OS Lab2

student ID	name
312551129	蔡政邦

tags : `OS`

1.

main thread

step1

我先做parse，policy 跟 priority 我先儲存成一個string，等等會再分解

```
for(int i=1 ; i<argc ; i+=2){
    if(strcmp(argv[i],"-n")==0){
        thread_nums = atoi(argv[i+1]);
    }else if(strcmp(argv[i],"-t")==0){
        char *eptr;
        time_wait = strtod(argv[i+1],&eptr);
    }else if(strcmp(argv[i],"-s")==0){
        policy = argv[i+1];
    }else if(strcmp(argv[i],"-p")==0){
        priority = argv[i+1];
    }
}
```

宣告policys為二維陣列，儲存各個thread的policy為FIFO or Normal
宣告prioritys為一維陣列，儲存各個thread的priority
用strtok去parse剛剛儲存的string

```
int count = 0;
char policys[thread_nums][32];
int prioritys[thread_nums];

char *token = strtok(policy,",");

while(token!=NULL){
    strcpy(policys[count++],token);
```

```
        token = strtok(NULL,",");
    }

    count=0;
    token = strtok(priority,",");
    while(token!=NULL){
        prioritys[count++] = atoi(token);
        token = strtok(NULL,",");
    }
```

step2

宣告各個thread以及需要的info和attribute

```
    pthread_t t[thread_nums];
    thread_info_t info[thread_nums];
    pthread_attr_t attr[thread_nums]; //attr
    cpu_set_t cpuset; //affinity
    struct sched_param param[thread_nums]; //priority
```

step3

set cpu affinity，宣告使用第0個cpu

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

step4

init barrier

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

set info to each thead,然後判斷sched_policy是SCHED_OTHER or SCHED_FIFO

```
    info[i].thread_id = (pthread_t)i;
    info[i].thread_num = thread_nums;
    info[i].sched_policy = (strcmp(policys[i],"NORMAL")==0)? SCHED_OTHER : SCHED_FIFO;
    info[i].sched_priority = prioritys[i];
```

init attribute, and set affinity to thread

```
pthread_attr_init(&attr[i]);
if(pthread_attr_setaffinity_np(&attr[i],sizeof(cpu_set_t),&cpuset)!=0){
    printf("affinity error\\n");
}
```

set policy and priority,if policy is NORMAL,do not give priority to it

```
if(pthread_attr_setschedpolicy(&attr[i],info[i].sched_policy)){
    printf("policy error\\n");
}

if(info[i].sched_policy==SCHED_FIFO){
    param[i].sched_priority = prioritys[i];
    if(pthread_attr_setschedparam(&attr[i],&param[i])!=0){
        printf("param error\\n");
    }
}
```

不要讓thread繼承main thread的性質，所以要下額外的指令，宣告完後create thread

```
if(pthread_attr_setinheritsched(&attr[i],PTHREAD_EXPLICIT_SCHED)!=0){
    printf("inherit error \\n");
    return -1;
}
pthread_create(&t[i],&attr[i],thread_func,(void *)&info[i]);
```

step5

等待所有的thread做完，且destroy掉barrier

```
for(int i=0 ; i<thread_nums ; i++){
    pthread_join(t[i],NULL);
}
pthread_barrier_destroy(&barrier);
return 0;
```

Thread function

step1

設置barrier，等待所有的thread create完再執行

```
pthread_barrier_wait(&barrier);
```

step2

thread do the task

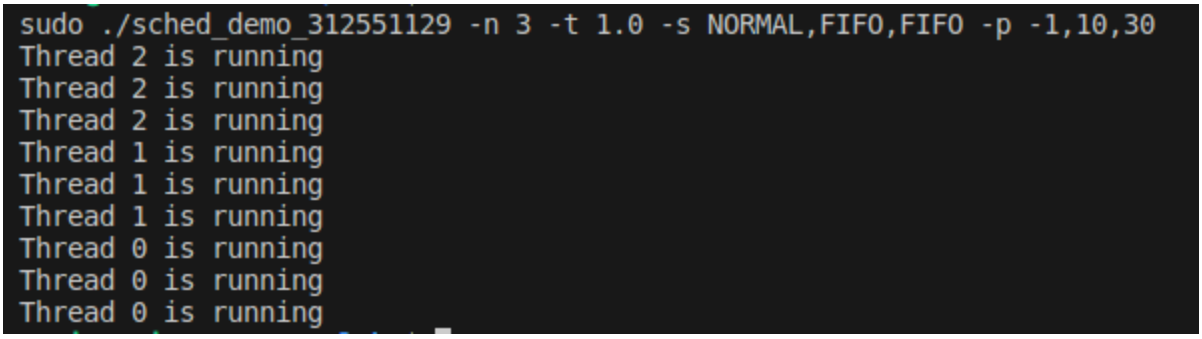
```
start = clock();
while(1){
    current = clock();
    double elapse = ((double)(current - start)) / CLOCKS_PER_SEC;
    if(elapse >= time_wait){
        break;
    }
}
```

step3

跑完則exit

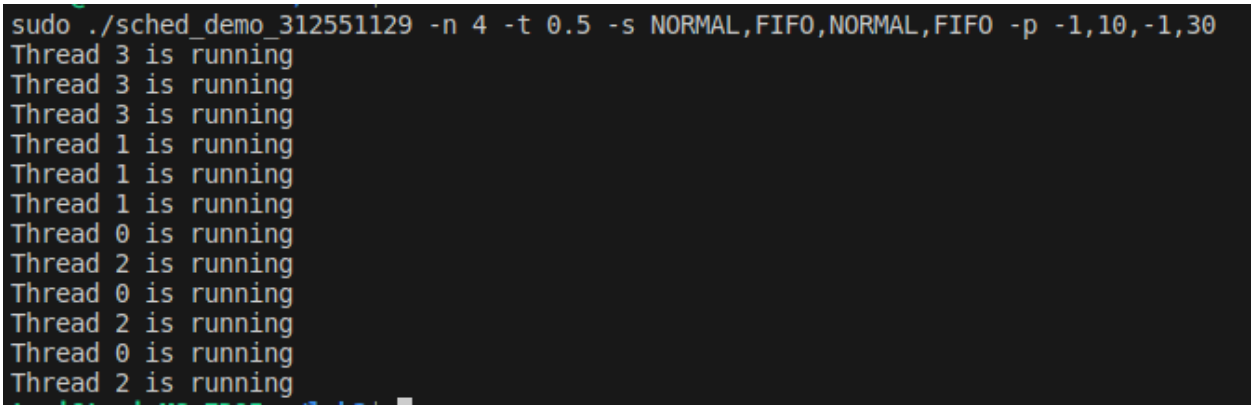
```
thread_exit(NULL);
```

2



因為FIFO是real time，所以thread 1 2會先做，而thread 2的priority為30，比thread 1的priority 10還大，所以會比它更早做。

3



因為thread1,3都是FIFO所以執行順序會比較快，且thread3的priority比thread1的priority高，所以會先做thread3再來thread1，而thread0跟2都是NORMAL，所以會輪流做，因此會輪流輸出。

4

先用start紀錄現在的時間，用一個無限迴圈去跑，每次紀錄current time，如果start-current>time_wait，則中止迴圈，這樣就不會讓它context switch。

```
start = clock();
while(1){
    current = clock();
    double elapse = ((double)(current - start)) / CLOCKS_PER_SEC;
    if(elapse >= time_wait){
        break;
    }
}
```