OS Project2: Android scheduler

叶增渝 519030910168

Introduction:

CPU scheduling is the basis of multithreading operating system. By building more different scheduling method, OS could make process execution more effective. In this project, we need to build a WRR scheduling method in Android.

Round-robin scheduling classify tasks into foreground groups and background groups. WRR assigned more milliseconds as a time slice for foreground groups. (In our problem, 100ms for fore and 10ms for back)

Analysis:

Every scheduler is defined through a sched_class, which contains as set of pointers. These pointers connect the function pointers with their corresponding implementations. For wrr scheduling, there's a wrr_sched_class. The core scheduler manages ready threads by enqueueing them into a run queue, which is implemented by struct rq. In the operating system, there are several CPU to execute tasks in the running list. When running, every schedule will own an entity which contains some necessary information. As for cfs_rq, it's a tree whose nodes are entities of tasks. OS will gradually finish all the tasks from the left lowest of the tree.

What we should do is to set some extra definition, revise some functions in the core.c and implement the functions in wrr.c like rt.c.

Implementation:

1./arch/arm/configs/goldfish_armv7_defconfig:

In line 13, add CONFIG_WRR_GROUP_SCHED=y

2./include/linux/sched.h

In line 42, add #define SCHED_WRR 6

In line 153, add struct wrr_rq;

In line 1252, add struct define sched_wrr_entity. In sched_wrr_entity, we have variable time_slice is to record the time running at RR, value will be 1 when task is in foreground and be 0 in background, run_list is used to connect the element in the same queue. To make WRR schedule simple and work well, we assume that all tasks in wrr will have a priority of 0.

In line 1264, add #define WRR_FOREGROUND_TIMESLICE as 100ms and WRR_BACKGROUND_TIMESLICE as 10ms.

3. /kernel/sched/Makefile

In line 14, add wrr.o at the end.

4./kernel/sched/sched.h

In line 82, add declaration of wrr_rq

In line 125, add members in the task_group

In line 318, add struct define wrr_rq. In wrr_rq, nr_wrr_running is used to record the number of tasks in the queue, total_value is used to record the sum of value

In line 389, add member wrr in the struct rq

In line 398, add a new list_head in the struct rq

In line 865, add extern const struct sched_class wrr_sched_class;

In line 1162, add extern void init_wrr_rq(struct wrr_rq *wrr_rq, struct rq *rq);

5./kernel/sched/core.c

In _sched_fork() function in line 1709, add INIT_LIST_HEAD(&p->wrr.run_list);

In scheduler_tick() function in line 3190, add information print when in WRR scheduling

In rt_mutex_setprio() function in line 3986, add priority setting when in WRR scheduling

In __setscheduler() function in line 4210, add sched_class setting when in WRR scheduling

In __sched_setscheduler () function in line 4246, add information print when in WRR __setscheduler especially the information of tasks in foreground or background

In do_set_cpus_allowed () function in line 5172, add allowed number of cpus setting when in WRR scheduling

In sched_init () function in line 7139, add initial setting of rq when in WRR scheduling

6./kernel/sched/wrr.c

In line 334, we define the struct wrr_sched_class, which is the major part of WRR. For RT didn't use task_fork() function and we don't' need to implement it, I revise wrr_sched_class to make it work correctly.

As for all functions implemention, I mainly follow and revise the corresponding functions in rt.c. We combine all the revision by using SMP and CONFIG_WRR_GROUP_SCHED. Also I add the state checking code in function in task_tick_wrr() function. Detailed implementation please check the source code.

Additionally, for function set_cpus_allowed_wrr(), I think it's just a basic WRR, we don't, thus it's unnecessary. And I don't find reference about the function task_fork_wrr(), thus I just comment out it.

7./kernel/sched/rt.c

In line 2038 of rt_sched_class definition, the next pointer is changed to wrr_sched_class, making all kinds of schedule policy becoming a correct circle.

Test Program:

1.set_sched.c

It can change the referred task into other schedule and priority

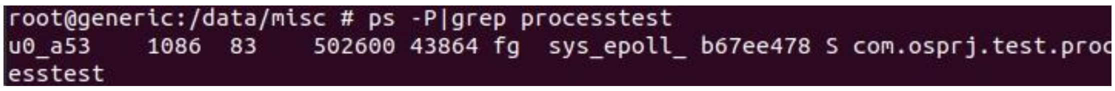
2.wrr_timeslice.c

It can tell the task's current task's schedule, tell timeslice if in WRR or RR and tell the state if in WRR.

set_sched.c is built by the syscall sched_getscheduler() and wrr_timeslice is built by the syscalls sched_getscheduler() and sched_rr_get_interval(). To use these two functions, we need first use ps -P|grep processtest(processtest is the app name) to get pid of the process, then according to pid, we could get information and set schedule and priority.

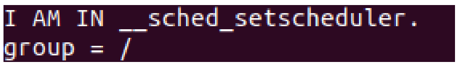
Test Result:

1.first, we use ps -P|grep processtest to get pid, we could see its pid 1119



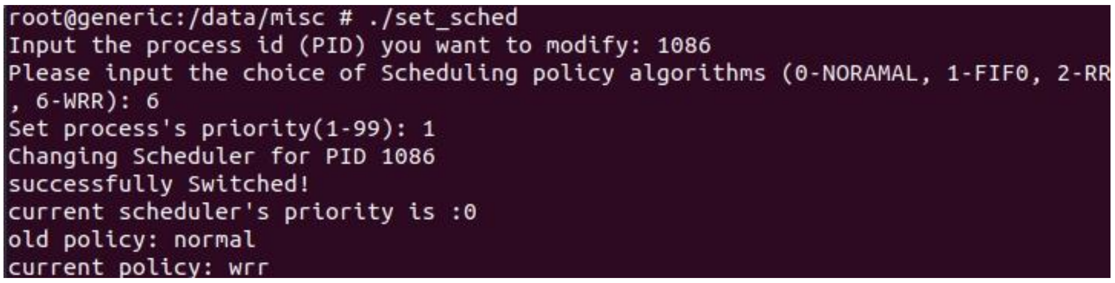
```
root@generic:/data/misc # ps -P|grep processtest
u0_a53    1086  83    502600 43864 fg  sys_epoll_ b67ee478 S com.osprj.test.proc
esstest
```

2.then show the kernel information

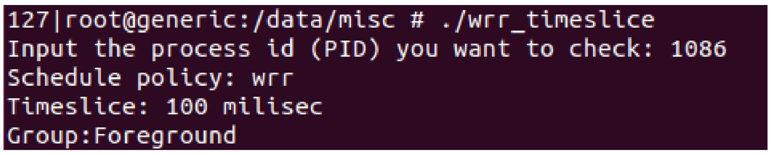


```
I AM IN __sched_setscheduler.
group = /
```

3.we use set_sched to change the process to WRR, we could see that through wrr_timeslice's information, it's changed to WRR and owns 100ms timeslice

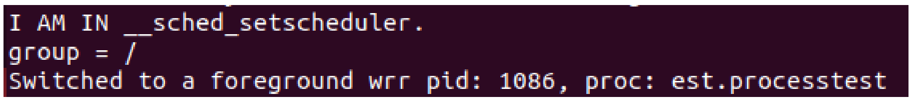


```
root@generic:/data/misc # ./set_sched
Input the process id (PID) you want to modify: 1086
Please input the choice of Scheduling policy algorithms (0-NORAMAL, 1-FIFO, 2-RR
, 6-WRR): 6
Set process's priority(1-99): 1
Changing Scheduler for PID 1086
successfully Switched!
current scheduler's priority is :0
old policy: normal
current policy: wrr
```



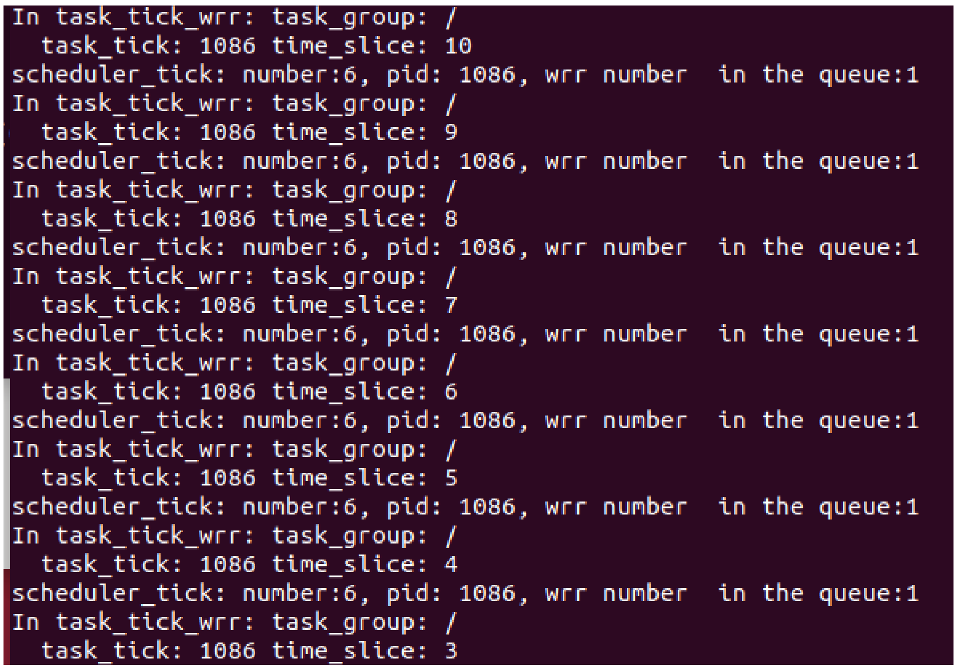
```
127|root@generic:/data/misc # ./wrr_timeslice
Input the process id (PID) you want to check: 1086
Schedule policy: wrr
Timeslice: 100 milisec
Group:Foreground
```

kernel information also indicates its pid and in foreground



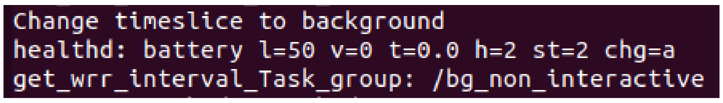
```
I AM IN __sched_setscheduler.
group = /
Switched to a foreground wrr pid: 1086, proc: est.processtest
```

Besides, we could see kernel information by task_tick() and scheduler_tick()



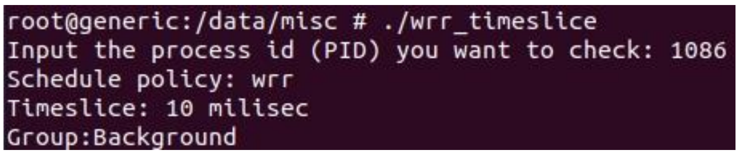
```
In task_tick_wrr: task_group: /
  task_tick: 1086 time_slice: 10
scheduler_tick: number:6, pid: 1086, wrr number  in the queue:1
In task_tick_wrr: task_group: /
  task_tick: 1086 time_slice: 9
scheduler_tick: number:6, pid: 1086, wrr number  in the queue:1
In task_tick_wrr: task_group: /
  task_tick: 1086 time_slice: 8
scheduler_tick: number:6, pid: 1086, wrr number  in the queue:1
In task_tick_wrr: task_group: /
  task_tick: 1086 time_slice: 7
scheduler_tick: number:6, pid: 1086, wrr number  in the queue:1
In task_tick_wrr: task_group: /
  task_tick: 1086 time_slice: 6
scheduler_tick: number:6, pid: 1086, wrr number  in the queue:1
In task_tick_wrr: task_group: /
  task_tick: 1086 time_slice: 5
scheduler_tick: number:6, pid: 1086, wrr number  in the queue:1
In task_tick_wrr: task_group: /
  task_tick: 1086 time_slice: 4
scheduler_tick: number:6, pid: 1086, wrr number  in the queue:1
In task_tick_wrr: task_group: /
  task_tick: 1086 time_slice: 3
```

4.Then we click the home button to turn the process into background and check the kernel information



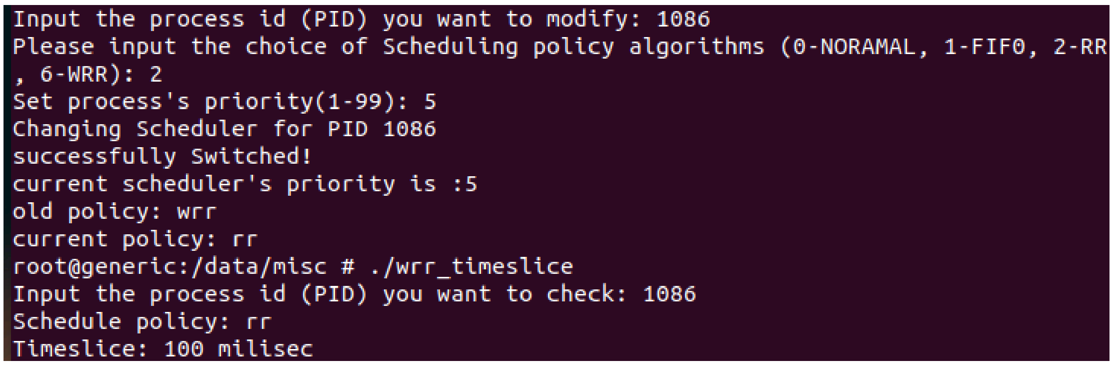
```
Change timeslice to background
healthd: battery l=50 v=0 t=0.0 h=2 st=2 chg=a
get_wrr_interval_Task_group: /bg_non_interactive
```

We use wrr_timeslice to check its state again, finding its timeslice shift to 10ms



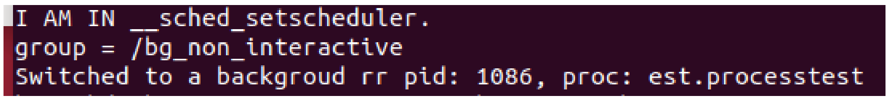
```
root@generic:/data/misc # ./wrr_timeslice
Input the process id (PID) you want to check: 1086
Schedule policy: wrr
Timeslice: 10 milisec
Group:Background
```

5.Besides, we could also use set_sched to change to other schedule policy,like RR which holds 100ms timeslice as well in the background.



```
Input the process id (PID) you want to modify: 1086
Please input the choice of Scheduling policy algorithms (0-NORAMAL, 1-FIFO, 2-RR
, 6-WRR): 2
Set process's priority(1-99): 5
Changing Scheduler for PID 1086
successfully Switched!
current scheduler's priority is :5
old policy: wrr
current policy: rr
root@generic:/data/misc # ./wrr_timeslice
Input the process id (PID) you want to check: 1086
Schedule policy: rr
Timeslice: 100 milisec
```

kernel information also indicates it's still in background.



```
I AM IN __sched_setscheduler.
group = /bg_non_interactive
Switched to a backgroud rr pid: 1086, proc: est.processtest
```

Eventually, we find our code could correctly change the schedule policy and could make change between background and foreground, achieving our basic goal.

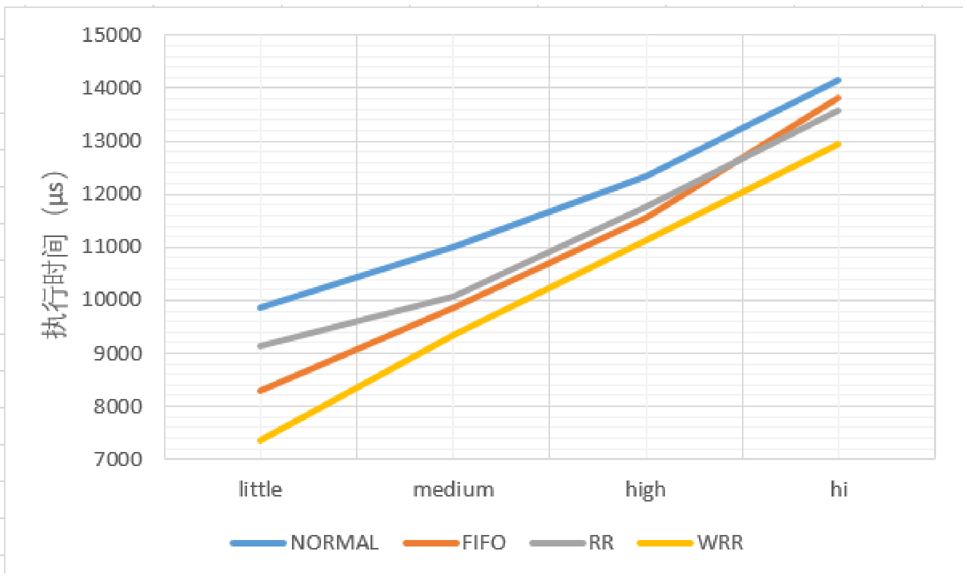
Extra bonus:

My additional work is about the comparison of Normal, FIFO, RR and WRR schedule policy.

We use function perf_compare() which contain 4 scales of subprocedures, choice of policy and priority(for WRR, we choose priority 0) to test different policies' performance. We fork 3 child threads, 4 threads will do the same thing. The time is between the last thread is forked and the root thread finish. We set wait(NULL) to get the correct order.（It might have error for every time run the same task, it'll get different time, so I choose the average time）

Finally ,we get a chart. I think because of running time is less then 100ms,we couldn't find evident difference between WRR, RR and FIFO. Normal will take a little longer for context switch. If we want to further study, we need to impl longer-time task.

running time(μs)	NORMAL	FIFO	RR	WRR
little	9865	8295	9132	7345
medium	11002	9875	10082	9353
high	12326	11548	11753	11143
hi	14157	13816	13589	12943



Obstacle and achievement:

At beginning of the project, I find it hard to handle not only because of its code quantity, but also its unfamiliarity, It's much harder than the first project. Thus, I have to read a lot of introduction and source codes to find solution. Thanks to reference of https://helix979.github.io/jkoo/post/os-scheduler/, I have a general idea of how a schedule policy works. Then I search use of unknown functions and follow rt.c. After hard and long work, I finally decide how the sched_wrr_entity is defined and finish wrr schedule policy.

Facing thousands of lines of codes, it's hard to carefully read the source code and understand how it works. But through this project, I learnt a lot about how the linux kernel is programmed and experience how to code in an OS kernel for another OS. It gave me impression about Android OS, which is undoubtedly beneficial.