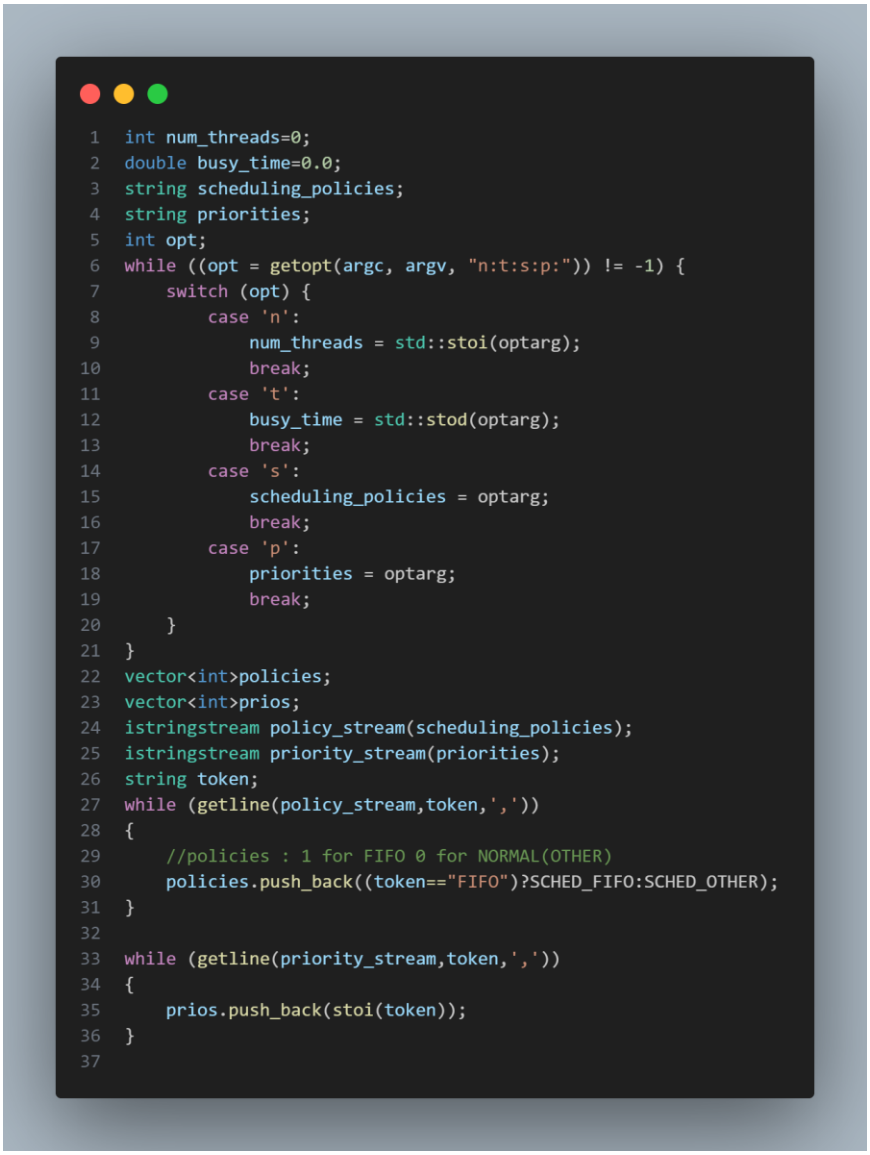
OS-Assignment2

多工所 313553052 李東諺

1. Detail of program

a. Main function



```cpp
int num_threads=0;
double busy_time=0.0;
string scheduling_policies;
string priorities;
int opt;
while ((opt = getopt(argc, argv, "n:t:s:p:")) != -1) {
    switch (opt) {
        case 'n':
            num_threads = std::stoi(optarg);
            break;
        case 't':
            busy_time = std::stod(optarg);
            break;
        case 's':
            scheduling_policies = optarg;
            break;
        case 'p':
            priorities = optarg;
            break;
    }
}
vector<int>policies;
vector<int>prios;
istringstream policy_stream(scheduling_policies);
istringstream priority_stream(priorities);
string token;
while (getline(policy_stream,token,','))
{
    //policies : 1 for FIFO 0 for NORMAL(OTHER)
    policies.push_back((token=="FIFO")?SCHED_FIFO:SCHED_OTHER);
}

while (getline(priority_stream,token,','))
{
    prios.push_back(stoi(token));
}

```

First of all, get the input argument using getopt() and switch case, do string spilt to the argument

```
pthread_barrier_init(&barrier, nullptr, num_threads);

vector<pthread_t> threads(num_threads);
vector<threadArgs> thread_args(num_threads);

cpu_set_t cpuset;
CPU_ZERO(&cpuset);
CPU_SET(0,&cpuset); //add cpu0 to cpuset
sched_setaffinity(0,sizeof(cpu_set_t),&cpuset); // set the all process to execute on cpu0
```

Initial the barrier for sync the threads, the barrier will wait for {num_threads} threads, and set the process's affinity to cpu0 so that all threads created by this process only allowed executing in cpu0

```
for (int i=0;i<num_threads;++i)
{

    pthread_attr_t attr;
    pthread_attr_init(&attr);
    pthread_attr_setschedpolicy(&attr, policies[i]);

    if (policies[i]== SCHED_FIFO)
    {                                       //struct sched_param{ int sched_priority};
    sched_param param{};
    param.sched_priority = prios[i];
    pthread_attr_setinheritsched(&attr, PTHREAD_EXPLICIT_SCHED); // set the created thread to not inherit the main thread
    pthread_attr_setschedparam(&attr,&param);
    }

    thread_args[i]= {i,policies[i],prios[i],busy_time};

    if(pthread_create(&threads[i],&attr,thread_function, &thread_args[i])!=0)
    {
        perror("pthread create fail");
        exit(1);
    }
    pthread_attr_destroy(&attr);
}

//waiting for all threads completed
for (auto &thread : threads)
{
    pthread_join(thread,nullptr);
}

pthread_barrier_destroy(&barrier);
```

For each threads, set the policy and priority to them, notice that we should do pthread_attr_setinheritsched(&attr, PTHREAD_EXPLICIT_SCHED); to the FIFO thread, Otherwise, it will inherit the attribute of main thread (NORMAL)

b. Thread function

```c
void *thread_function(void *arg)
{
    threadArgs *thread=(threadArgs *)arg;

    pthread_barrier_wait(&barrier);

    for (int i=0;i<3;i++)
    {
        timespec starttime, currenttime;
        clock_gettime(CLOCK_THREAD_CPUTIME_ID, &starttime);

        printf("Thread %d is starting\n", thread->thread_id);
        long long start=starttime.tv_sec*1e9+starttime.tv_nsec; //ns level
        while(1)
        {
            clock_gettime(CLOCK_THREAD_CPUTIME_ID,&currenttime);
            //calculate the time diff
            long long current=currenttime.tv_sec*1e9+currenttime.tv_nsec;
            long long end=thread->busy_time*1e9;
            if(current-start>=end)
                break;
        }
        sched_yield();
    }
    pthread_exit(NULL);
}
```

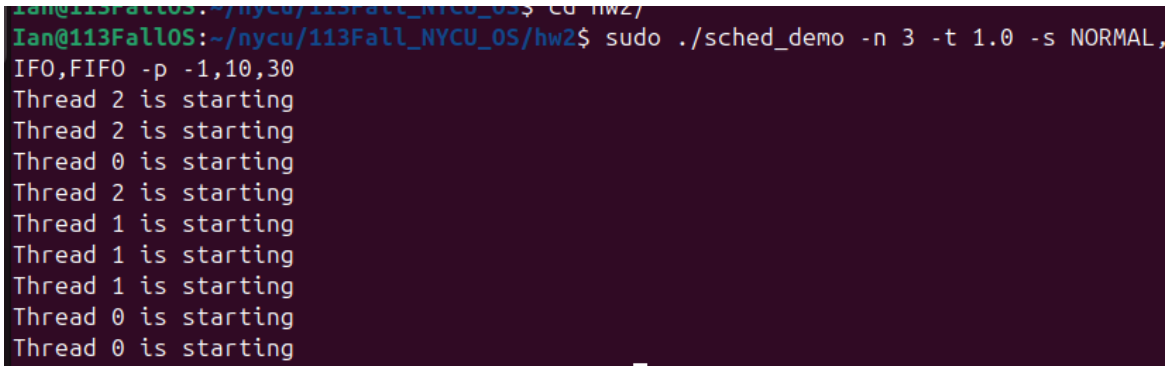
First, get the argument passed in main function, and wait for all threads

In the loop, get the cpu time (each threads are different) as starttime, transfrom the struct timespec to ns level time, in the while loop, keep calculating current time andexecute time, until it exceed the busy-waiting time.

```c
struct threadArgs
{
    int thread_id;
    int policy;
    int priority;
    double busy_time;
};
```

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

Result



At the beginning, Thread 2 is executed first because it has the highest priority. However, since sched_rt_runtime_us is set to 950000 and sched_rt_period_us is set to 1000000, Thread 2 (FIFO thread) is preempted by Thread 0 (NORMAL thread) after running for 0.95 seconds. Thread 0 then runs for 0.05 seconds and prints out Thread 0 is starting, but since it hasn't finished its 1-second busy wait, it doesn't enter the next loop.

Later, when a new scheduling period begins, Thread 2 preempts Thread 0 and resumes execution. Once Thread 2 finishes its busy wait, it yields the CPU, allowing Thread 1 to execute since it has the next highest priority. During this time, Thread 1 is also preempted by Thread 0 due to the runtime limit, but since Thread 0 hasn't completed its busy wait condition of 1 second, it doesn't print anything in the output.

3. Describe the results of sudo ./sched_demo -n 4 -t 0.5 -s NORMAL,FIFO,NORMAL,FIFO -p -1,10,-1,30, and what causes that.

Result

```
Ian@113FallOS:~/nycu/113Fall_NYCU_OS/hw2$ sudo ./sched_demo -n 4 -t 0.5 -s NORMAL,F
IFO,NORMAL,FIFO -p -1,10,-1,30
Thread 3 is starting
Thread 3 is starting
Thread 3 is starting
Thread 0 is starting
Thread 2 is starting
Thread 1 is starting
Thread 1 is starting
Thread 1 is starting
Thread 0 is starting
Thread 2 is starting
Thread 0 is starting
Thread 2 is starting
```

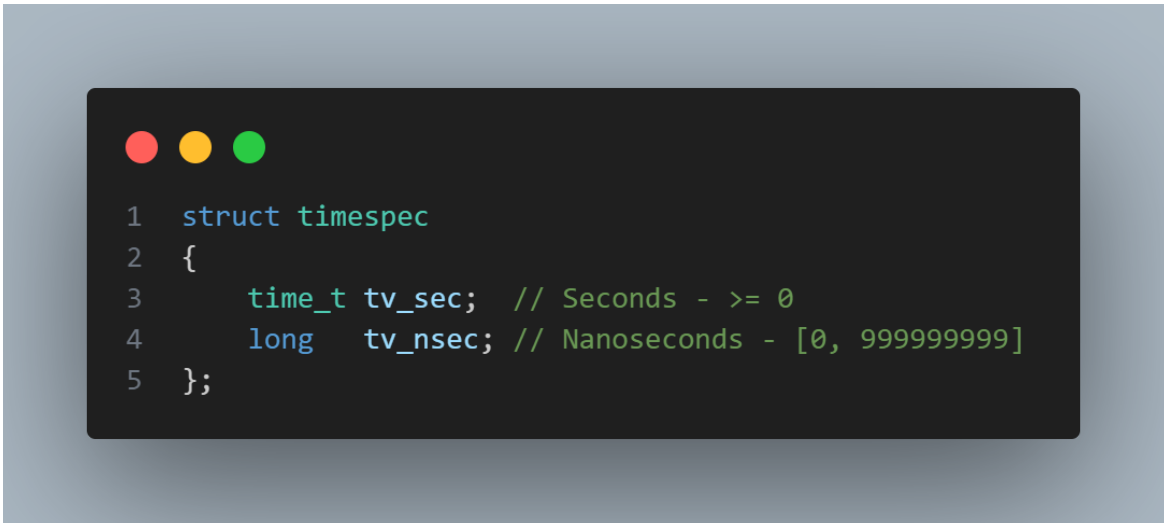
The result is very similar to Question 2, the highest priority thread (thread)is preemped by normal thread (thread 0) after the runtime restrict, but here are two normal thread, they execute alternatively.

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

```c
for (int i=0;i<3;i++)
{
    timespec starttime, currenttime;
    clock_gettime(CLOCK_THREAD_CPUTIME_ID, &starttime);

    printf("Thread %d is starting\n", thread->thread_id);
    long long start=starttime.tv_sec*1e9+starttime.tv_nsec; //ns level
    while(1)
    {
        clock_gettime(CLOCK_THREAD_CPUTIME_ID,&currenttime);
        //calculate the time diff
        long long current=currenttime.tv_sec*1e9+currenttime.tv_nsec;
        long long end=thread->busy_time*1e9;
        if(current-start>=end)
            break;
    }
```

First, get the cpu time (each threads are different) as starttime, transfrom the struct timespec to ns level time, in the while loop, keep calculating current time and execute time, until it exceed the busy-waiting time.



```
struct timespec
{
    time_t tv_sec;  // Seconds - >= 0
    long   tv_nsec; // Nanoseconds - [0, 999999999]
};
```

5. What does the kernel.sched_rt_runtime_us effect? If this setting is changed, what will happen?

The kernel.sched_rt_runtime_us effect the realtime process(thread) execute time, the value indicate that how many millisecond can the realtime process excute in 1 second.

If we change the setting to 1000000, that is, the realtime process will never preempted by normal process.

In the case sudo ./sched_demo -n 3 -t 1.0 -s NORMAL,FIFO,FIFO -p -1,10,30, change the runtime to 1000000, the result will be

```
Ian@113FallOS:~/nycu/113Fall_NYCU_OS/hw2$ cat /proc/sys/kernel/sched_rt_runtime_us
1000000
Ian@113FallOS:~/nycu/113Fall_NYCU_OS/hw2$ cat /proc/sys/kernel/sched_rt_period_us
1000000
Ian@113FallOS:~/nycu/113Fall_NYCU_OS/hw2$ sudo ./sched_demo -n 3 -t 1.0 -s NORMAL,F
IFO,FIFO -p -1,10,30
Thread 2 is starting
Thread 2 is starting
Thread 2 is starting
Thread 1 is starting
Thread 1 is starting
Thread 1 is starting
Thread 0 is starting
Thread 0 is starting
Thread 0 is starting
```