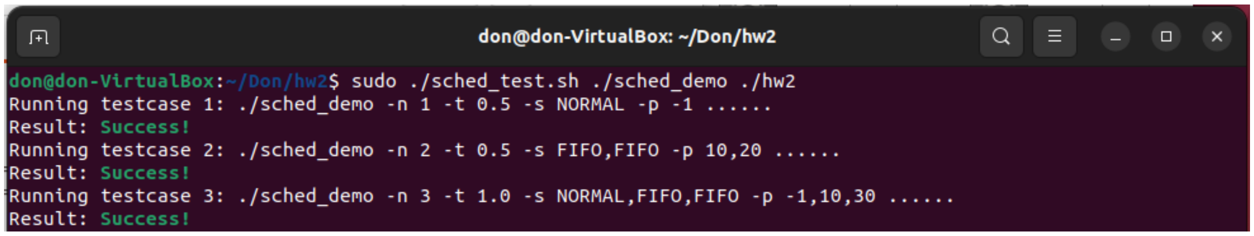
Operation System

310551003 陳嘉慶

Program Source Part

3 Test cases has been tested by provided command as below, all of them are completed successfully:



```
don@don-VirtualBox: ~/Don/hw2
don@don-VirtualBox:~/Don/hw2$ sudo ./sched_test.sh ./sched_demo ./hw2
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!
```

Report

1. Describe how you implemented the program in detail. (20%)

There are total 5 part of program implementation:

1. Parse program argument:
   It is used for store parameter input by command, *getopt* combine with *switch case* can be added the options input in command line. Except number of thread and busy time of every thread, there are 2 multi-values option input - process policy type and process priority. *plc_sep* and *pri_sep* are the function to separate multi-values input one by one to a vector.
2. Create thread:
   Threads can be created easily by *pthread_t tid* with input number of threads.
3. Set CPU affinity:
   First, create CPU mask for CPU setting. Then, zeroized currenting running CPU setting and set to corresponding CPU(e.g.: 0) with CPU mask created. Finally, set the affinity by *sched_setaffinity* command.
4. Set attribute to thread and start all thread at once:
   Through thread parameters to setup a thread by *thread_func* in *pthread_create*. In *thread_func*, setup process to *SCHED_FIFO* by *sched_setscheduler* if 'FIFO' input from command, utilize *pthread_barrier_wait* to wait for all threads complete their setup. Create 3 loops as requested and utilized *gettimeofday* to count 1 second inside the loop so that a busy process can be controlled by a variable 'time'.
5. Wait for all threads finish
   Utilize *pthread_join* to wait for all threads finished if any remaining process still processing in CPU.

Operation System

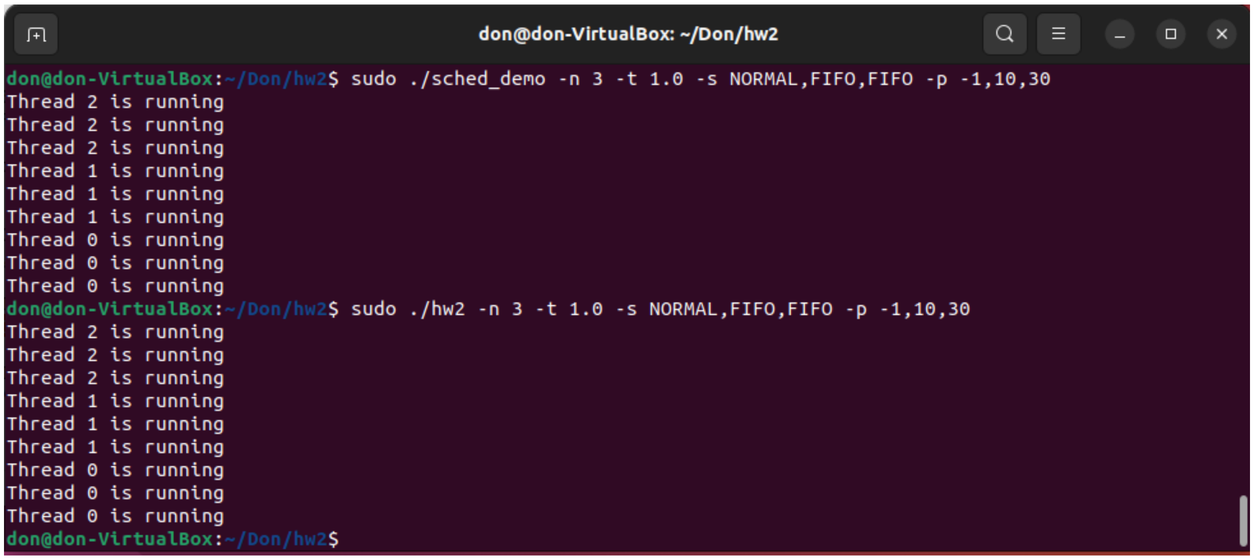
310551003 陳嘉慶

```
#include <iostream>
#include <string.h>
#include <thread>
#include <chrono>
#include <stdio.h>
#include <sched.h> // for sched_getcpu()
#include <pthread.h>
#include <unistd.h>
#include <stdlib.h>
#include <vector>
#include <mutex>
#include <assert.h>
#include <errno.h>
#include <sys/syscall.h>
#include <sys/types.h>
#include <sys/time.h>
using namespace std;

extern char* optarg;
extern int optind;
extern int opterr;
extern int optopt;

pthread_barrier_t barrier;  //create barrier variable;

typedef struct {
    int thread_id;
    double time;
    int sched_policy;
    int sched_priority;
} thread_info_t;

vector <int> plc_sep(char *plc_total, int num_thread){
    vector <int> plc_vec;
    char *plc = strtok(plc_total, ",");

    if(plc) {

            if (!strcmp(plc, "FIFO")) { //strcmp return 0 if equal
                plc_vec.push_back(1);
            }
            else{
                plc_vec.push_back(0);
        }
    }
    for (int i=1; i<num_thread; i++){
        plc = strtok(NULL, ",");
        if(plc) {
            if (!strcmp(plc, "FIFO")) { //strcmp return 0 if equal
                plc_vec.push_back(1);
            }
            else{
                plc_vec.push_back(0);
            }
        }
    }
    return plc_vec;
}


vector <int> pri_sep(char *pri_total, int num_thread){
    vector <int> pri_vec;
    char *pri = strtok(pri_total, ",");
    if(pri) {
        pri_vec.push_back(atoi(pri));
    }
    for (int i=1; i<num_thread; i++){
        pri = strtok(NULL, ",");
        if(pri) {
            pri_vec.push_back(atoi(pri));
        }
    }
    return pri_vec;
}


void* thread_func(void *arg) {  //int t,
    thread_info_t tmp = *(thread_info_t *)arg;
    int mid = tmp.thread_id;  //dereference void *arg point to int *arg
    double time = tmp.time;
    int policy = tmp.sched_policy;
    int priority = tmp.sched_priority;
    int s = 0;

    if (policy==1) {
        struct sched_param param;
        param.sched_priority = priority;
        s = sched_setscheduler(0, SCHED_FIFO, &param);
    }
    pthread_barrier_wait(&barrier);  //wait for all thread and start at once
    for (int it = 0; it < 3; it++) {
        printf("Thread %d is running\n", mid);
        // busy 1 second, time is 1second ratio
        struct timeval start;
        struct timeval end;
        unsigned long diff;
        unsigned long target=1000000*time;

        gettimeofday(&start,NULL);
        do{
            gettimeofday(&end,NULL);
            diff = 1000000 * (end.tv_sec-start.tv_sec) + end.tv_usec-
start.tv_usec;
        }
        while(diff<target);
    }
    return (void *) 0;
}

int main(int argc, char *argv[]) {  //argc: index, argv: value
    // Initial variable
    int scheduler = 0;  // default scheduler
    thread_info_t thread_para;
    int opt, num_thread;
    double time;
    char *plc_total=0, *pri_total;
    vector <int> plc_vec;
    vector <int> pri_vec;
    // Part 1: Parse program arghuments
    while ((opt = getopt(argc, argv, "n:t:s:p:")) != -1)
    {
        switch(opt)
        {
        case 'n':
        num_thread = atoi(optarg);  //convert char array to int
        break;
        case 't':
        time = atof(optarg);
        break;
        case 's':  //policy
        plc_total = optarg;
        break;
        case 'p':  //priority
        pri_total = optarg;
        break;
        case ':':
        puts("oops");
        break;
        case '?':
        puts("wrong command");
        break;
        }
    }

    //Seperate char array by each thread policy and save to vector.
    plc_vec = plc_sep(plc_total, num_thread);

    //Seperate char array by each thread priority and save to vector.
    pri_vec = pri_sep(pri_total, num_thread);

    pthread_t tid[num_thread];  // Part 2: Create thread

    //Part 3: Set CPU affinity
    cpu_set_t mask;
    CPU_ZERO(&mask);  # Remove current running CPU setting
    CPU_SET(0, &mask);  # Set to run CPU0 only
    sched_setaffinity(0, sizeof(mask), &mask);

    pthread_barrier_init(&barrier, NULL, num_thread);  //init barrier

    //Part 4: Set attribute to threads and start all threads at once
    for (int i = num_thread; i >0; i--) {
        thread_para.thread_id=i-1;
        thread_para.time=time;
        thread_para.sched_policy=plc_vec[i-1];
        thread_para.sched_priority=pri_vec[i-1];
        int err = pthread_create(&tid[i], NULL, thread_func, &thread_para);
        assert(err == 0);
        sleep(1);
    }
    //Part 5: Wait for all threads finish if any remaining
    for (int i = num_thread; i >0; i--)
        pthread_join(tid[i], NULL);
    return 0;
}
```

310551003 陳嘉慶

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

   The result of testcase 3 is following completed sequence from 2 to 0 thread. As real time scheduling policy priority have higher priority than fair scheduling when running on the CPU, *SCHED_FIFO* will be planned to run before *SCHED_NORMAL*. Moreover, priority 30 thread will be scheduled to run before 10, that is, we have priority 30 *SCHED_FIFO* thread 2 run first, then *SCHED_FIFO* with priority 10. Finally, *SCHED_NORMAL*. As *SCHED_FIFO* will not consider time slicing, it always preempts immediately, thread will start after previous high priority thread completed. Combining with *sched_rt_runtime* and busy time=1, we have following result.



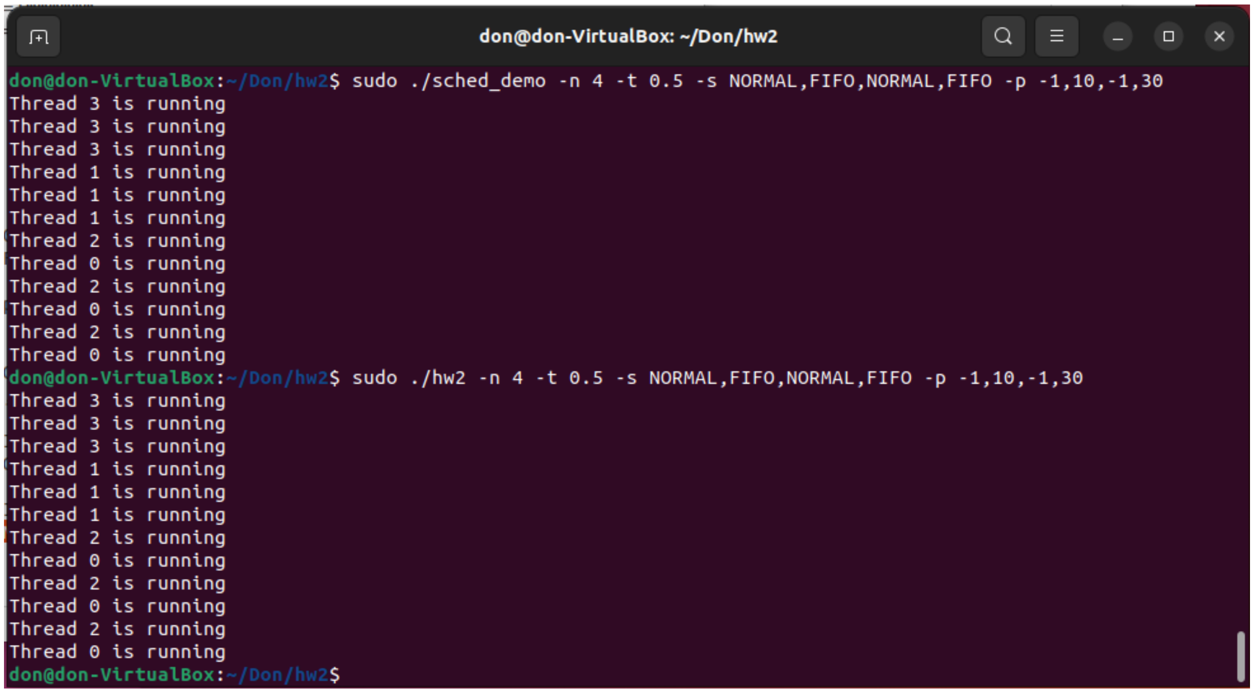
```
don@don-VirtualBox:~/Don/hw2$ sudo ./sched_demo -n 3 -t 1.0 -s NORMAL,FIFO,FIFO -p -1,10,30
Thread 2 is running
Thread 2 is running
Thread 2 is running
Thread 1 is running
Thread 1 is running
Thread 1 is running
Thread 0 is running
Thread 0 is running
Thread 0 is running
don@don-VirtualBox:~/Don/hw2$ sudo ./hw2 -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
don@don-VirtualBox:~/Don/hw2$
```

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. (10%)**

   As previous question mentioned, we now add 1 more *SCHED_NORMAL* thread and change the busy time=0.5, the running sequence will be first priority 30 thread 2, then priority 10 thread 1 as previous question. After, will be thread 2 and thread 1 as both of them are fair scheduling policy. In here, it depends on which thread you set attribute first, then it will be run first, but CPU will swap to another thread during part of current thread completed. It will be 020202 running sequence if the program set thread 0 first.

Operation System

310551003 陳嘉慶



```
don@don-VirtualBox: ~/Don/hw2
don@don-VirtualBox:~/Don/hw2$ 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 2 is running
Thread 0 is running
Thread 2 is running
Thread 0 is running
don@don-VirtualBox:~/Don/hw2$ sudo ./hw2 -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 2 is running
Thread 0 is running
Thread 2 is running
Thread 0 is running
don@don-VirtualBox:~/Don/hw2$
```

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

   gettimeofday is a function to get current time and loop it until it reaches our target time. To do this, we first store start time of process, use do while to check current time until current time reach what we set at target variable. Note that it still contain error in us.

```
struct timeval start;
        struct timeval end;
        unsigned long diff;
        unsigned long target=1000000*time;

        gettimeofday(&start,NULL);
        do{
            gettimeofday(&end,NULL);
            diff = 1000000 * (end.tv_sec-start.tv_sec) + end.tv_usec-
start.tv_usec;
        }
        while(diff<target);
```