Assignment 3

Please complete a report.

The files to be submitted for this assignment are:

1. report.pdf

1. CPU Scheduling [50pts]

Consider the following single-threaded processes, and their arrival times, estimated CPU costs and their priorities (a process with a higher priority number has priority over a process with lower priority number):

Process	Estimated CPU Cost	Arrives	Priority
A	6	1	1
B	1	2	2
C	3	5	3
D	2	4	4

Please note:

- Ignore context switching overhead.
- If a process arrives at time x, they are ready to run at the beginning of time x.
- Highest response ratio next (HRRN) is a non-preemptive scheduling algorithm. In HRRN, the next job is not that with the shorted estimated run time, but that with the highest response ratio defined as: 1 + waiting time / estimated CPU time.
- Newly arrived processes are scheduled last for RR. When the RR quanta expires, the currently running thread is added at the end of to the ready list before any newly arriving threads.
- The quanta for RR is 1 unit of time.
- Average turn-around time is the average time a process takes to complete after it arrives.
- SJF is non_preemptive.
- Priority scheduler is preemptive.

Given the above information please fill in the following table.

Time	HRRN	FIFO/FCFS	RR	SJF	Priority
1	A	A	A	A	A

Time	HRRN	FIFO/FCFS	RR	SJF	Priority
2	A	A	A	A	B
3	A	A	B	A	A
4	A	A	A	A	D
5	A	A	D	A	D
6	A	A	A	A	C
7	B	B	C	B	C
8	D	D	D	D	C
9	D	D	A	D	A
10	C	C	C	C	A
11	C	C	A	C	A
12	C	C	C	C	A
Avg. Turn-around Time	6.5	6.5	6.5	6.5	4.75

2. Process scheduling [50pts]

Core code in schedule algorithm:

```c
// kern/schedule/default_sched.c

// pick the 'goddest' for the next item
static struct proc_struct *RR_pick_next(struct run_queue *rq)
{
    list_entry_t *le = list_next(&(rq->run_list));
    if (le != &(rq->run_list)) {
        int max_goodness = -1;
        struct proc_struct *selected_proc = NULL;
        for (; le != &(rq->run_list); le = list_next(le)) {
            struct proc_struct *cur_proc = le2proc(le, run_link);
            int cur_goodness = cur_proc->labschedule_good;
            if (cur_goodness > max_goodness) {
                selected_proc = cur_proc;
                max_goodness = cur_goodness;
            }
        }
        return selected_proc;
    }
    return NULL;
}

// Disable the time interrput tick
static void RR_proc_tick(struct run_queue *rq, struct proc_struct *proc)
{
    // if (proc->time_slice > 0) {
    //  proc->time_slice--;
    // }
    // if (proc->time_slice == 0) {
    //  proc->need_resched = 1;
    // }
}

```