Assignment2

1. Program

```
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. Report

Q1.

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

1. parse the argument with `getopt`

   用enum設定值，讓後續用到可以知道參數意思

   因為FIFO NORMAL 是先用string存起來，所以多寫一個function去轉成數字

   main 去切割字串並拿到所有參數照種類分別儲存，而index就是之後建立thread的idx

```
// helper function
enum val{
    NORMAL = 0,
    FIFO = 1
};

void parse_policies (vector<string>& policies_str, vector<int>& policies) {
    for (int i = 0; i < policies_str.size(); i++) {
        if (policies_str[i] == "NORMAL")  {
            policies.push_back(0);
        }
        else if (policies_str[i]== "FIFO")  {
            policies.push_back(1);
        }
        else {
            policies.push_back(-1);
        }
    }
    return;
}
```

```
// main
int     opt;
while ((opt = getopt(argc, argv, "n:t:s:p:")) != -1) {
        switch (opt) {
            case 'n':
                num_threads = stoi(optarg);
                break;
            case 't':
                time_wait = stof(optarg);
                break;
            case 's':
                {
                    char* token = strtok(optarg, ",");
                    while (token != nullptr) {
                        policies_str.push_back(token);
                        token = strtok(nullptr, ",");
                    }
                }
                break;
            case 'p':
                {
                    char* token = strtok(optarg, ",");
                    while (token != nullptr) {
                        priorities.push_back(stoi(token));
                        token = strtok(nullptr, ",");
                    }
                }
                break;
            default:
                cerr << "Invalid arguments. Usage: -n <num_threads> -t <time_wait> \
                -s <policies> -p <priorities>" << endl;
                exit(EXIT_FAILURE);
        }
    }
    if (policies_str.size() != num_threads || priorities.size() != num_threads) {
        cerr << "Invalid arguments nums. Usage: -n <num_threads> -t <time_wait> \
        -s <policies> -p <priorities>" << endl;
        exit(EXIT_FAILURE);
    }
    parse_policies(policies_str, policies);
```

2. set cpu affinity

   讓每個thread都要在CPU-0，所以直接在main function設定就好

```
int     cpu_id = 0;
cpu_set_t cpu_set;
CPU_ZERO(&cpu_set);
CPU_SET(cpu_id, &cpu_set);
sched_setaffinity(0, sizeof(cpu_set), &cpu_set);
```

3. thread 的建立與同步，將我們拿到的參數作為thread建立的資料

因為需要讓所有thread同步，所以利用function `pthread_barrier_init()` , `pthread_barrier_wait()` 去等待所有 workthread建完後再同步放其執行，避免可能會導致先創立好的workthread會先被執行的狀況

```
pthread_barrier_init(&bar, NULL, num_threads);
thread_info_t threads_info[num_threads];

for (int i = 0; i < num_threads; i++) {
    threads_info[i].thread_id = i;
    threads_info[i].sched_policy = policies[i];
    threads_info[i].sched_priority = priorities[i];
    threads_info[i].time = time_wait;

    int ret = pthread_create(&threads_id[i], NULL, thread_func, &threads_info[i]);
    assert(ret == 0);
}
```

4. `thread_function`

wait 後，設定FIFO的case讓thread可以照我們預期去排程，因為normal是default，所以可以不用特別改，do it’s task 就讓他在while loop 裡面繞到預期時間

```
void *thread_func(void *arg)
{
    thread_info_t *cur_thread = new thread_info_t;
    memcpy(cur_thread, arg, sizeof(thread_info_t));

    if (cur_thread->sched_policy == FIFO) {
        struct sched_param param;
        param.sched_priority = cur_thread->sched_priority;
        int ret = sched_setscheduler(0, SCHED_FIFO, &param);
    }
    /* 1. Wait until all threads are ready */
    pthread_barrier_wait(&bar);

    /* 2. Do the task */
    clock_t start_time , end_time;
    double run_time = 0.0;
    for (int i = 0; i < 3; i++) {
        printf("Thread %d is running\n", cur_thread->thread_id);
        /* Busy for <time_wait> seconds */
        start_time = clock();
        while (run_time < cur_thread->time) {
            end_time = clock();
            run_time = (double) (end_time - start_time) / CLOCKS_PER_SEC;
        }
    }

    /* 3. Exit the function  */
    delete cur_thread;
```

```
    pthread_exit(NULL);
}
```

Q2.

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

因為 `SCHED_FIFO` 為real-time policy，所以他會優先於 `SCHED_OTHER` ，然後 `SCHED_FIFO` 會根據Priority(1-99)的大小決定誰先執行，數字越大優先度越高

RUN: thread 2 執行三次 → thread1 執行三次→ thread0 執行三次

Q3.

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%)

所有為 `SCHED_FIFO` 的thread執行完畢，剩下來為 `SCHED_OTHER` 類型(normal)的thread會平均分配CPU資源，表現2,0,2,0,2,0交叉的狀況

RUN:

thread 3 執行三次 → thread1 執行三次

→ thread2 執行一次→ thread0 執行一次

→ thread2 執行一次→ thread0 執行一次

→ thread2 執行一次→ thread0 執行一次

Q4

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

用 `clock()` loop外開始計時，在loop裡面去計算結尾時間，與中間持續時間，直到時間大於arg的時間才能跳出while迴圈。

```
/* 2. Do the task */
clock_t start_time , end_time;
double run_time = 0.0;
```

```
    for (int i = 0; i < 3; i++) {
        printf("Thread %d is running\n", cur_thread->thread_id);
        /* Busy for <time_wait> seconds */
        start_time = clock();
        while (run_time < cur_thread->time) {
            end_time = clock();
            run_time = (double) (end_time - start_time) / CLOCKS_PER_SEC;
        }
    }
```