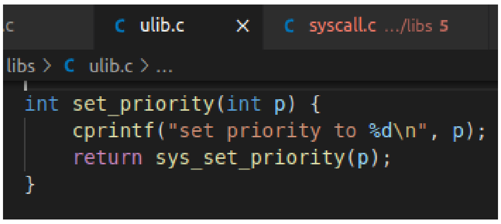
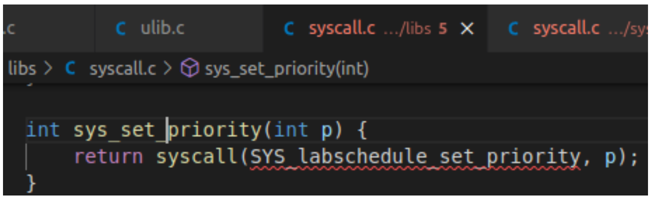
0.

Time	HRRN	FIFO	RR	SJF	Priority
1 A	A	A	A	A	A
2 B	A	A	A	A	B
3	A	A	B	A	A
4 D	A	A	A	A	D
5 C	B	B	D	B	D
6	D	D	C	D	C
7	D	D	A	D	C
8	C	C	D	C	C
9	C	C	C	C	A
10	C	C	C	C	A
Avg. Turn-around Time	4.5	4.5	5	4.5	4.25

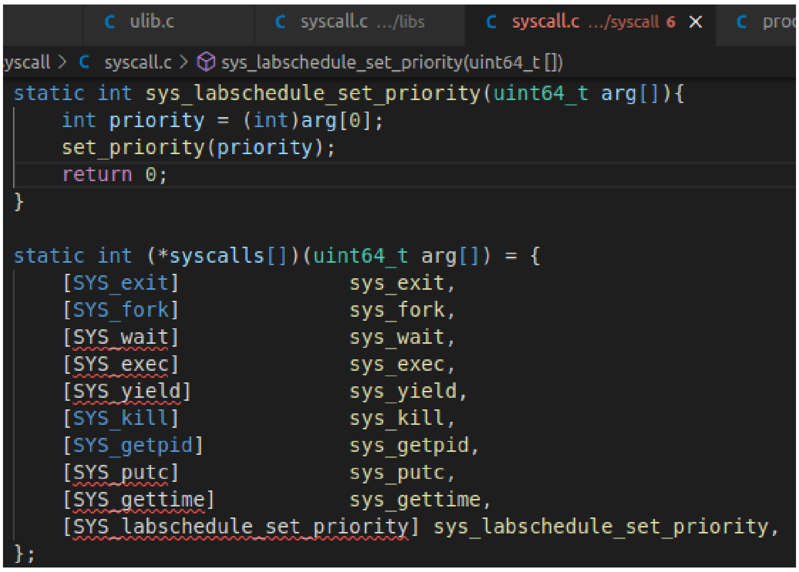
1. Design idea: add new function “set_priority” in ulib.c and add the corresponding functions that call this function one by one in the correct files. In proc.c add the “set_priority” function so that we can modify the labschedule_priority.



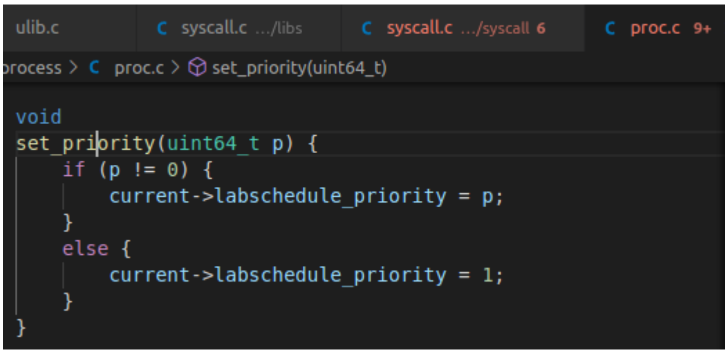
```
int set_priority(int p) {
    cprintf("set priority to %d\n", p);
    return sys_set_priority(p);
}
```



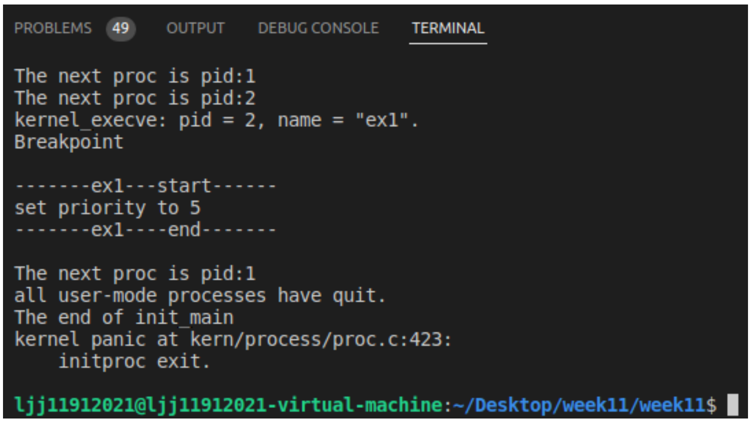
```
int sys_set_priority(int p) {
    return syscall(SYS_labschedule_set_priority, p);
}
```


ulib.c
syscall.c .../libs
syscall.c .../syscall 6
proc
yscall > syscall.c > sys_labschedule_set_priority(uint64_t [])

```
static int sys_labschedule_set_priority(uint64_t arg[]){
    int priority = (int)arg[0];
    set_priority(priority);
    return 0;
}

static int (*syscalls[])(uint64_t arg[]) = {
    [SYS_exit]              sys_exit,
    [SYS_fork]              sys_fork,
    [SYS_wait]              sys_wait,
    [SYS_exec]              sys_exec,
    [SYS_yield]             sys_yield,
    [SYS_kill]              sys_kill,
    [SYS_getpid]            sys_getpid,
    [SYS_putc]              sys_putc,
    [SYS_gettime]           sys_gettime,
    [SYS_labschedule_set_priority] sys_labschedule_set_priority,
};
```

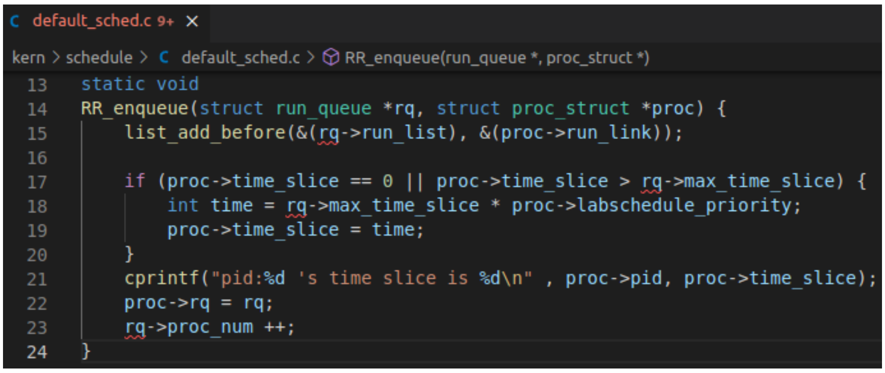


```
void
set_priority(uint64_t p) {
    if (p != 0) {
        current->labschedule_priority = p;
    }
    else {
        current->labschedule_priority = 1;
    }
}
```

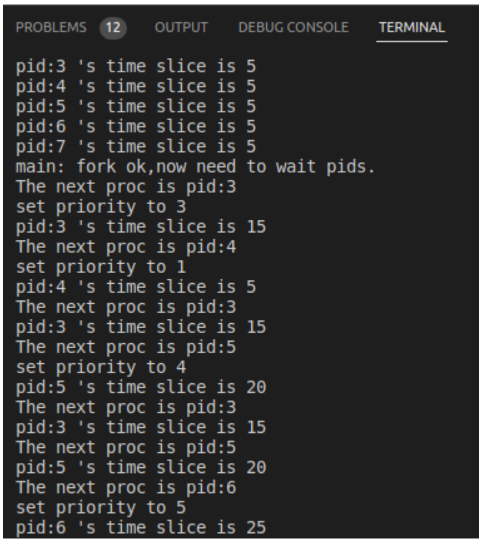


```
The next proc is pid:1
The next proc is pid:2
kernel_execve: pid = 2, name = "ex1".
Breakpoint

-------ex1---start------
set priority to 5
-------ex1----end-------

The next proc is pid:1
all user-mode processes have quit.
The end of init_main
kernel panic at kern/process/proc.c:423:
    initproc exit.

ljj11912021@ljj11912021-virtual-machine:~/Desktop/week11/week11$
```

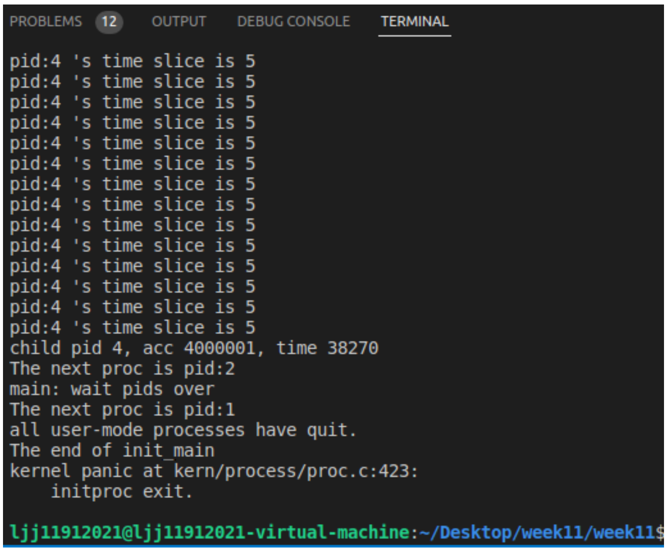
2. Design idea: modify the RR_enqueue function in default_sched.c, by setting the time slice to max_time_slice * priority and print it out.


default_sched.c 9+ ×
kern > schedule > default_sched.c > RR_enqueue(run_queue *, proc_struct *)

```
static void
RR_enqueue(struct run_queue *rq, struct proc_struct *proc) {
    list_add_before(&(rq->run_list), &(proc->run_link));

    if (proc->time_slice == 0 || proc->time_slice > rq->max_time_slice) {
        int time = rq->max_time_slice * proc->labschedule_priority;
        proc->time_slice = time;
    }
    cprintf("pid:%d 's time slice is %d\n" , proc->pid, proc->time_slice);
    proc->rq = rq;
    rq->proc_num ++;
}
```

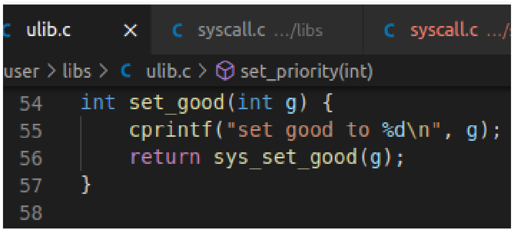


```
pid:3 's time slice is 5
pid:4 's time slice is 5
pid:5 's time slice is 5
pid:6 's time slice is 5
pid:7 's time slice is 5
main: fork ok,now need to wait pids.
The next proc is pid:3
set priority to 3
pid:3 's time slice is 15
The next proc is pid:4
set priority to 1
pid:4 's time slice is 5
The next proc is pid:3
pid:3 's time slice is 15
The next proc is pid:5
set priority to 4
pid:5 's time slice is 20
The next proc is pid:3
pid:3 's time slice is 15
The next proc is pid:5
pid:5 's time slice is 20
The next proc is pid:6
set priority to 5
pid:6 's time slice is 25
```

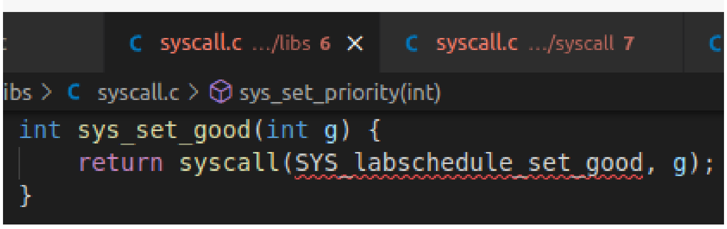


```
pid:4 's time slice is 5
pid:4 's time slice is 5
pid:4 's time slice is 5
pid:4 's time slice is 5
pid:4 's time slice is 5
pid:4 's time slice is 5
pid:4 's time slice is 5
pid:4 's time slice is 5
pid:4 's time slice is 5
pid:4 's time slice is 5
pid:4 's time slice is 5
pid:4 's time slice is 5
pid:4 's time slice is 5
pid:4 's time slice is 5
child pid 4, acc 4000001, time 38270
The next proc is pid:2
main: wait pids over
The next proc is pid:1
all user-mode processes have quit.
The end of init_main
kernel panic at kern/process/proc.c:423:
    initproc exit.

ljj11912021@ljj11912021-virtual-machine:~/Desktop/week11/week11$
```

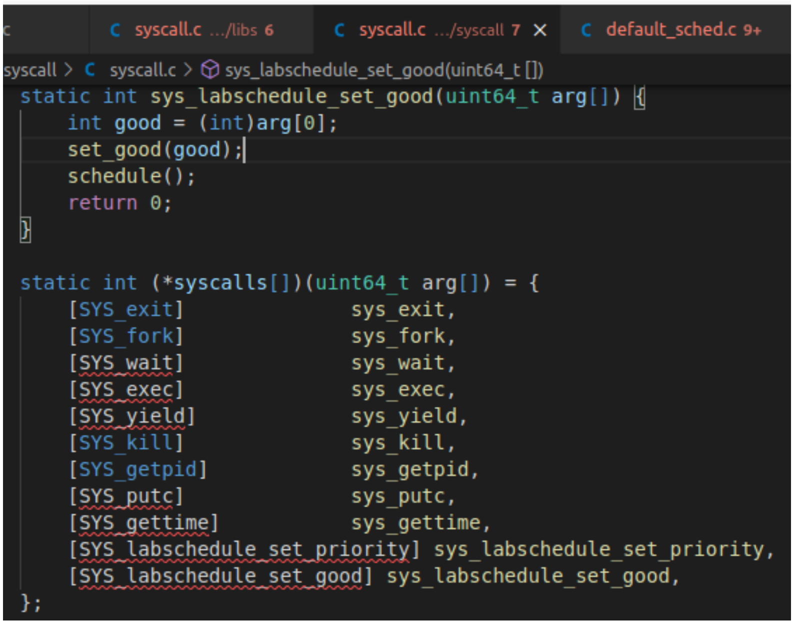
3. Design idea: add set_good function like the procedure in ex1, but modify the RR_enqueue function so that it traverses the queue to get the process with the largest good value to run. In syscall, we use schedule function, so that whenever schedule is called, it calls the enqueue function and the process with larger good value will be run first.



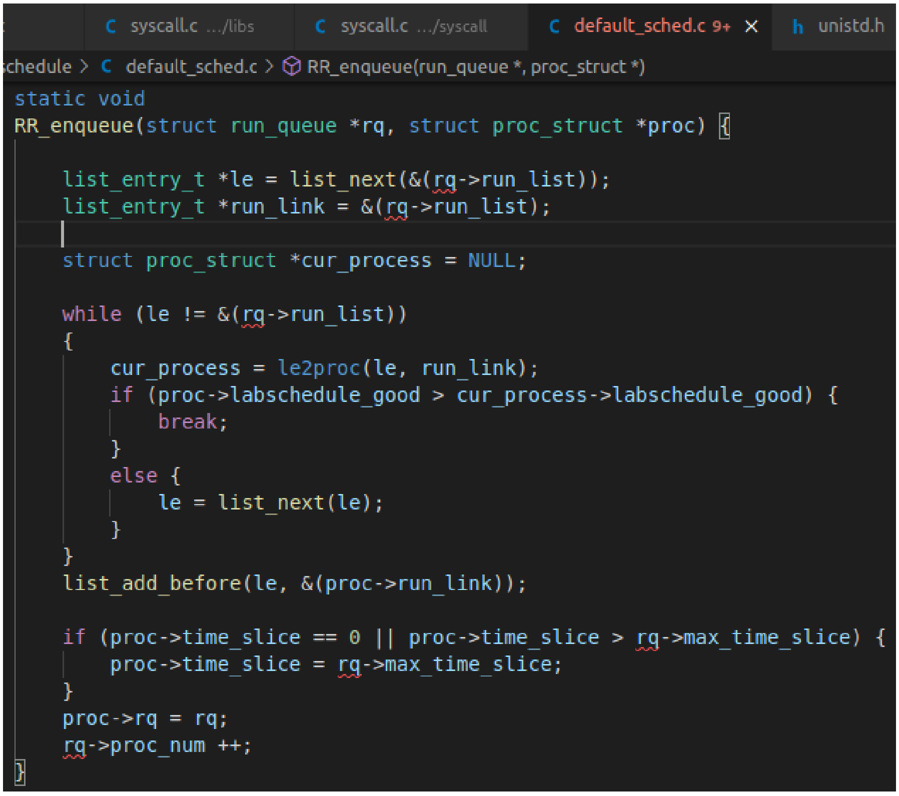
```c
int set_good(int g) {
    cprintf("set good to %d\n", g);
    return sys_set_good(g);
}
```



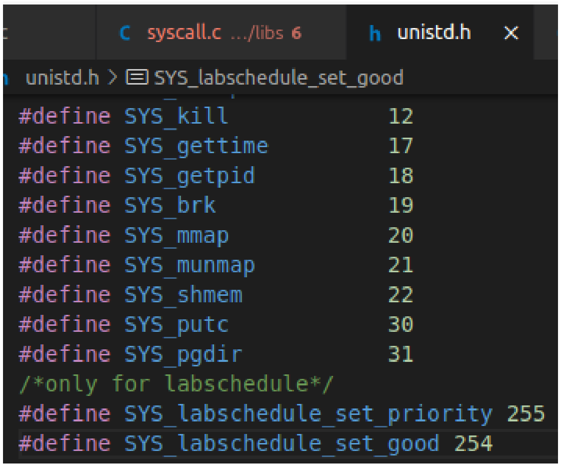
```c
int sys_set_good(int g) {
    return syscall(SYS_labschedule_set_good, g);
}
```



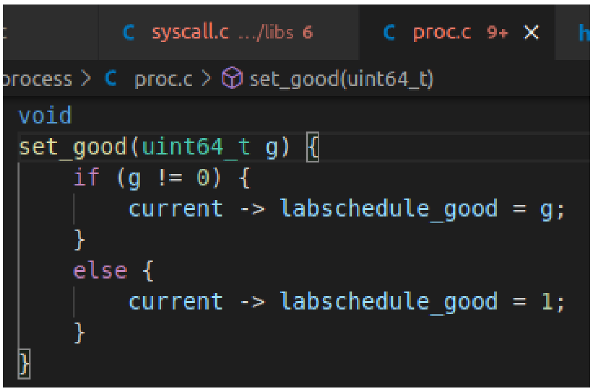
```c
static int sys_labschedule_set_good(uint64_t arg[]) {
    int good = (int)arg[0];
    set_good(good);
    schedule();
    return 0;
}

static int (*syscalls[])(uint64_t arg[]) = {
    [SYS_exit]              sys_exit,
    [SYS_fork]              sys_fork,
    [SYS_wait]              sys_wait,
    [SYS_exec]              sys_exec,
    [SYS_yield]             sys_yield,
    [SYS_kill]              sys_kill,
    [SYS_getpid]            sys_getpid,
    [SYS_putc]              sys_putc,
    [SYS_gettime]           sys_gettime,
    [SYS_labschedule_set_priority] sys_labschedule_set_priority,
    [SYS_labschedule_set_good] sys_labschedule_set_good,
};
```


syscall.c .../libs
syscall.c .../syscall
default_sched.c 9+
unistd.h
chedule > default_sched.c > RR_enqueue(run_queue *, proc_struct *)

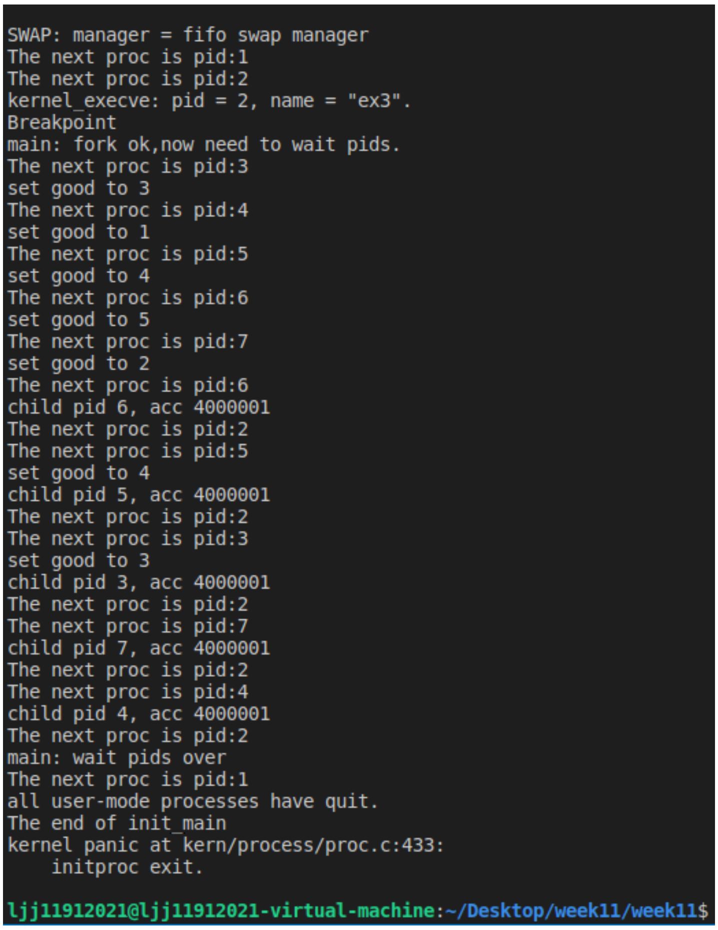
```c
static void
RR_enqueue(struct run_queue *rq, struct proc_struct *proc) {

    list_entry_t *le = list_next(&(rq->run_list));
    list_entry_t *run_link = &(rq->run_list);

    struct proc_struct *cur_process = NULL;

    while (le != &(rq->run_list))
    {
        cur_process = le2proc(le, run_link);
        if (proc->labschedule_good > cur_process->labschedule_good) {
            break;
        }
        else {
            le = list_next(le);
        }
    }
    list_add_before(le, &(proc->run_link));

    if (proc->time_slice == 0 || proc->time_slice > rq->max_time_slice) {
        proc->time_slice = rq->max_time_slice;
    }
    proc->rq = rq;
    rq->proc_num ++;
}
```



```
#define SYS_kill              12
#define SYS_gettime           17
#define SYS_getpid            18
#define SYS_brk               19
#define SYS_mmap              20
#define SYS_munmap            21
#define SYS_shmem             22
#define SYS_putc              30
#define SYS_pgdir             31
/*only for labschedule*/
#define SYS_labschedule_set_priority 255
#define SYS_labschedule_set_good 254
```



```
void
set_good(uint64_t g) {
    if (g != 0) {
        current -> labschedule_good = g;
    }
    else {
        current -> labschedule_good = 1;
    }
}
```



```
SWAP: manager = fifo swap manager
The next proc is pid:1
The next proc is pid:2
kernel_execve: pid = 2, name = "ex3".
Breakpoint
main: fork ok,now need to wait pids.
The next proc is pid:3
set good to 3
The next proc is pid:4
set good to 1
The next proc is pid:5
set good to 4
The next proc is pid:6
set good to 5
The next proc is pid:7
set good to 2
The next proc is pid:6
child pid 6, acc 4000001
The next proc is pid:2
The next proc is pid:5
set good to 4
child pid 5, acc 4000001
The next proc is pid:2
The next proc is pid:3
set good to 3
child pid 3, acc 4000001
The next proc is pid:2
The next proc is pid:7
child pid 7, acc 4000001
The next proc is pid:2
The next proc is pid:4
child pid 4, acc 4000001
The next proc is pid:2
main: wait pids over
The next proc is pid:1
all user-mode processes have quit.
The end of init_main
kernel panic at kern/process/proc.c:433:
    initproc exit.

ljj11912021@ljj11912021-virtual-machine:~/Desktop/week11/week11$
```