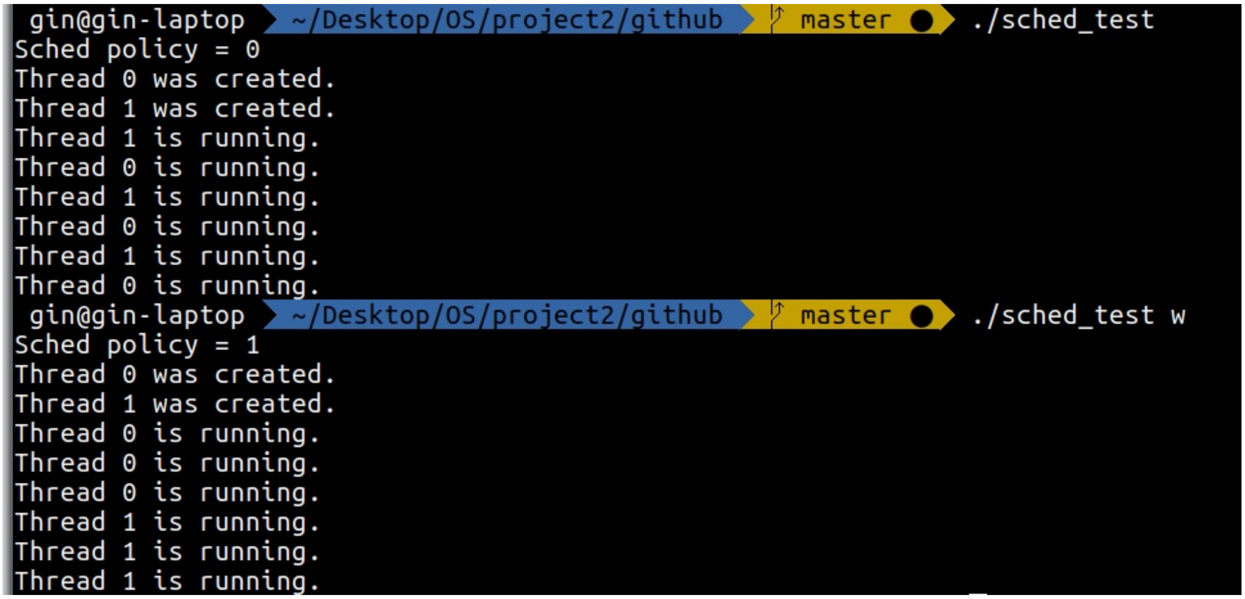
B03902078 資工二 林書瑾
B03902071 資工二 葉奕廷

Operation Systems Project 2

Part 1 Result



```
 gin@gin-laptop  ~/Desktop/OS/project2/github  master ●  ./sched_test
Sched policy = 0
Thread 0 was created.
Thread 1 was created.
Thread 1 is running.
Thread 0 is running.
Thread 1 is running.
Thread 0 is running.
Thread 1 is running.
Thread 0 is running.
 gin@gin-laptop  ~/Desktop/OS/project2/github  master ●  ./sched_test w
Sched policy = 1
Thread 0 was created.
Thread 1 was created.
Thread 0 is running.
Thread 0 is running.
Thread 0 is running.
Thread 1 is running.
Thread 1 is running.
Thread 1 is running.
```

```
Sched policy = 0
Thread 0 was created.
Thread 1 was created.
Thread 1 is running.
Thread 0 is running.
Thread 1 is running.
Thread 0 is running.
Thread 1 is running.
Thread 0 is running.
```

```
Sched policy = 1
Thread 0 was created.
Thread 1 was created.
Thread 0 is running.
Thread 0 is running.
Thread 0 is running.
Thread 1 is running.
Thread 1 is running.
Thread 1 is running.
```

Part 1 Implementation Details

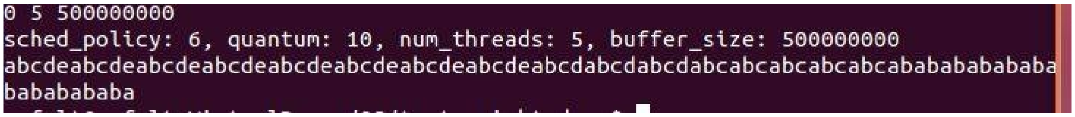
Create pthread to run busy waiting.
Busy waiting:

```
void *RunSleep(void *num)
{
  int thrnum = (int)num;
  for(int i = 0 ; i < 3 ; i++) {
    printf("Thread %d is running.\n", thrnum);
    for (int i = 0; i < 10000000; i++) {}
  }
}
```

Create pthreads:

```
pthread_t pthreads[threadnum];
for(int thr_id = 0 ; thr_id < threadnum ; thr_id++) {
  if(pthread_create(&pthreads[thr_id], NULL, RunSleep, (void *)thr_id) != 0)
    error("create thread error\n");
  else
    printf("Thread %d was created.\n", thr_id);
}
for(int i = 0 ; i < threadnum ; i++)
  pthread_join(pthreads[i], NULL);
```

Part 2 Result



```
0 5 500000000
sched_policy: 6, quantum: 10, num_threads: 5, buffer_size: 500000000
abcdeabcdeabcdeabcdeabcdeabcdeabcdeabcdeabcdabcdabcdabcabcabcabcababababababa
babababa
```

```
sched_policy: 6, quantum: 10, num_threads: 5, buffer_size: 500000000
abcdeabcdeabcdeabcdabcdabcdabcabcabcabcabababababababababababa
```

Part 2 Implementation Details

- Enqueue

```
static void enqueue_task_weighted_rr(struct rq *rq, struct task_struct *p, int wakeup, bool b)
{
    struct weighted_rr_rq *wrr_rq = &(rq->weighted_rr);
    p->task_time_slice = p->weighted_time_slice;
    list_add_tail(&(p->weighted_rr_list_item), &(wrr_rq->queue));
    wrr_rq->nr_running ++;
}
```

- 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);
  struct weighted_rr_rq *wrr_rq = &(rq->weighted_rr);
  p->task_time_slice = 0;

  list_del(&(p->weighted_rr_list_item));
  wrr_rq->nr_running --;
}
```

- Yield

```
static void yield_task_weighted_rr(struct rq *rq)
```

```
{
  struct task_struct *p = rq->curr;
  p->task_time_slice = p->weighted_time_slice;
  requeue_task_weighted_rr(rq, p);
  set_tsk_need_resched(p);
}
```

- Pick Next

```
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);
    if(list_empty(queue))
        return NULL;
    next = list_first_entry(queue, struct task_struct, weighted_rr_list_item);
    next->se.exec_start = rq->clock;

    return next;
}
```

- Task Tick

```
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);

  if(!task_has_weighted_rr_policy(p)) return;

  p->task_time_slice --;
  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 (RRR) Scheduler

We implement Random Round-Robin (RRR) scheduler. Instead of setting a weighted time slice, RRR sets the length of time slice randomly in each process.

That is, the time slice depends on the probability completely.

Implementation Deatils

In kernel sched.c:
(for get_random_bytes())

```
#include <linux/random.h>
```

In sched_rrr.c, modify each task_time_slice to a randnum between 1 to 100.

```
unsigned int randnum;
get_random_bytes(&randnum, sizeof(unsigned int));
int ratio = randnum % 100 + 1;
p->task_time_slice *= ratio;
```

Result

(The policy number is still 6.)

```
sched_policy: 6, num_threads: 5, buffer_size: 500000000
abcde
sched_policy: 6, num_threads: 5, buffer_size: 500000000
abcececececcececeeceeccececcecececececcecceeceeceeeecececececececececececececececcecece
sched_policy: 6, num_threads: 5, buffer_size: 500000000
abcdeab
```

```
sched_policy: 6, num_threads: 20, buffer_size: 500000000
abcdefghijklmnopqrstb
sched_policy: 6,  num_threads: 20,  buffer_size: 500000000
abcdefghijklmnopqrst
sched_policy: 6,  num_threads: 20,  buffer_size: 500000000
abcdefgcgcgcgcgcgcgcgcgcgcgcgcg......
```

Sometimes the time slice is too small, the times of context switch will be very large.