OS Project-2

Chi-Chia Huang 311512039

1. **First part**

- **Parse command-line**

  Using function `getopt()` to parse the command-line，利用"-x argument"的形式字符後面跟隨一個選項參數，optarg會指向選項參數，可以先將各選項參數儲存到陣列當中。

  此外`char *strtok(char *str, const char *delim)`，delim為分割的符號，可將字串切開。

```
int num_threads=atoi(argv[2]);
int thread_id[num_threads];
char *policies[num_threads];
int priorities[num_threads];
char *d=",";
char *s_p;
char *p_p;
int busy_time;
int main(int argc,char *argv[]){
  while((ch=getopt(argc,argv,"n:t:s:p:")) !=-1){
        switch(ch){
            case 'n':
                for(int i=0;i<num_threads;i++){
                    thread_id[i]=i;
                }
                break;
            case 't':
                busy_time=atoi(optarg);
                break;
            case 's':
                char *s_string=optarg;
                s_p=strtok(s_string,d);
                for(int i=0;i<num_threads;i++){
                    policies[i]=s_p;
                    s_p=strtok(NULL,d);
                }
                break;
            case 'p':
                char *p_string=optarg;
                p_p=strtok(p_string,d);
                for(int i=0;i<num_threads;i++){
                    priorities[i]=atoi(p_p);
                    p_p=strtok(NULL,d);
                }
                break;
                  }
    }
}
```

- **Set cpu affinity**

將thread都設定在CPU-0，如果不這樣做在thread排程上可能不會按照給定的優先權排程，可參考以下網站

C/C++ Linux/Unix 讓執行緒跑在指定 CPU 的方法 sched_setaffinity

本篇 ShengYu 介紹 C/C++ Linux/Unix 執行緒設定 CPU 的方法 sched_setaffinity()， 主執行緒要設定跑在哪顆 CPU 的話，可以直接在 main 裡的主執行緒使用 sched_setaffinity() 設定即可， sched_setaffinity() 的第一個參數為 pid，以我的電腦來說是單 CPU 4 核心，所以有 CPU0~CPU3 可以選擇，這邊示範選擇跑在 CPU3， 也可以使用 pthread_setaffinity_np() 來設定 main 主

https://shengyu7697.github.io/cpp-sched_setaffinity/

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

- **Set the attributes to each thread**

  先初始化thread參數，並且選擇繼承方式，接著設定排程Policy以及Priority(如果是real-time scheduling police才需要Priority)，在創建work-thread的時候一併傳入。

```
pthread_attr_t attr[num_threads];
struct sched_param param[num_threads];

pthread_attr_init(&attr[i]);
pthread_attr_setinheritsched(&attr[i],PTHREAD_EXPLICIT_SCHED);
pthread_attr_setschedpolicy(&attr[i],policy_int);
param[i].sched_priority=priorities[i];
pthread_attr_setschedparam(&attr[i],&param[i]);
pthread_create(&thread[i],&attr[i],thread_func,&thread_information[i]);
```

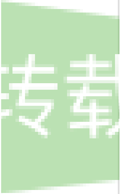
- **Start all threads at once**

  利用function `pthread_barrier_init()` , `pthread_barrier_wait()` , `pthread_barrier_destroy()` ,去等待所有work-thread全布建完成後再同步放其執行，如果沒有這樣做的話，可能會導致先創立好的work-thread會先被執行，這樣就不會按到預先設定的排程輸出，可參考以下網站

pthread_barrier_init,pthread_barrier_wait简介_qq405180763的博客-CSDN博客_pthread_barrier_init

pthread_barrier 系列函数在 中定义，用于多线程的同步，它包含三个函数： --pthread_barrier_init() --pthread_barrier_wait() --pthread_barrier_destroy() 那么pthread_barrier_*是用来做什么的？这三个函数又怎么配合使用呢？ ...

https://blog.csdn.net/qq405180763/article/details/23919191



- **Wait for all threads to finish**

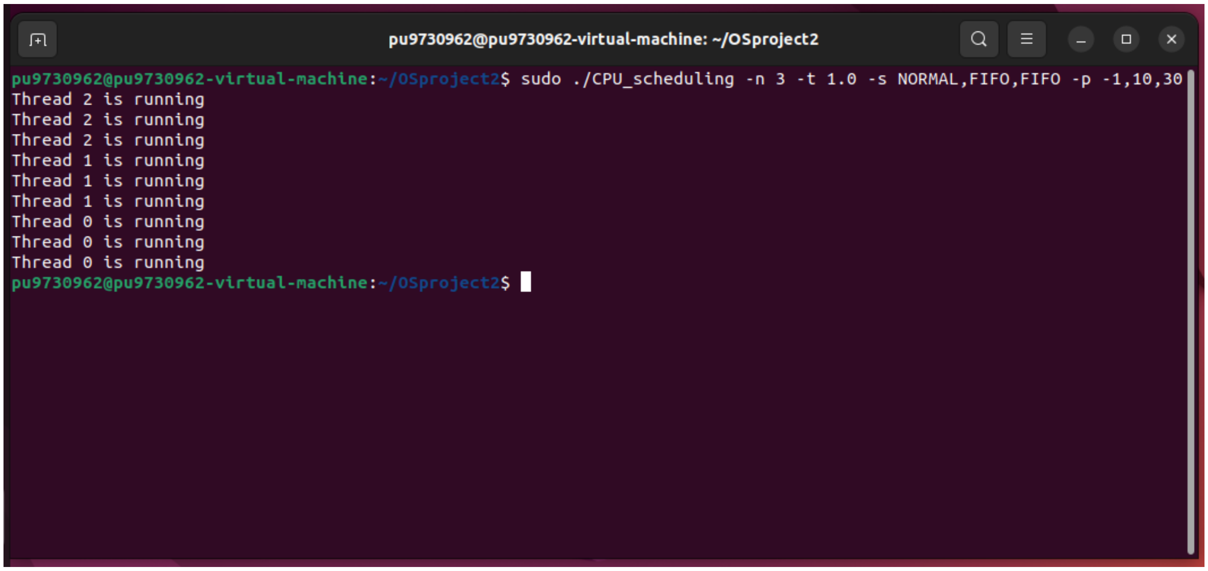
利用function pthread_join(pthread_t thread, void **retval)，main-thread會等到所有work-thread都執行完畢才接著繼續執行，可參考以下網

2. **Second part**

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

`SCHED_FIFO`為real-time policy，所以他會優先於`SCHED_OTHER`，然後`SCHED_FIFO`會依照Priority(1-99)去決定誰先執行，數字越大優先度越高，所以輸出結果會如圖示

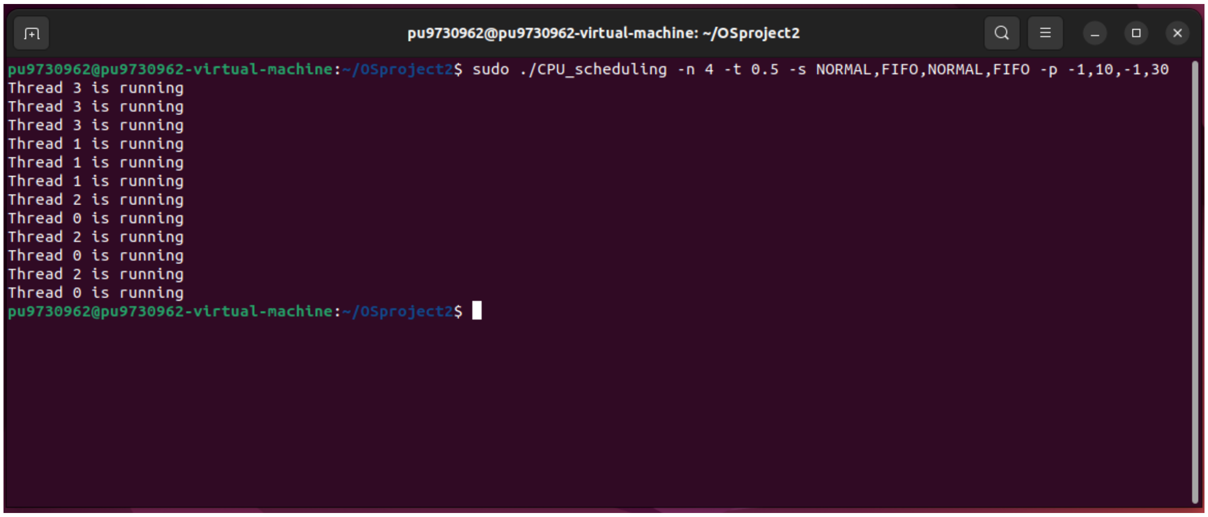
> 💡 sched_rt_runtime_us=950000(μs)是預設real-time thread 執行時間，必須先設定sched_rt_runtime_us=1000000(μs)，然而sched_rt_period_us=1000000為CPU調度排程的週期，如果不將sched_rt_runtime_us設定為1000000(μs)，會導致normal thread在real-time thread執行完後先搶佔CPU



```
pu9730962@pu9730962-virtual-machine:~/OSproject2$ sudo ./CPU_scheduling -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
pu9730962@pu9730962-virtual-machine:~/OSproject2$ 
```

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：

當所有為`SCHED_FIFO` 的thread執行完畢，剩下來為`SCHED_OTHER` 類型的thread會平均分配CPU資源，所以會呈現2,0,2,0,2,0交叉的輸出結果



```
pu9730962@pu9730962-virtual-machine:~/OSproject2$ sudo ./CPU_scheduling -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
pu9730962@pu9730962-virtual-machine:~/OSproject2$
```

3. **Third part**

Describe how did you implement n-second-busy-waiting?

利用 `clock()` 先設定一個初始時間，直到時間大於設定的busy_wait的時間才能跳出while迴圈，再執行下一次的printf

💡 `CLOCKS_PER_SEC`表示一秒鐘內CPU運行的時鐘週期數，單位為1000/sec，所以除上`CLOCKS_PER_SEC`就可以獲得秒數

```
for (int i = 0; i < 3; i++) {
        start_time=clock();
        printf("Thread %d is running\n", thread_num);
        /* Busy for <time_wait> seconds */
        while(1){
            end_time=clock();
            if(((double)(end_time-start_time)/CLOCKS_PER_SEC)>busy_wait){
                break;
            }
        }
        sched_yield();
    }
```