1. CPU Scheduling

Time	HRRN	FIFO/FCFS	RR	SJF	Priority
1	A	A	A	A	A
2	A	A	A	A	B
3	A	A	B	A	A
4	A	A	A	A	D
5	B	B	D	B	D
6	D	D	A	D	C
7	D	D	C	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	4.75	4.5	4.25

2. Preemptive process scheduling

Design idea:通过触发 ecall 进行系统调用，实现 U 态函数 set_good 经过层层包装在 S 态下修改进程的属性 labschedule_good，同时设置当前进程 need_resched 为 1，并通过文件中自带的 skew_heap，包装后实现以 labschedule_good 为 key 的大顶堆（值相同时，pid 小为堆顶），从而通过该大顶堆进行 CPU 调度。

the running sequence of processes ：

proc1->proc2->proc3->proc4->proc5->proc6->proc7->proc6->proc2->proc5->proc2->proc3->proc2->proc7->proc2->proc4->proc2->proc1.

Modified code:

ulib.c

```
49  void
50  set_good(int good){
51      cprintf("set good to %d\n", good);
52      return sys_setgood(good);
53  }
54
```

user/libs/syscall.c

```
79  int
80  sys_setgood(int64_t good){
81      return syscall(SYS_setgood, good);
82  }
83
```

kern/syscall/syscall.c

```
66  static int sys_setgood(uint64_t arg[]){
67      return do_setgood(arg[0]);
68  }
69
70  static int (*syscalls[])(uint64_t arg[]) = {
71      [SYS_exit]              sys_exit,
72      [SYS_fork]              sys_fork,
73      [SYS_wait]              sys_wait,
74      [SYS_exec]              sys_exec,
75      [SYS_yield]             sys_yield,
76      [SYS_kill]              sys_kill,
77      [SYS_getpid]            sys_getpid,
78      [SYS_putc]              sys_putc,
79      [SYS_gettime]           sys_gettime,
80      [SYS_setgood]           sys_setgood,
81  };
```

default_sched.c

```c
static int

heap_comp_f(void *a, void *b)
{
    struct proc_struct *p = le2proc(a, run_pool_entry);
    struct proc_struct *q = le2proc(b, run_pool_entry);
    int32_t c = p->labschedule_good - q->labschedule_good;
    if (c > 0) return -1;
    else if (c == 0){
       if(p->pid < q->pid){
           return -1;
       }else{
           return 1;
       }
    }else return 1;
}

static void
Good_init(struct run_queue *rq) {
    list_init(&(rq->run_list));
    rq->labschedule_run_pool = NULL;
    rq->proc_num = 0;
}
```

```c
static void
Good_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) {
    // //     proc->time_slice = rq->max_time_slice;
    // // }
    rq->labschedule_run_pool = skew_heap_insert(rq->labschedule_run_pool, &(proc->run_pool_entry),heap_co
    proc->rq = rq;
    rq->proc_num ++;
}

static void
Good_dequeue(struct run_queue *rq, struct proc_struct *proc) {
    // assert(!list_empty(&(proc->run_link)) && proc->rq == rq);
    // list_del_init(&(proc->run_link));
    rq->labschedule_run_pool = skew_heap_remove(rq->labschedule_run_pool, &(proc->run_pool_entry),heap_co
    rq->proc_num --;
}
```

```
52  Good_pick_next(struct run_queue *rq) {
53      // list_entry_t *le = list_next(&(rq->run_list));
54      // if (le != &(rq->run_list)) {
55      //     return le2proc(le, run_link);
56      // }
57      if(rq->labschedule_run_pool != NULL){
58          struct proc_struct *p = le2proc(rq->labschedule_run_pool, run_pool_entry);
59          return p;
60      }
61      else{
62          return NULL;
63      }
64      // list_entry_t *le = list_next(&(rq->run_list));
65
66      // if (le == &rq->run_list)
67      //     return NULL;
68
69      // struct proc_struct *p = le2proc(le, run_link);
70      // le = list_next(le);
71      // while (le != &rq->run_list)
72      // {
73      //  struct proc_struct *q = le2proc(le, run_link);
74      //     if ((int32_t)(p->labschedule_good - q->labschedule_good) < 0 ||
75      //     (int32_t)(p->labschedule_good - q->labschedule_good)== 0 && q->pid < p->pid){
76      //         p = q;
77      //     }
78      //     le = list_next(le);
79      // }
80      // return p;
81  }
```

```
83  static void
84  Good_proc_tick(struct run_queue *rq, struct proc_struct *proc) {
85      // if (proc->time_slice > 0) {
86      //     proc->time_slice --;
87      // }
88      // if (proc->time_slice == 0) {
89      //     proc->need_resched = 1;
90      // }
91  }
92
93  struct sched_class default_sched_class = {
94      .name = "Good_scheduler",
95      .init = Good_init,
96      .enqueue = Good_enqueue,
97      .dequeue = Good_dequeue,
98      .pick_next =Good_pick_next,
99      .proc_tick = Good_proc_tick,
100 };
101
```

proc.c

```
647 // only for labschedule_good
648 int
649 do_setgood(int good) {
650     current->labschedule_good = good;
651     current->need_resched = 1;
652     return 0;
653 }
```

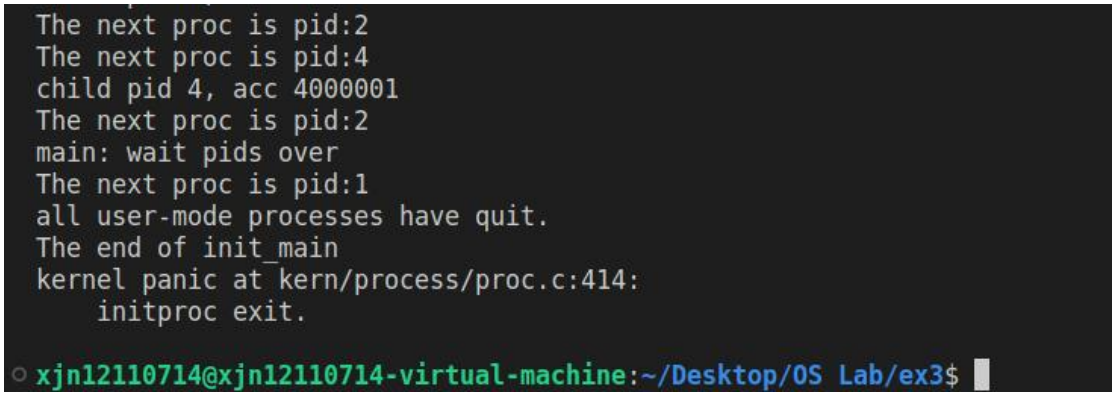
unistd.h

```
16   /*only for labschedule_good*/
17   #define SYS_setgood          16
18
```

Tips:default_sched.c 中大段注释的代码为用链表实现该题功能。

Running result

```
memory management: default_pmm_manager
physcial memory map:
  memory: 0x08800000, [0x80200000, 0x885fffff].
sched class: Good_scheduler
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:414:
initproc exit.
xjn12110714@xjn12110714-virtual-machine:~/Desktop/OS Lab/ex3$