

上海交通大学

School of Electronic Information and Electrical Engineering

Operating System Project 2

Title Android Scheduler

Name YuYe Zhu

Student Number 516030910148

Class No. CS359

Adivisor Fan Wu

目 录

| | |
|---------------------------------------|-----------|
| 1 Introduction | 1 |
| 1.1 Objectives | 1 |
| 1.2 Environment | 1 |
| 1.3 Compile the Linux Kernel | 1 |
| 2 Main WRR Program | 2 |
| 2.1 Sched Class Definiton | 2 |
| 2.2 Functions | 2 |
| 3 Modifications in other files | 6 |
| 3.1 rt.c | 6 |
| 3.2 core.c | 7 |
| 3.3 debug.c | 10 |
| 3.4 sched.h | 10 |
| 3.5 linux/sched.h | 11 |
| 4 Test Program | 12 |
| 4.1 System Call | 12 |
| 4.2 Test File | 13 |
| 5 Results | 15 |
| 5.1 Foreground | 15 |
| 5.2 Background | 15 |
| 6 Summary | 18 |
| A wr.c | 19 |

1 Introduction

1.1 Objectives

- Compile the Android kernel.
- Familiarize Android scheduler
- Implement a weighted round robin scheduler.
- Get experience with software engineering techniques.

1.2 Environment

- AVD(Android Virtual Devices) SDK version r24.4.1
- Development Linux (64-bits) Ubuntu(16.04)

1.3 Compile the Linux Kernel

Environment Variables

```

1 export JAVA_HOME=/usr/lib/jdk-9.0.4
2 export JRE_HOME=/usr/lib/jdk-9.0.4/jre
3 export CLASSPATH=.:$CLASSPATH:$JAVA_HOME/lib:$JRE_HOME/lib
4 export PATH=$PATH:$JAVA_HOME/bin:$JRE_HOME/bin
5 export PATH="/home/zyy/apps/android-ndk-linux":$PATH
6 export PATH="/home/zyy/apps/android-ndk-linux/tools":$PATH
7 export PATH="/home/zyy/apps/android-sdk-linux/platform-tools":$PATH
8 export PATH="/home/zyy/apps/android-ndk-linux/toolchains/
9     arm-linux-androideabi-4.9/prebuilt/linux-x86_64/bin":$PATH

```

Modify Makefile in the kernel

```

1 export KBUILD_BUILDHOST := $(SUBARCH)
2 ARCH      ?= arm
3 CROSS_COMPILE ?= arm-linux-androideabi-

```

Set the kernel

- Set Kernel hacking → Compile the kernel with debug info to be true
- Set Enable loadable module support → Forced module loading to be true
- Set Enable loadable module support → Module unloading to be true
- Set Enable loadable module support → Module unloading → Forced module unloading to be true

Compile the kernel

Type make -j* in the command line. The number of -j* depends on the number of cores in the system, where it should be make -j4.

2 Main WRR Program

2.1 Sched Class Definiton

```

1  const struct sched_class wrr_sched_class = {
2      .next      = &fair_sched_class,
3      .dequeue_task  = dequeue_task_wrr,
4      .enqueue_task  = enqueue_task_wrr,
5      .yield_task    = yield_task_wrr,
6      .check_preempt_curr = check_preempt_curr_wrr,
7      .pick_next_task  = pick_next_task_wrr,
8      .put_prev_task   = put_prev_task_wrr,
9
10     #ifdef CONFIG_SMP
11         .select_task_rq  = select_task_rq_wrr,
12         .rq_online      = rq_online_wrr,
13         .rq_offline     = rq_offline_wrr,
14         .task_woken     = task_woken_wrr,
15         .switched_from  = switched_from_wrr,
16         .pre_schedule   = pre_schedule_wrr,
17         .post_schedule  = post_schedule_wrr,
18     #endif
19
20     .set_curr_task  = set_curr_task_wrr,
21     .task_tick      = task_tick_wrr,
22
23     .get_rr_interval = get_rr_interval_wrr,
24
25     .prio_changed   = prio_changed_wrr,
26     .switched_to    = switched_to_wrr,
27 };

```

2.2 Functions

- enqueue_task_wrr

```

1  {
2      struct sched_wrr_entity *wrr_se;
3      struct wrr_rq *wrr_rq;
4      struct list_head *queue;
5
6      wrr_se = &p->wrr;
7      wrr_rq = &rq->wrr;
8      queue = &wrr_rq->wrr_rq_list;

```

```

9
10
11     list_add_tail(&wrr_se->run_list, queue);
12     wrr_rq->wrr_nr_running++;
13
14     inc_nr_running(rq);
15 }

```

- dequeue_task_wrr

```

1 static void dequeue_task_wrr(struct rq *rq, struct task_struct *p,
   int flags)
2 {
3     struct sched_wrr_entity *wrr_se;
4     struct wrr_rq *wrr_rq;
5
6     wrr_se = &p->wrr;
7     wrr_rq = &rq->wrr;
8
9     list_del_init(&wrr_se->run_list);
10    --wrr_rq->wrr_nr_running;
11    dec_nr_running(rq);
12 }

```

- yield_task_wrr

```

1 static void yield_task_wrr(struct rq *rq)
2 {
3     requeue_task_wrr(rq, rq->curr, 0);
4 }

```

```

1 static void requeue_task_wrr(struct rq *rq, struct task_struct *p,
   int flags)
2 {
3     struct sched_wrr_entity *wrr_se;
4     struct wrr_rq *wrr_rq;
5     struct list_head *head;
6
7     wrr_se = &p->wrr;
8     wrr_rq = &rq->wrr;
9     head = &wrr_rq->wrr_rq_list;
10
11     if (sizeof(wrr_rq->wrr_nr_running) == 1)
12         return;

```

```

13
14     list_move_tail(&wrr_se->run_list, head);
15 }

```

- pick_next_task_wrr

```

1 static struct task_struct *pick_next_task_wrr(struct rq *rq)
2 {
3     struct sched_wrr_entity *wrr_se;
4     struct task_struct *p;
5     struct wrr_rq *wrr_rq;
6     struct list_head *queue;
7
8     wrr_rq = &rq->wrr;
9
10    if (!wrr_rq->wrr_nr_running)
11        return NULL;
12
13    queue = &wrr_rq->wrr_rq_list;
14    wrr_se = list_entry(queue->next, struct sched_wrr_entity,
15                        run_list);
16    return container_of(wrr_se, struct task_struct, wrr);\

```

- check_preempt_curr_wrr

```

1 static void check_preempt_curr_wrr(struct rq *rq,
2                                     struct task_struct *p, int flags)
3 {
4     if (p->prio < rq->curr->prio) {
5         resched_task(rq->curr);
6         return;
7     }
8 }

```

- put_prev_task_wrr

```

1 static void put_prev_task_wrr(struct rq *rq, struct task_struct
2                               *prev) { }

```

- task_tick_wrr

Note that the function `task_group_path` tells whether a process is foreground and background. For foreground processes, it returns `"/"`. For background processes it returns `"/bg_non_interactive"`

```

1 static void task_tick_wrr(struct rq *rq, struct task_struct *p, int
   queued)
2 {
3     printk("task_tick_wrr pid:%d timeslice:
         %d\n",p->pid,p->wrr.time_slice);
4     struct sched_wrr_entity *wrr_se;
5
6     wrr_se = &p->wrr;
7     if (--p->wrr.time_slice)
8         return;
9     //char *tmp=task_group_path(p->sched_task_group);
10
11     //printk("tmp: %s",tmp);
12     //printk("task_group_path:
         %s",task_group_path(p->sched_task_group));
13
14     if(task_group_path(p->sched_task_group) [1] != 'b')
15     {
16         printk("WRR Fore Ground\n");
17         p->wrr.time_slice = WRR_TIMESLICE_FG;
18     }
19     else
20     {
21         printk("WRR Back Ground\n");
22         p->wrr.time_slice = WRR_TIMESLICE_BG;
23     }
24
25     if (wrr_se->run_list.prev != wrr_se->run_list.next) {
26         requeue_task_wrr(rq, p, 0);
27         set_tsk_need_resched(p);
28         return;
29     }
30 }

```

- switched_to_wrr

```

1 static void switched_to_wrr(struct rq *rq, struct task_struct *p)
2 {
3     struct sched_wrr_entity *wrr_se;
4
5     wrr_se = &p->wrr;
6     if(task_group_path(p->sched_task_group) [1] != 'b'
7         p->wrr.time_slice = WRR_TIMESLICE_FG;
8

```

```

9     else
10         p->wrr.time_slice = WRR_TIMESLICE_BG;
11
12     INIT_LIST_HEAD(&wrr_se->run_list);
13 }

```

- init_wrr_rq

```

1 void init_wrr_rq(struct wrr_rq *wrr_rq, struct rq *rq)
2 {
3     wrr_rq->wrr_nr_running = 0;
4     INIT_LIST_HEAD(&wrr_rq->wrr_rq_list);
5 }

```

- get_rr_interval_wrr

```

1 {
2     if (p == NULL)
3         return -EINVAL;
4
5     if(task_group_path(p->sched_task_group) [1] != 'b')
6         return WRR_TIMESLICE_FG;
7     else
8         return WRR_TIMESLICE_BG;
9 }

```

3 Modifications in other files

3.1 rt.c

Modify .next to wrr_sched_class, otherwise wrr.c cannot get to work.

```

1 const struct sched_class rt_sched_class = {
2     .next      = &wrr_sched_class,
3     .enqueue_task  = enqueue_task_rt,
4     .dequeue_task  = dequeue_task_rt,
5     .yield_task   = yield_task_rt,
6
7     .check_preempt_curr = check_preempt_curr_rt,
8
9     .pick_next_task   = pick_next_task_rt,
10    .put_prev_task    = put_prev_task_rt,
11    ....
12 }

```


3.2 core.c

- Add INIT_LIST_HEAD for WRR in __sched_fork

```

1 static void __sched_fork(struct task_struct *p)
2 {
3     ....
4
5     INIT_LIST_HEAD(&p->rt.run_list);
6
7     INIT_LIST_HEAD(&p->wrr.run_list);
8
9     ...
10 }
```

- Modify sched_fork

We need to tell whether a task belongs to wrr_sched_class.

```

1 void sched_fork(struct task_struct *p)
2 {
3     ...
4
5     if(rt_prio(p->prio))
6     {
7         if (p->policy == SCHED_WRR)
8             p->sched_class = &wrr_sched_class; //wrr
9         else
10            p->sched_class = &rt_sched_class;
11     }
12     else
13         p->sched_class = &fair_sched_class;
14
15     ...
16 }
```

- Modify __setscheduler

Same as sched_fork.

```

1 static void
2 __setscheduler(struct rq *rq, struct task_struct *p, int policy, int
   prio)
3 {
4     ...
5
6     if(rt_prio(p->prio))
```

```

7  {
8      if (p->policy == SCHED_WRR)
9          p->sched_class = &wrr_sched_class;
10     else
11         p->sched_class = &rt_sched_class;
12 }
13
14 ...
15 }

```

- Add some printk information in function `__sched_setscheduler`

```

1  static int __sched_setscheduler(struct task_struct *p, int policy,
2                                const struct sched_param *param, bool user)
3  {
4      ....
5      printk("I AM IN __SCHED_SETSCHEDULER\n");
6      printk("group=");
7      printk("%s\n", task_group_path(p->sched_task_group));
8
9      switch(policy)
10     {
11         case 0:
12             printk("NORMAL entity,");break;
13         case 1:
14             printk("FIFO entity,");break;
15         case 2:
16             printk("RR entity,");break;
17         case 6:
18             printk("WRR entity,");break;
19     }
20
21     if(task_group_path(p->sched_task_group) [1] != 'b')
22         printk("foregroup\t");
23     else
24         printk("backgroup\t");
25     printk("pid=%d, ", p->pid);
26     printk("proc=est.processtest,\n");
27     return 0;
28 }

```

- Add `init_wrr_rq`

```

1  for_each_possible_cpu(i) {
2      struct rq *rq;

```

```

3     ...
4     init_cfs_rq(&rq->cfs);
5     init_rt_rq(&rq->rt, rq);
6     init_wrr_rq(&rq->wrr, rq); //wrr
7     ...
8 }

```

- Modify System Call 159: sched_get_priority_max

```

1 SYSCALL_DEFINE1(sched_get_priority_max, int, policy)
2 {
3     int ret = -EINVAL;
4
5     switch (policy) {
6     case SCHED_FIFO:
7     case SCHED_RR:
8         ret = MAX_USER_RT_PRIO-1;
9         break;
10    case SCHED_NORMAL:
11    case SCHED_BATCH:
12    case SCHED_IDLE:
13        ret = 0;
14    case SCHED_WRR:
15        ret = MAX_USER_WRR_PRIO-1;
16        break;
17    }
18    return ret;
19 }

```

- Modify System Call 160: sched_get_priority_min

```

1 SYSCALL_DEFINE1(sched_get_priority_min, int, policy)
2 {
3     int ret = -EINVAL;
4
5     switch (policy) {
6     case SCHED_FIFO:
7     case SCHED_RR:
8         ret = 1;
9         break;
10    case SCHED_NORMAL:
11    case SCHED_BATCH:
12    case SCHED_IDLE:
13        ret = 0;
14        break;

```

```

15     case SCHED_WRR:
16         ret = 1;
17         break;
18     }
19     return ret;
20 }

```

- Other definitions

```

1 int sysctl_sched_wrr_runtime = 950000;
2 unsigned int sysctl_sched_wrr_period = 1000000;

```

3.3 debug.c

- Remove "static" from the definition of function task_group_path so that task_group_path can be utilized as an external function.

```

1 char group_path[PATH_MAX];
2
3 char *task_group_path(struct task_group *tg)
4 {
5     ....
6 }

```

3.4 sched.h

- Declare a wrr_rq struct

```

1 struct cfs_rq;
2 struct rt_rq;
3 struct wrr_rq;

```

- Define a new struct wrr_rq

```

1 struct wrr_rq {
2     unsigned long wrr_nr_running;
3     struct list_head wrr_rq_list;
4     struct task_struct* curr;
5     raw_spinlock_t wrr_rq_lock;
6 };

```

- Add a wrr_rq variable and a list_head variable and to struct rq

```

1 struct cfs_rq cfs;
2 struct rt_rq rt;

```

```

3     struct wrr_rq wrr;
4
5 #ifdef CONFIG_RT_GROUP_SCHED
6     struct list_head leaf_rt_rq_list;
7 #endif
8
9 #ifdef CONFIG_WRR_GROUP_SCHED
10    struct list_head leaf_wrr_rq_list;
11 #endif

```

- Declare some extern variables and functions

```

1 extern const struct sched_class wrr_sched_class;
2
3 extern void init_sched_rt_class(void);
4 extern void init_sched_fair_class(void);
5 extern void init_sched_wrr_class(void);
6
7 extern void print_cfs_stats(struct seq_file *m, int cpu);
8 extern void print_rt_stats(struct seq_file *m, int cpu);
9 extern void print_wrr_stats(struct seq_file *m, int cpu);
10
11 extern void init_cfs_rq(struct cfs_rq *cfs_rq);
12 extern void init_rt_rq(struct rt_rq *rt_rq, struct rq *rq);
13 extern void init_wrr_rq(struct wrr_rq *wrr_rq, struct rq *rq);

```

3.5 linux/sched.h

- Define SCHED_WRR

```

1 #define SCHED_NORMAL 0
2 #define SCHED_FIFO 1
3 #define SCHED_RR 2
4 #define SCHED_BATCH 3
5 /* SCHED_ISO: reserved but not implemented yet */
6 #define SCHED_IDLE 5
7 #define SCHED_WRR 6

```

- Define sched_wrr_entity

```

1 struct sched_wrr_entity {
2     struct list_head run_list;
3     unsigned long timeout;
4     unsigned int time_slice;
5     int nr_cpus_allowed;

```

```

6
7     struct sched_wrr_entity *back;
8 #ifdef CONFIG_WRR_GROUP_SCHED
9     struct sched_wrr_entity *parent;
10    /* rq on which this entity is (to be) queued: */
11    struct wrr_rq    *wrr_rq;
12    /* rq "owned" by this entity/group: */
13    struct wrr_rq    *my_q;
14 #endif
15 };

```

- Define time slice for WRR

```

1 #define RR_TIMESLICE  (100 * HZ / 1000) //100ms
2
3 #define WRR_TIMESLICE_FG  (100 * HZ / 1000) //foreground 100ms
4 #define WRR_TIMESLICE_BG  (10 * HZ / 1000) //background 10ms

```

- Add a sched_wrr_entity variable to task_struct

```

1 struct task_struct {
2     ....
3     unsigned int rt_priority;
4     unsigned int wrr_priority;
5     const struct sched_class *sched_class;
6     struct sched_entity se;
7     struct sched_rt_entity rt;
8     struct sched_wrr_entity wrr;
9     ...
10 }

```

- Declare a wrr_rq struct

```

1 struct seq_file;
2 struct cfs_rq;
3 struct wrr_rq;
4 struct task_group;

```

4 Test Program

4.1 System Call

Given Kernel file calls.S and in syscalls.h, we get the following system calls:

- System call 156: CALL(sys_sched_setscheduler) 156

```

1 asmlinkage long sys_sched_setscheduler(pid_t pid, int policy,
2      struct sched_param __user *param);

```

- System call 157: CALL(sys_sched_getscheduler)

```

1 asmlinkage long sys_sched_getscheduler(pid_t pid);

```

- System call 159: CALL(sys_sched_get_priority_max)

```

1 asmlinkage long sys_sched_get_priority_max(int policy);

```

- System call 160: CALL(sys_sched_get_priority_min)

```

1 asmlinkage long sys_sched_get_priority_min(int policy);

```

4.2 Test File

With the help of these system calls, I am able to change the scheduler. My test file is shown as follows:

```

1 #include <stdio.h>
2 #include <unistd.h>
3 #include <stdlib.h>
4 #include <sys/time.h>
5 #include <sys/resource.h>
6 #include <sched.h>
7
8 #include <linux/sched.h>
9
10 #define SCHED_WRR 6
11
12 int main(void)
13 {
14     struct sched_param pp;
15
16     int prev,policy,pid,prio,prio_min,prio_max,flag=1;
17
18     while(flag){
19         printf("Please input the Choice of Scheduling algorithms
20             (0-NORMAL,1-FIFO,2-RR,6-WRR):");
21         scanf("%d",&policy);
22         printf("Current scheduling algorithm is ");
23         switch(policy) {

```

```

24     printf("SCHED_NORMAL\n");
25     flag=0;
26     break;
27 case 1:
28     printf("SCHED_FIFO\n");
29     flag=0;
30     break;
31 case 2:
32     printf("SCHED_RR\n");
33     flag=0;
34     break;
35 case 6:
36     printf("SCHED_WRR\n");
37     flag=0;
38     break;
39 }
40 if(flag)
41     printf("Invalid input! Please input policy again!\n");
42 }
43
44 printf("Please input the id of the testprocess :");
45 scanf("%d",&pid);
46
47 prio_min=syscall(160,SCHED_WRR);
48 prio_max=syscall(159,SCHED_WRR);
49 printf("Set Process's priority (%d-%d) :",prio_min,prio_max);
50 scanf("%d",&prio);
51
52 printf("current scheduler's priority is : %d\n",prio);
53 prev=syscall(157,pid); //get scheduler
54
55 printf("pre scheduler:");
56
57 switch(prev) {
58 case 0:
59     printf("SCHED_NORMAL\n");
60     break;
61 case 1:
62     printf("SCHED_FIFO\n");
63     break;
64 case 2:
65     printf("SCHED_RR\n");
66     break;
67 case 6:

```



```
68     printf("SCHED_WRR\n");
69     break;
70 }
71
72
73 pp.sched_priority=prio;
74
75 prev=syscall(156,pid,policy,&pp); //set scheduler & priority
76 printf("set scheduler result: %d\n",prev);
77
78 prev=syscall(157,pid); //get scheduler
79 printf("current scheduler: %d\n", prev);
80
81 return 0;
82 }
```

5 Results

5.1 Foreground

For foreground process, we observe that processtest.apk continuously executes and corresponding timeslice is 10.

5.2 Background

For background process, we observe that processtest.apk rarely executes and corresponding timeslice is 1.

```

zyy@zyy-Lenovo-Rescuer-15ISK: ~/apps/kernel/goldfish
u0_a29    1038  79    497620 27840 20    0    0    0    sys_epoll_ b3022478 S
com.android.exchange
u0_a53    1072  79    502668 43880 20    0    0    0    sys_epoll_ b3022478 S
com.osprj.test.processtest
root      1086  68    3020   1156 20    0    0    0    sys_rt_sig aa4b0e18 S
/system/bin/sh
root      1090  1086  2700   1036 20    0    0    0           0 ae6e190c R
ps
root@generic:/ # cd data/misc/
root@generic:/data/misc # ./testwrr
Please input the Choice of Scheduling algorithms (0-NORMAL,1-FIFO,2-RR,6-WRR):6
Current scheduling algorithm is SCHED_WRR
Please input the id of the testprocess :1072
Set Process's priority (1-99) :60
current scheduler's priority is : 60
pre scheduler:SCHED_NORMAL
set scheduler result: 0
current scheduler: 6
root@generic:/data/misc #

zyy@zyy-Lenovo-Rescuer-15ISK: ~/apps/android-sdk-linux/tools
I AM IN __SCHED_SETSCHEDULER
group=/
WRR entity,foregroup  pid=1072, proc=est.processtest,
task_tick_wrr pid:1072 timeslice: 10
task_tick_wrr pid:1072 timeslice: 9
task_tick_wrr pid:1072 timeslice: 8
task_tick_wrr pid:1072 timeslice: 7
task_tick_wrr pid:1072 timeslice: 6
task_tick_wrr pid:1072 timeslice: 5
task_tick_wrr pid:1072 timeslice: 4
task_tick_wrr pid:1072 timeslice: 3
task_tick_wrr pid:1072 timeslice: 2
task_tick_wrr pid:1072 timeslice: 1
WRR Fore Ground
task_tick_wrr pid:1072 timeslice: 10
task_tick_wrr pid:1072 timeslice: 9
task_tick_wrr pid:1072 timeslice: 8
task_tick_wrr pid:1072 timeslice: 7
task_tick_wrr pid:1072 timeslice: 6
task_tick_rt 60 timeslice: 10
task_tick_wrr pid:1072 timeslice: 5
task_tick_wrr pid:1072 timeslice: 4
task_tick_wrr pid:1072 timeslice: 3
task_tick_wrr pid:1072 timeslice: 2
task_tick_wrr pid:1072 timeslice: 1

```

图 1: Foreground WRR

[illegible]

图 2: Background WRR

6 Summary

Comparing two figures above, it is convincing that whether a process is foreground or not matters a lot in the scheduling algorithm. Given that the `processtest` rarely executes when background and that my algorithm does not allow preemption, it is rather difficult for us to directly observe what was happening when switching between foreground and background. To be specific, when I switch the process to background, it takes a long time to consume the foreground timeslice; when the process is switched to foreground, the background timeslice is immediately used up.

Generally speaking, this project is an arduous one but also a rewarding one. After reading codes in the kernel, I get to know how a scheduling algorithm works. Also, working in Linux enables me to get familiar with the command lines. When trying to write my own scheduling algorithm, I come across several tricky problems. A variable named `.next` in scheduling class is an example.

When I was first debugging, I am not aware that I should change the `.next` variable for other scheduling classes. As a result, the function named `task_tick_wrr()` in `wrr.c` never works. Unfortunately, it costs me several hours to realize it.

Besides, I have an idea that we can compare the performance by forking lots of processes. Recall what we have done in project one, we can set the scheduling policy to WRR through DFS algorithm. However, since I had no idea how to exactly measure the execution time for a process, I did not finish it.

A wrr.c

```

1  #include "sched.h"
2  #include <linux/sched.h>
3
4  extern char *task_group_path(struct task_group *tg);
5
6  static void dequeue_task_wrr(struct rq *rq, struct task_struct *p, int
       flags)
7  {
8      struct sched_wrr_entity *wrr_se;
9      struct wrr_rq *wrr_rq;
10
11     wrr_se = &p->wrr;
12     wrr_rq = &rq->wrr;
13
14     list_del_init(&wrr_se->run_list);
15     --wrr_rq->wrr_nr_running;
16     dec_nr_running(rq);
17 }
18
19
20
21 static void enqueue_task_wrr(struct rq *rq, struct task_struct *p, int
       flags)
22 {
23     struct sched_wrr_entity *wrr_se;
24     struct wrr_rq *wrr_rq;
25     struct list_head *queue;
26
27     wrr_se = &p->wrr;
28     wrr_rq = &rq->wrr;
29     queue = &wrr_rq->wrr_rq_list;
30
31
32     list_add_tail(&wrr_se->run_list, queue);
33     wrr_rq->wrr_nr_running++;
34
35     inc_nr_running(rq);
36     //printk("enqueue_task_wrr 2 %d\n",p->pid);
37 }
38
39 static void requeue_task_wrr(struct rq *rq, struct task_struct *p, int

```

```

    flags)
40 {
41     //printfk("requeue_task_wrr 1 %d\n",p->pid);
42     struct sched_wrr_entity *wrr_se;
43     struct wrr_rq *wrr_rq;
44     struct list_head *head;
45
46     wrr_se = &p->wrr;
47     wrr_rq = &rq->wrr;
48     head = &wrr_rq->wrr_rq_list;
49
50     if (sizeof(wrr_rq->wrr_nr_running) == 1)
51         return;
52
53     list_move_tail(&wrr_se->run_list, head);
54
55     //printfk("requeue_task_wrr 1 %d\n",p->pid);
56 }
57
58 static void yield_task_wrr(struct rq *rq)
59 {
60     requeue_task_wrr(rq, rq->curr, 0);
61 }
62
63 static struct task_struct *pick_next_task_wrr(struct rq *rq)
64 {
65     //printfk("pick_next_task_wrr 1\n");
66     struct sched_wrr_entity *wrr_se;
67     struct task_struct *p;
68     struct wrr_rq *wrr_rq;
69     struct list_head *queue;
70
71     wrr_rq = &rq->wrr;
72
73     if (!wrr_rq->wrr_nr_running)
74         return NULL;
75
76     queue = &wrr_rq->wrr_rq_list;
77     wrr_se = list_entry(queue->next, struct sched_wrr_entity, run_list);
78     return container_of(wrr_se, struct task_struct, wrr);
79     //printfk("pick_next_task_wrr 2\n");
80 }
81
82 static void put_prev_task_wrr(struct rq *rq, struct task_struct *prev)

```

```

83 {
84
85 }
86
87 static void task_tick_wrr(struct rq *rq, struct task_struct *p, int queued)
88 {
89     printk("task_tick_wrr pid:%d timeslice: %d\n",p->pid,p->wrr.time_slice);
90     struct sched_wrr_entity *wrr_se;
91
92     wrr_se = &p->wrr;
93     if (--p->wrr.time_slice)
94         return;
95     //char *tmp=task_group_path(p->sched_task_group);
96
97     //printk("tmp: %s",tmp);
98     //printk("task_group_path: %s",task_group_path(p->sched_task_group));
99
100    if(task_group_path(p->sched_task_group) [1] != 'b')
101    {
102        printk("WRR Fore Ground\n");
103        p->wrr.time_slice = WRR_TIMESLICE_FG;
104    }
105    else
106    {
107        printk("WRR Back Ground\n");
108        p->wrr.time_slice = WRR_TIMESLICE_BG;
109    }
110
111    if (wrr_se->run_list.prev != wrr_se->run_list.next) {
112        requeue_task_wrr(rq, p, 0);
113        set_tsk_need_resched(p);
114        return;
115    }
116 }
117
118 static void set_curr_task_wrr(struct rq *rq)
119 {
120     struct task_struct *p;
121
122     p = rq->curr;
123     p->se.exec_start = rq->clock_task;
124
125 }
126

```

```

127 static void check_preempt_curr_wrr(struct rq *rq,
128                                   struct task_struct *p, int flags)
129 {
130     if (p->prio < rq->curr->prio) {
131         resched_task(rq->curr);
132         return;
133     }
134 }
135
136 static void switched_to_wrr(struct rq *rq, struct task_struct *p)
137 {
138     struct sched_wrr_entity *wrr_se;
139
140     wrr_se = &p->wrr;
141     if(task_group_path(p->sched_task_group) [1] != 'b')
142         p->wrr.time_slice = WRR_TIMESLICE_FG;
143
144     else
145         p->wrr.time_slice = WRR_TIMESLICE_BG;
146
147     INIT_LIST_HEAD(&wrr_se->run_list);
148 }
149
150 static void prio_changed_wrr(struct rq *rq, struct task_struct *p, int old)
151 { }
152
153
154 static unsigned int get_rr_interval_wrr(struct rq *rq, struct task_struct
155                                         *p)
156 {
157     if (p == NULL)
158         return -EINVAL;
159
160     if(task_group_path(p->sched_task_group) [1] != 'b')
161         return WRR_TIMESLICE_FG;
162     else
163         return WRR_TIMESLICE_BG;
164 }
165
166 void init_wrr_rq(struct wrr_rq *wrr_rq, struct rq *rq)
167 {
168     wrr_rq->wrr_nr_running = 0;
169     INIT_LIST_HEAD(&wrr_rq->wrr_rq_list);
170 }

```



```
170
171
172 const struct sched_class wrr_sched_class = {
173     .next      = &fair_sched_class,
174     .dequeue_task  = dequeue_task_wrr,
175     .enqueue_task  = enqueue_task_wrr,
176     .yield_task    = yield_task_wrr,
177     .check_preempt_curr = check_preempt_curr_wrr,
178     .pick_next_task  = pick_next_task_wrr,
179     .put_prev_task   = put_prev_task_wrr,
180
181     #ifdef CONFIG_SMP
182     .select_task_rq  = select_task_rq_wrr,
183     .rq_online       = rq_online_wrr,
184     .rq_offline      = rq_offline_wrr,
185     .task_woken      = task_woken_wrr,
186     .switched_from   = switched_from_wrr,
187     .pre_schedule    = pre_schedule_wrr,
188     .post_schedule   = post_schedule_wrr,
189     #endif
190
191     .set_curr_task   = set_curr_task_wrr,
192     .task_tick       = task_tick_wrr,
193
194     .get_rr_interval = get_rr_interval_wrr,
195
196     .prio_changed    = prio_changed_wrr,
197     .switched_to     = switched_to_wrr,
198 };
```