Operating System Project2 Report: Android Scheduler

1.Scheduling Class

The android system has three scheduling class: SCHED_NORMAL, SCHED_FIFO, SCHED_RR.

SCHED_NROMAL is a completely fair scheduler, use a red-black tree to implements a "timeline" of future task execution. Every time when running a new process, it will choose the leftmost leaf of the red-black tree, which has the lowest spent execution time. Certainly, every process has priority, which slightly affect the scheduling. The priority will be used when calculate the execution time of the process. Its time complexity is O (log n).

SCHED_FIFO is a real-time first in first out scheduler. It orders the processes with their real-time priority. Every priority has a process list, which organizes processes with first in first out queue. Every time when running a new process, it will choose the highest priority process list, and choose the head of this list, in other word, the oldest process of the list. Its time complexity is O (1).

SCHED_RR is a real-time round robin scheduler. It also orders the processes with their real-time priority. Every priority has a process list, organized with queue. Every time of context switch, it will choose the highest priority and pick up next task to run. Its time complexity is O (1).

Comparison:

The SCHED_FIFO and SCHED_RR are real-time schedulers. They have higher priority than the SCHED_NORMAL.

The SCHED_RR and SCHED_NORMAL all align time slices to process, which is more fair to the process than the SCHED_FIFO. SCHED_FIFO may let the process which has the same priority wait for a long time.

The SCHED_FIFO and SCHED_RR use real-time priority to pick up task, which may cause starvation, since process with low priority may never be picked. The SCHED_NORMAL doesn't have this problem, since in its algorithm, all processes would be fairly served.

2.My Program

For the first problem to familiar with the scheduler in Linux, I write four program:

ptreeSyscall is edited from the project1. It's a system call module. It uses depth first search to search the whole process tree, and return the process tree with some parameters in task_strcut as we need. I use it to get process id of some process, so that I can change its scheduler, and also get the entire tree for debugging.

callPtree is also edited from project1. It calls ptreeSyscall to get the process tree and print it on the screen. It is mainly used for printing the scheduler and the priority for processes when debugging.

Test is used to calculate the executing time of simple loops on behalf of Linux executable files. It can change its own policy and priority.

SetScheduler is used to change the scheduler of the android application. It gets the process id from the ptree system call, then change the process we want into a new scheduler according to the parameters in command line.

For the second problem to modify the scheduler in Linux kernel, I change two code files:
core.c:

Change the default scheduler for the descendants of process zygote.

Since I tried to set scheduler when the task initialized its scheduler or when it joined in the cpu running list, but they all failed. Therefore, I use dfs like the syscall in project1 to change all the descendants of main when a new task joins in the cpu running list. Since the task number and the running task number are all small, the time complexity is acceptable. However, it brings a problem that we can't change their scheduler anymore, since every task woken up will update their scheduler to be default. As we just want to change their scheduler and see the difference, I think this bug won't affect my work, so after other change can't work, I choose this change to finish the modification.

All changes are included in two macros. The CHANGE macro includes the change of scheduler, and the DEBUG macro includes a few debug codes to print about normal scheduler works.

rt.c:

Change the way for picking next task of the real-time scheduler.

The real-time scheduling class has two levels of scheduling. The first is priority pick and the second is group scheduling. In order to pick task randomly, I use Mersenne Twister to generate random number. I tried two methods, one is just pick priority randomly, the other is pick priority as well as the task in the priority group randomly. When debugging, I found that the second method couldn't pick a random task (I don't know why), while the first one can basically pick tasks randomly. Thus, I choose it to implement the scheduling to pick next task randomly.

Like core.c, all changes are included in two macros. The CHANGE macro includes the change of picking way, and the DEBUG macro includes a few debug codes to print about real-time scheduler works.

3.Result

For the first problem, I run the two programs separately and together.

Since other programs hardly consume the cpu resources, when these two programs run separately, their executing time in every scheduler looks similar.

For running two test applications together, as the result, it costs least time in the SCHED_FIFO scheduler. This is because the first program runs nearly the same as it runs separately, and the second program doesn't calculate its executing time until it is run in cpu. The SCHED_RR is a little faster than SCHED_NORMAL, since other programs aren't real-time so they haven't been run while these two programs are running.

When all descendants of zygote are set to SCHED_RR, they will occupy a few resources in cpu. Thus it would be a little slower than they all SCHED_NORMAL. The priority two test applications will not affect the running time.

For the second problem, since the scheduler pick new task randomly, it would hardly be affect by the priority. But the time complexity is larger than the default one, so that the executing time of test applications would be a little larger.

The first time I ran the test applications I get very good performance. But other time I ran them, this performance didn't appear any more. I captured it, but I can't explain why. Maybe it is because the different performance of the host computer I guess.

Since the executing time isn't large, and I use virtual machine to run the Ubuntu, whose performance will be affect by the host machine, the result has random error. Thus the result analysis may be not so accurate.