Name: Ningyuan Zhang
Section: G
University ID: 708235564

Lab 5b Report

Lab Questions:

2.1:

2pts What does the *#define* on line 26 do?

Define ver. of constant time of RR scheduler is 1.3

2pts Which lines are included and which lines are excluded due to the *#if* on line 43, assuming we are using kernel 2.4.06?

Included: 44-45
Excluded: 47-48

2pts What is the difference between the two? (Hint: describe the type of each declaration)

Sched_policy *ctrr_of_cpu(NR_CPUS)'s parameter is the # of CPUs that kernel will use. sched_policy round_robin does not have such one.

2.3:

2pt What does the following code do (line 158)?

It inits a new schedule policy, which is RR

4pts What is the purpose of lines 161 and 162?

161: chooses the tasking going to process and pass it to a new var to store it
162: gets the preempt ability of two different tasks and return positive number if the thief is better, negative number if current task is better, then pass it to a new var to store it

2pts Find the lines that initialize this array.

166

2pts Which lines handle the unregistration of the module?

167-170

2.4:

3pts Based on line 161, what must this function do? *ctrr_choose_task* takes two parameters: the currently running task, and the number of the CPU running the scheduler.

Within the two parameters, the function must return an idle task for the given CPU. It could be current task or the first task in the queue, and it depends on if we could keep running current task or we need to yield for other tasks

4pts Describe lines 67-70 in words. Translate the syntax into plain English one line at a time, but keep it brief.

If current match all following requirements, keep running it.

1. Current task's state is running.
2. Current task is NOT in the scheduled yield policy.
3. Current task is NOT the idle task for the particular CPU.
4. Current task has ticks left in the process.

3pts On what line is the counter replenished? Does this happen for all tasks every time this algorithm is invoked?

93
No, it only happens when current task is not decided to stay, and current task gets few # of ticks left there.

2pts Study lines 97-139. In what case is the idle task returned?

Ver. before 2.4: when current task is decided to stay
Ver. after 2.4: when current task is decided to stay, the other tasks have few # of ticks left there

3pts On line 142, what is the purpose of the function *ctrr_preemptability* (in words)?

Check the preempt ability of two different tasks, current task and thief task. It will return positive number if the thief is better, negative number if current task is better. 0 if these two are same

3:

3pts Find all of the lines in *pset.c* where goodness is used. Note any differences in the parameters passed.

132 & 142
Chosen tasks are different among them. 132 is current task and 142 is from queue

2pts What is *IDLE_WEIGHT* defined as? Why? You will need to look in the include files to answer the first part, and the definition of goodness for the second part.(Refer to the patched sched.h file we provide.)

-1000, which means never pick this.

3pts Where is *can_choose* defined? Where is *is_visible* defined? When answering, pick the correct lines based on the values of *CONFIG_SMP* we gave you.

Can_choose: line 94
Is_cisible: line 85

4pts What are the initial values of *high* and *choice*?

High: -1000
Choice: value of *idle*

3pts Under what conditions is choice set to *curr_task*? (lines 130-135)

Current task is running, is visible to CPU and goodness value > 0

5pts In one sentence, which task does the code on lines 137-148 pick?

The most runnable task in pset of CPU.

2pts If there are tasks that are runnable but have no ticks left, then what is the value of *high*? (see line 151)

0

2pts Where does line 158 goto?

Line 126

5pts Will lines 150-159 ever run twice in a row? Why or why not? In what two cases will it not run, and what task does the algorithm return in those two cases?

Look again at lines 130-135.

We need to see the next task of the head of queue. If goodness value = 0 for the task this time, it runs again, or, it does not. In the first case, it will recheck and maybe it will return the next task. In the second case, it will return the head task of the queue

2pts - Extra If this *"if"* is removed (the whole thing: lines 130-135), in what case will the task chosen be different? Note that the list of tasks is not sorted in any particular manner. Think of cases where some of the tasks have the same goodness, and the current task is closer to the end of the list.

When current task has excellent value of goodness i.e., its goodness value > 0. In this case, if the "if" is removed, it will choose the head task or recheck instead of choosing current task

2pts - Extra Why include this *"if"*? Think in terms of penalties incorporated into goodness.

When current task's goodness > 0, it doesn't have to yield for other tasks