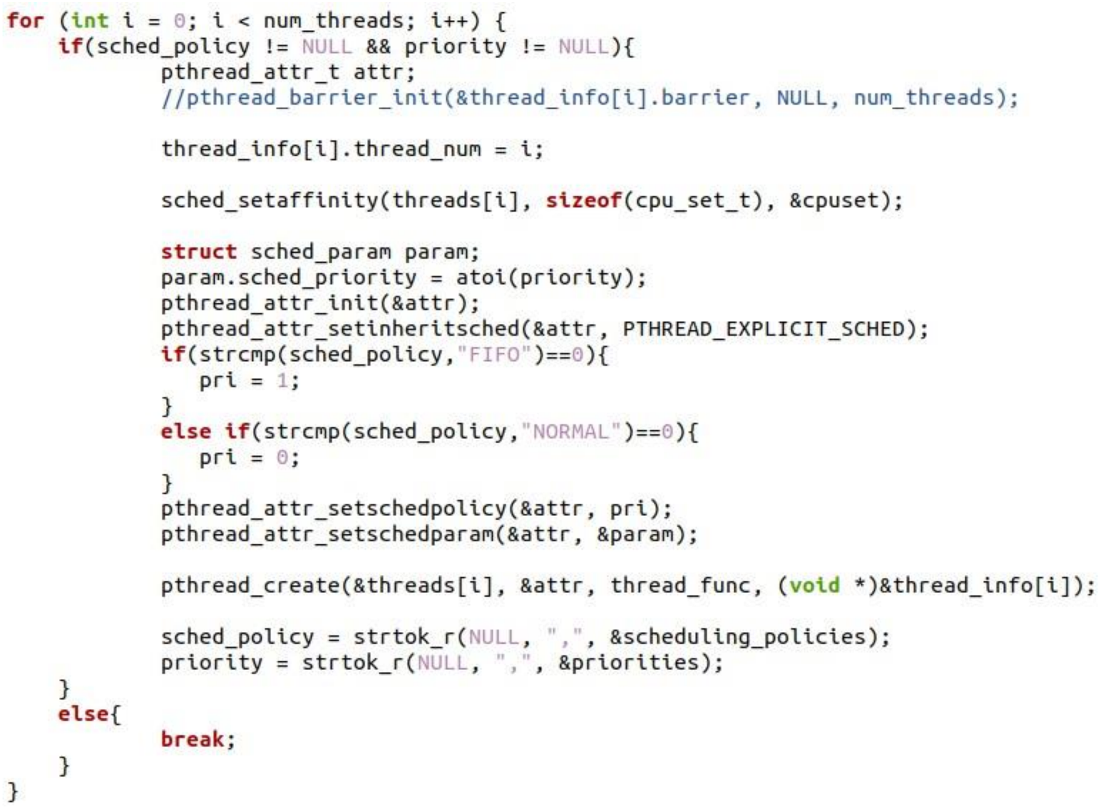
Step4:

Use sched_setaffinity to set affinity and direct them to the same CPU.

Use pthread_attr_init, pthread_attr_setinheritsched, pthread_attr_setschedpolicy, pthread_attr_setschedpolicy, pthread_attr_setschedparam to set the policy and priority for each thread.

Use pthread_create to create each thread.

Lastly, I use another strtok_r to get the next policy and priority.



```c
for (int i = 0; i < num_threads; i++) {
    if(sched_policy != NULL && priority != NULL){
            pthread_attr_t attr;
            //pthread_barrier_init(&thread_info[i].barrier, NULL, num_threads);

            thread_info[i].thread_num = i;

            sched_setaffinity(threads[i], sizeof(cpu_set_t), &cpuset);

            struct sched_param param;
            param.sched_priority = atoi(priority);
            pthread_attr_init(&attr);
            pthread_attr_setinheritsched(&attr, PTHREAD_EXPLICIT_SCHED);
            if(strcmp(sched_policy,"FIFO")==0){
               pri = 1;
            }
            else if(strcmp(sched_policy,"NORMAL")==0){
               pri = 0;
            }
            pthread_attr_setschedpolicy(&attr, pri);
            pthread_attr_setschedparam(&attr, &param);

            pthread_create(&threads[i], &attr, thread_func, (void *)&thread_info[i]);

            sched_policy = strtok_r(NULL, ",", &scheduling_policies);
            priority = strtok_r(NULL, ",", &priorities);
    }
    else{
            break;
    }
}
```

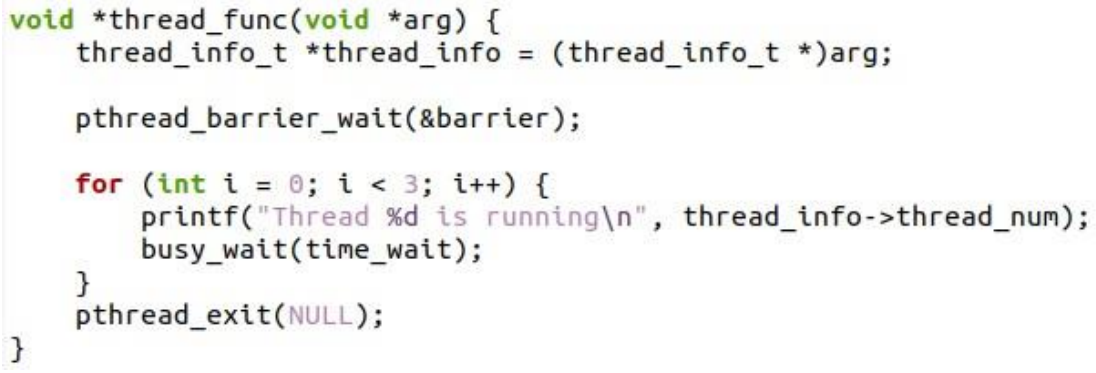
Step5:

Start the thread's function.

Each thread will first acquire their information.

Stocked by pthread_barrier_wait to wait other thread.

Every thread performs 3 times loop.

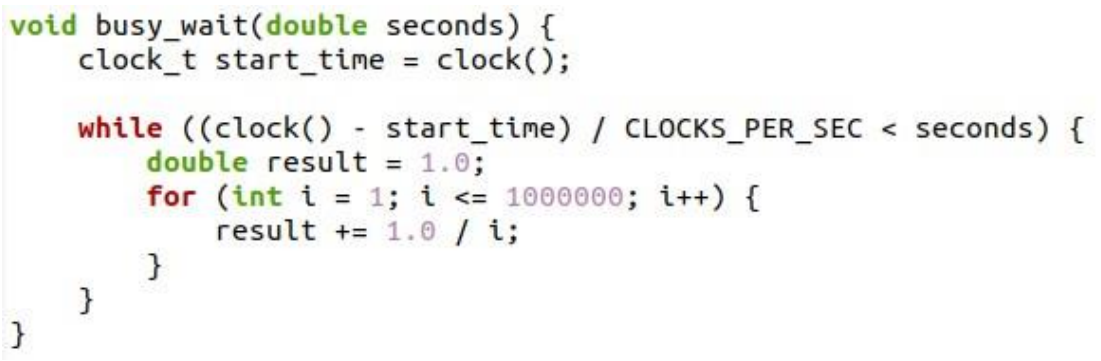


```c
void *thread_func(void *arg) {
    thread_info_t *thread_info = (thread_info_t *)arg;

    pthread_barrier_wait(&barrier);

    for (int i = 0; i < 3; i++) {
        printf("Thread %d is running\n", thread_info->thread_num);
        busy_wait(time_wait);
    }
    pthread_exit(NULL);
}
```

Busy_wait calculates the difference between start time and the current time and use a while loop to keep track of the difference to see if it exceeds the required period or not.

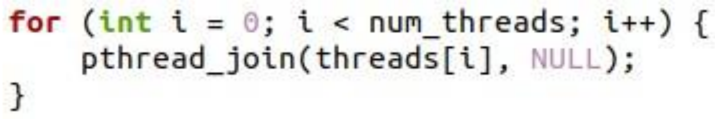


```c
void busy_wait(double seconds) {
    clock_t start_time = clock();

    while ((clock() - start_time) / CLOCKS_PER_SEC < seconds) {
        double result = 1.0;
        for (int i = 1; i <= 1000000; i++) {
            result += 1.0 / i;
        }
    }
}
```

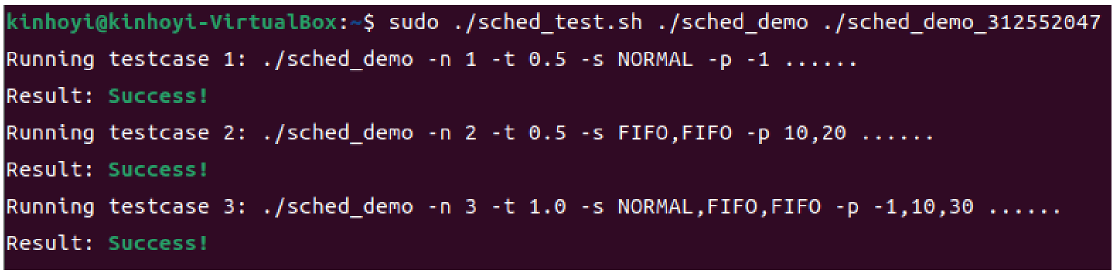
Step6:

Use pthread_join to collect the threads.



```c
for (int i = 0; i < num_threads; i++) {
    pthread_join(threads[i], NULL);
}
```

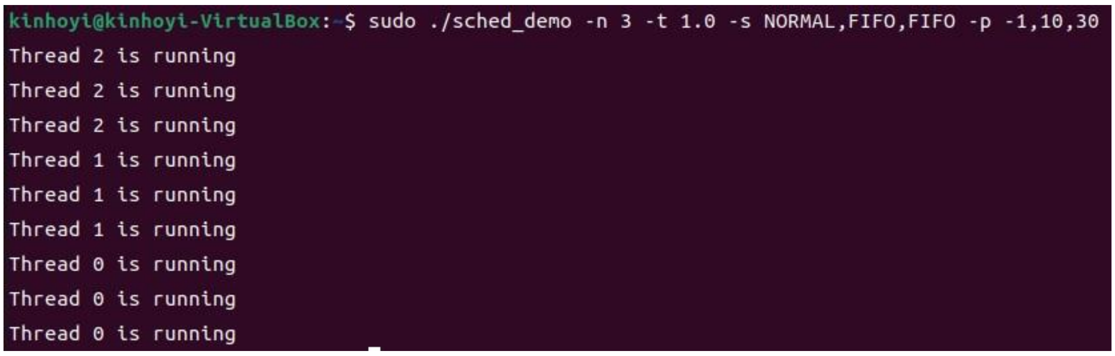
Result:



```
kinhoyi@kinhoyi-VirtualBox:~$ sudo ./sched_test.sh ./sched_demo ./sched_demo_312552047
Running testcase 1: ./sched_demo -n 1 -t 0.5 -s NORMAL -p -1 ......
Result: Success!
Running testcase 2: ./sched_demo -n 2 -t 0.5 -s FIFO,FIFO -p 10,20 ......
Result: Success!
Running testcase 3: ./sched_demo -n 3 -t 1.0 -s NORMAL,FIFO,FIFO -p -1,10,30 ......
Result: Success!
```

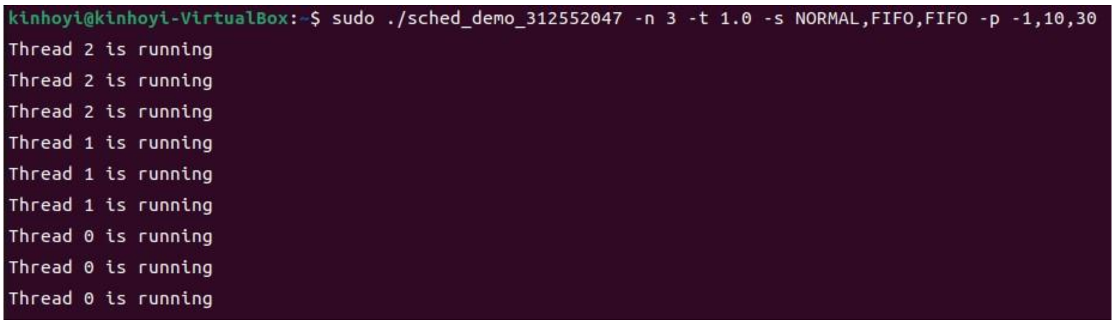
2. Describe the results of `./sched_demo -n 3 -t 1.0 -s NORMAL,FIFO,FIFO -p -1,10,30` and what causes that.

sched_demo:



```
kinhoyi@kinhoyi-VirtualBox:~$ sudo ./sched_demo -n 3 -t 1.0 -s NORMAL,FIFO,FIFO -p -1,10,30
Thread 2 is running
Thread 2 is running
Thread 2 is running
Thread 1 is running
Thread 1 is running
Thread 1 is running
Thread 0 is running
Thread 0 is running
Thread 0 is running
```

sched_demo_312552047:

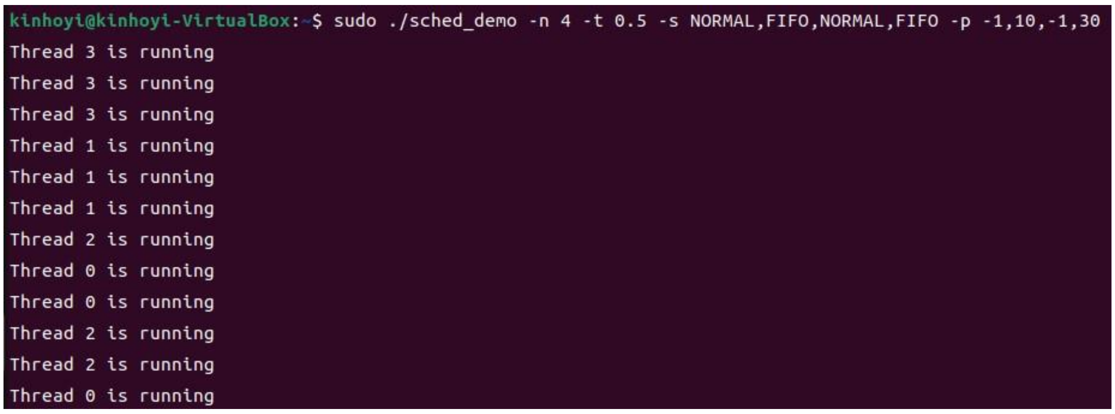


```
kinhoyi@kinhoyi-VirtualBox:~$ sudo ./sched_demo_312552047 -n 3 -t 1.0 -s NORMAL,FIFO,FIFO -p -1,10,30
Thread 2 is running
Thread 2 is running
Thread 2 is running
Thread 1 is running
Thread 1 is running
Thread 1 is running
Thread 0 is running
Thread 0 is running
Thread 0 is running
```

FIFO threads will preempt NORMAL threads, thread 1, 2 will run first.
Besides, the priority of thread 2 (30) is higher than that of thread 1 (10), so thread 2 will preempt thread 1.
Thread 2 -> Thread 1 -> Thread 0

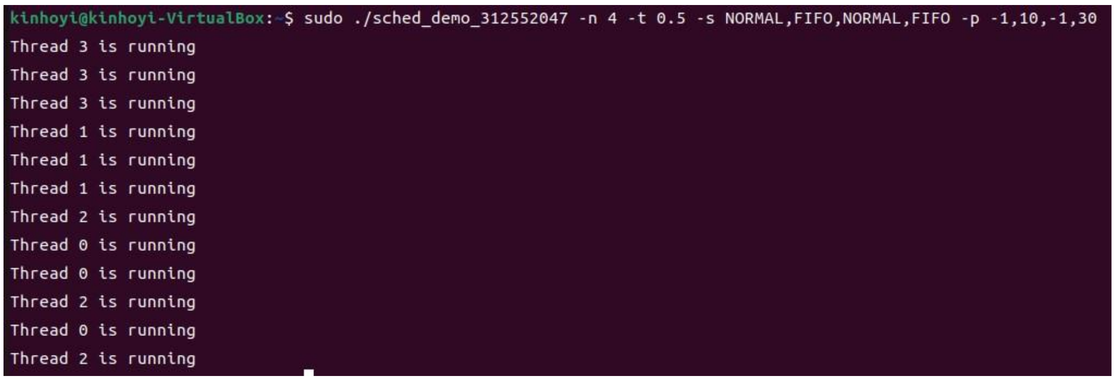
3. Describe the results of `./sched_demo -n 4 -t 0.5 -s NORMAL,FIFO,NORMAL,FIFO -p -1,10,-1,30`, and what causes that.

sched_demo:



```
kinhoyi@kinhoyi-VirtualBox:~$ sudo ./sched_demo -n 4 -t 0.5 -s NORMAL,FIFO,NORMAL,FIFO -p -1,10,-1,30
Thread 3 is running
Thread 3 is running
Thread 3 is running
Thread 1 is running
Thread 1 is running
Thread 1 is running
Thread 2 is running
Thread 0 is running
Thread 0 is running
Thread 2 is running
Thread 2 is running
Thread 0 is running
```

sched_demo_312552047:



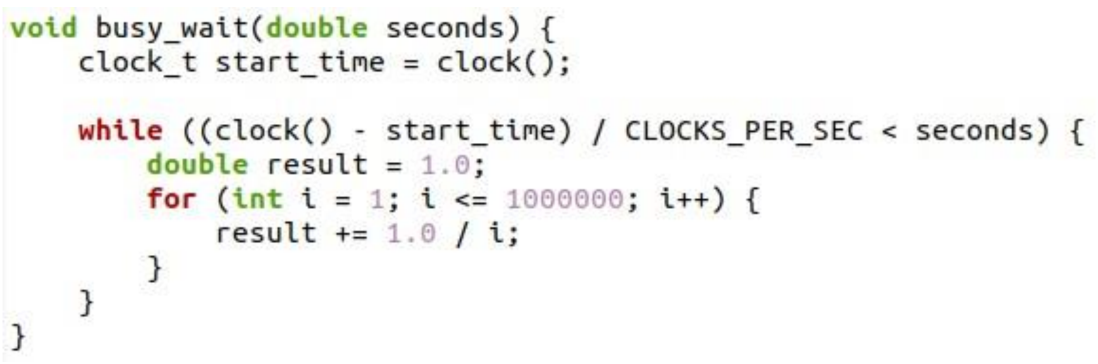
```
kinhoyi@kinhoyi-VirtualBox:~$ sudo ./sched_demo_312552047 -n 4 -t 0.5 -s NORMAL,FIFO,NORMAL,FIFO -p -1,10,-1,30
Thread 3 is running
Thread 3 is running
Thread 3 is running
Thread 1 is running
Thread 1 is running
Thread 1 is running
Thread 2 is running
Thread 0 is running
Thread 0 is running
Thread 2 is running
Thread 0 is running
Thread 2 is running
```

FIFO threads will preempt NORMAL threads, thread 1, 3 will run first.
Besides, the priority of thread 3 (30) is higher than that of thread 1 (10), so

thread 3 will preempt thread 1.
Because thread 0, 2 are both NORMAL policies, they will run in the round_robin manner to fairly in their time slice iteratively. But sometimes the order is difference. The reason is that some schedulers introduce a level of randomness or unpredictability to prevent certain patterns of behavior. This van lead to variations in the scheduling order even for threads with the same priority and policy. Besides, the scheduler will consider the overall system load and may adjust the order of thread execution accordingly.

## 4. Describe how did you implement n-second-busy-waiting?

I use start_time to record the start time, and use a while loop to check the time duration by calculating the difference between current time and start time. If the desired busy waiting time hasn't been reached yet, continue performing meaningless computation within the loop.



```
void busy_wait(double seconds) {
    clock_t start_time = clock();

    while ((clock() - start_time) / CLOCKS_PER_SEC < seconds) {
        double result = 1.0;
        for (int i = 1; i <= 1000000; i++) {
            result += 1.0 / i;
        }
    }
}
```