# OS project 2

張耿健 R05922092
孫凡耘 B04902045

## Part 1

1. Result

```
willy@willy-X555LB:~/Desktop/OS/pj2$ sudo ./sched_test
the number of CPUs: 1
Create Thread 0
Create Thread 1
Thread 1 is running
Thread 0 is running
Thread 1 is running
Thread 0 is running
Thread 1 is running
Thread 0 is running
After Thread 0
After Thread 1
willy@willy-X555LB:~/Desktop/OS/pj2$ sudo ./sched_test SCHED_FIFO
the number of CPUs: 1
Create Thread 0
Create Thread 1
Thread 0 is running
Thread 0 is running
Thread 0 is running
Thread 1 is running
Thread 1 is running
Thread 1 is running
After Thread 0
After Thread 1
```

Implementation details

A. Set CPU affinity

```c
//set CPU affinity
cpu_set_t cmask;
unsigned long len = sizeof(cmask);
CPU_ZERO(&cmask);       /* 初始化 cmask */
CPU_SET(0, &cmask); /* 指定第一個處理器 */
printf("the number of CPUs: %d\n", CPU_COUNT(&cmask));
/* 設定自己由指定的處理器執行 */
if (sched_setaffinity(0, len, &cmask) == -1 ) {
    printf("Could not set cpu affinity for current process.\n");
    exit(1);
}
```

B. Use sched_setscheduler() to set scheduling policy

```
//更改调用进程以使用最强的FIFO优先级
struct sched_param param;
int maxpri;
maxpri = sched_get_priority_max(SCHED_FIFO);
if(maxpri == -1) {
    perror("sched_get_priority_max() failed");
    exit(1);
}
param.sched_priority = maxpri;
//设置优先级
if (sched_setscheduler(getpid(), SCHED_FIFO, &param) == -1) {
     perror("sched_setscheduler() failed");
     exit(1);
}
```

C. Create thread

```
pthread_t tid[2];
for(int i=0;i<2;i++) {
    int *num = malloc(sizeof(int *));
    *num = i;
    printf("Create Thread %d\n", i);
    pthread_create(&tid[i], NULL, myThreadFun, (void *)num );
}
```

D. busy waiting in the thread function

```
void wait ( float seconds ) {
  clock_t endwait;
  endwait = clock () + seconds * CLOCKS_PER_SEC ;
  while (clock() < endwait) {}
}

// A normal C function that is executed as a thread when its name
// is specified in pthread_create()
void *myThreadFun(void *vargp) {
    int *num = (int *)vargp;
    for(int i = 0; i < 3; i++) {
        printf("Thread %d is running\n", *(int *)vargp);
        wait(0.5);
    }
    free(num);
}
```

E. Join thread

```
for(int i=0;i<2;i++) {
    pthread_join(tid[i], NULL);
    printf("After Thread %d\n", i);
}
```

# Part2

Result

```
spentplaying@spentplaying-VirtualBox:/usr/src/linux-2.6.32.60/test_weighted_rr$
./test_weighted_rr weighted_rr 10 5 500000000
sched_policy: 6, quantum: 10, num_threads: 5, buffer_size: 500000000
abcdeabcdeabcdeabcdabcdabcdabcabcabcabcabababababababababababa
```

Implementation details

A. enqueue_task_weighted_rr

Using function list_add_tail() to enqueue

```
static void enqueue_task_weighted_rr(struct rq *rq, struct task_struct *p, int wakeup, bool b)
{
        // not yet implemented
        struct weighted_rr_rq *wrq = &(rq->weighted_rr);
    //setting the time slice
    p->task_time_slice = p->weighted_time_slice;
    //put it into queue
    list_add_tail(&(p->weighted_rr_list_item),&(wrq->queue));
    //number of processes increased by 1
    wrq->nr_running++;
        // ...
}
```

B. dequeue_task_weighted_rr

Using function list_del() to dequeue

```
static void dequeue_task_weighted_rr(struct rq *rq, struct task_struct *p, int sleep)
{
        // first update the task's runtime statistics
        update_curr_weighted_rr(rq);
        // not yet implemented
        struct weighted_rr_rq *wrq = &(rq->weighted_rr);
        //setting time slice
        p->task_time_slice = 0;
        //delete form the waitting list
    list_del(&(p->weighted_rr_list_item));
    //number of processes decreased by 1
    wrq->nr_running--;
        // ...
}
```

C. yield_task_weighted_rr

Using  requeue function to remove the current process to the tail of queue

```
yield_task_weighted_rr(struct rq *rq)
{
        // not yet implemented
    struct task_struct *p = rq->curr;
    p->task_time_slice = p->weighted_time_slice;
        //move the process to the tail
    requeue_task_weighted_rr(rq, p);
    set_tsk_need_resched(p);
        // ...
}
```

D. pick_next_task_weighted_rr

Select a task to run , if there is no task do nothing

```
static struct task_struct *pick_next_task_weighted_rr(struct rq *rq)
{
        struct task_struct *next;
        struct weighted_rr_rq *weighted_rr_rq = &(rq->weighted_rr);
        struct list_head *queue = &(weighted_rr_rq->queue);
        // not yet implemented
        // if no task in the queue
        if(list_empty(queue)) return NULL;
        //else set the first process in the list become next process
    next = list_first_entry(queue, struct task_struct, weighted_rr_list_item);
    next->se.exec_start = rq->clock;
        /* you need to return the selected task here */
        return next;
}
```

E. task_tick_weighted_rr

Check if time is expired

```
static void task_tick_weighted_rr(struct rq *rq, struct task_struct *p,int queued)
{
        struct task_struct *curr;
        struct weighted_rr_rq *weighted_rr_rq;

        // first update the task's runtime statistics
        update_curr_weighted_rr(rq);

        // not yet implemented
        if(!task_has_weighted_rr_policy(p)) return;
    p->task_time_slice --;
    //if time expired
    if(p->task_time_slice <= 0){
       p->task_time_slice = p->weighted_time_slice;
       requeue_task_weighted_rr(rq ,p);
       set_tsk_need_resched(p);
    }
        // ...
        return;
}
```

# Bonus

Random Round-Robin Scheduler(RRR scheduler): the time slice depends on the probability completely

Implementation details

In kernel sched.c, add header #include <linux/random.h>

In sched_rrr.c modify each task_time_slice to a random num between 1 to 100

```
unsigned int randnum;
get_random_bytes(&randnum, sizeof(unsigned int));
int ratio = randnum % 100 + 1;
p->task_time_slice *= ratio;
printk("4 byte RANDNUM IS %u\n", randnum);
```

Discussion :

The time slice may be too small, which results in frequent context switch ⇒ increases overhead