Operating System Assignment2:

Scheduling Policy Demonstration Program

周睦鈞 311553060

1. Describe how you implemented the program in detail

a. Parse program arguments

用 `getopt` 來處理 Command-line (Ex: `sudo ./sched_demo -n 4 -t 0.5 -s NORMAL,FIFO,NORMAL,FIFO -p -1,10,-1,30`)

我使用 `strtok` 將 `getopt` 讀取到的 schedule policy 以及 thread priority 分別存在 vector 中，方便後續創立 Thread 使用

The meaning of the command-line arguments:

- -n : Number of threads to run simultaneously
- -t : Duration of "busy" period
- -s: Scheduling policy for each thread, `SCHED_FIFO` or `SCHED_NORMAL`
- -p: Real-time thread priority for real-time threads

```
/* 1. Parse program arguments */
while ((opt = getopt(argc, argv, "n:t:s:p:")) != -1) {
    switch(opt) {
        case 'n':
            num_threads = atoi(optarg);
            break;

        case 't':
            time_slice = atof(optarg);
            break;

        case 's':
            for (char *token = strtok(optarg, ","); token != nullptr; token = strtok(nullptr, ",")) {
                if (strcmp(token, "NORMAL") == 0) {
                    schedule_policy.push_back(SCHED_OTHER);
                }
                else if (strcmp(token, "FIFO") == 0){
                    schedule_policy.push_back(SCHED_FIFO);
                }
            }
            break;
        case 'p':
            for (char *token = strtok(optarg, ","); token != nullptr; token = strtok(nullptr, ",")) {
                priority.push_back(atoi(token));
            }
            break;
        default:
            cerr << "Usage: " << argv[0] << " -n <num_threads> -t <time_slice> -s <policies> -p <priorities>\n";
            return 1;
    }
}
```

b. Create <num_threads> worker threads

使用 `pthread_t` 創立使用者輸入的 Thread 數量 (`num_threads`)，並且透過 `Arg` 來設定這些 thread 的資訊，例如 Schedule policy、Priory、Time slice 等

```
/* 2. Create <num_threads> worker threads */
pthread_t threads[num_threads];

/* 5. Start all threads at once */
pthread_barrier_init(&print_barrier, NULL, num_threads);

Arg args[num_threads];
for (int i = 0; i < num_threads; i++) {
    args[i].thread_id = i;
    args[i].sched_policy = schedule_policy[i];
    args[i].sched_priority = priority[i];
    args[i].sched_time_slice = time_slice;
}
```

```
typedef struct {
    int thread_id;
    int sched_policy;
    int sched_priority;
    float sched_time_slice;
} Arg;
```

c. Set CPU affinity and the attributes to each thread

1. 使用 `CPU_ZERO` 、`CPU_SET` 設定 CPU affinity 並且將所有 Thread 都指定到同一個 CPU ，以下我將所有 Thread 都指派給第 0 號 CPU
2. 用 `pthread_attr_t` 建立每個 Thread 的 attribute ，之後透過 `pthread_attr_setaffinity_np` 、`pthread_attr_setschedpolicy` 、`pthread_attr_setschedparam` 來分別設定該 Thread 要指定使用的 CPU 、Schedule policy 、Priority
3. 設定好每個 Thread 的 attribute 就可以用 `pthread_create` 來創立該 Thread
4. 創立好 Thread 後可以用 `pthread_attr_destroy` 將 attribute 回收

```
/* 3. Set CPU affinity */
cpu_set_t cpu_set;
// Initial CPU Set and dedicate one CPU to a particular thread
CPU_ZERO(&cpu_set);
CPU_SET(0, &cpu_set);

for (int i = 0; i < num_threads; i++) {
    struct sched_param param;
    param.sched_priority = args[i].sched_priority;

    /* 4. Set the attributes to each thread */
    pthread_attr_t attr;
    pthread_attr_init(&attr);
    pthread_attr_setinheritsched(&attr,PTHREAD_EXPLICIT_SCHED);
    pthread_attr_setaffinity_np(&attr, sizeof(cpu_set_t), &cpu_set);
    pthread_attr_setschedpolicy(&attr, args[i].sched_policy);
    pthread_attr_setschedparam(&attr, &param);

    // Create thread and set its corresponding arg and attr
    pthread_create(&threads[i], &attr, print_thread, (void *) &args[i]);

    pthread_attr_destroy(&attr);
}
```

d. Start all threads at once

為了確定所有 Thread 都會在同一個時間點開始起跑，不會因為先創立的 Thread 先跑，可以使用 barrier 來達成，下方分別為宣告、初始化、回收 barrier 的表示方法

```
pthread_barrier_t print_barrier;
pthread_barrier_init(&print_barrier, NULL, num_threads);
pthread_barrier_destroy(&print_barrier);
```

當宣告以及初始化 barrier 後， 在 Thread function 可以使用 `pthread_barrier_wait` 來設定同個起跑點

```
void *print_thread(void *args)
{
    /* 1. Wait until all threads are ready */
    Arg *arg = (Arg *) args;
    int thread_id = arg -> thread_id;
    float time_slice = arg -> sched_time_slice;

    pthread_barrier_wait(&print_barrier);

    /* 2. Do the task */
    for (int i = 0; i < 3; i++) {
        printf("Thread %d is running\n", thread_id);

        time_t start = time(NULL);
        /* Busy for <time_wait> seconds */
        while (1) {
            if ((time(NULL) - start) > time_slice)
                break;
        }

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

e. Wait for all threads to finish

這邊使用 `pthread_join` 來等待所有 Thread 完成才會繼續後續的程式，這樣的好處是避免當某一個 Thread 提前完成後就結束程式進而影響到其他 Thread

```
/* 6. Wait for all threads to finish  */
for (int i = 0; i < num_threads; i++) {
    pthread_join(threads[i], NULL);
}
```

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

從 command-line 可以得知會創立 3 個 Thread

- Thread 0 : normal policy
- Thread 1 : real-time FIFO policy , priority: 30
- Thread 2 : real-time FIFO policy , priority: 10

根據 Real time process 的 priority 會高於 Normal Process ，因此理論上 Thread 順序為 **Thread 2 → Thread 1 → Thread 0** ，但是因為根據 CFS 的原因會平均分配 Thread 使用的時間，因此部分 Thread 0 會被插入在 Real-time process 之間

```
michael@OperatingSystem:~/Desktop/HW2$ sudo ./sched_demo -n 3 -t 1.0 -s NORMAL,F
IFO,FIFO -p -1,10,30
Thread 2 is running
Thread 2 is running
Thread 0 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
```

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

從 command-line 可以得知會創立 4 個 Thread

- Thread 0 : normal policy
- Thread 1 : real-time FIFO policy , priority: 10
- Thread 2 : normal policy
- Thread 3 : real-time FIFO policy , priority: 30

同上的原因，我們可以知道理論上 Thread 的順序為 **Thread 3 → Thread 1 → Thread 0** 、**Thread 2** ，但是因為 CFS 的原因，部分 Thread 0 和 Thread 2 會被插到 Real-time process 之間

```
michael@OperatingSystem:~/Desktop/HW2$ sudo ./sched_demo -n 4 -t 0.5 -s NORMAL,F
IFO,NORMAL,FIFO -p -1,10,-1,30
Thread 3 is running
Thread 3 is running
Thread 3 is running
Thread 1 is running
Thread 2 is running
Thread 0 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
```

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

由於使用 `sleep` 會將 process 放到 sleeping state，和這次作業想實作的 busy-waiting 概念不同

因此我使用 timer 來實作 busy-waiting，每次 loop iteration 都要 busy for <time_wait> seconds

```
void *print_thread(void *args)
{
    /* 1. Wait until all threads are ready */
    Arg *arg = (Arg *) args;
    int thread_id = arg -> thread_id;
    float time_slice = arg -> sched_time_slice;

    pthread_barrier_wait(&print_barrier);

    /* 2. Do the task */
    for (int i = 0; i < 3; i++) {
        printf("Thread %d is running\n", thread_id);

        time_t start = time(NULL);
        /* Busy for <time_wait> seconds */
        while (1) {
            if ((time(NULL) - start) > time_slice)
                break;
        }

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