上海交通大學

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 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

```
export JAVA_HOME=/usr/lib/jdk-9.0.4
export JRE_HOME=/usr/lib/jdk-9.0.4/jre
export CLASSPATH=.:$CLASSPATH:$JAVA_HOME/lib:$JRE_HOME/lib
export PATH=$PATH:$JAVA_HOME/bin:$JRE_HOME/bin
export PATH="/home/zyy/apps/android-ndk-linux":$PATH
export PATH="/home/zyy/apps/android-ndk-linux/tools":$PATH
export PATH="/home/zyy/apps/android-sdk-linux/platform-tools":$PATH
export PATH="/home/zyy/apps/android-ndk-linux/toolchains/
      arm-linux-androideabi-4.9/prebuilt/linux-x86_64/bin":$PATH
```

Modify Makefile in the kernel

```
export KBUILD_BUILDHOST := $(SUBARCH)
ARCH    ?= arm
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

```
const struct sched_class wrr_sched_class = {
    .next       = &fair_sched_class,
    .dequeue_task   = dequeue_task_wrr,
    .enqueue_task   = enqueue_task_wrr,
    .yield_task   = yield_task_wrr,
    .check_preempt_curr = check_preempt_curr_wrr,
    .pick_next_task    = pick_next_task_wrr,
    .put_prev_task   = put_prev_task_wrr,

#ifdef CONFIG_SMP
    .select_task_rq    = select_task_rq_wrr,
    .rq_online    = rq_online_wrr,
    .rq_offline    = rq_offline_wrr,
    .task_woken    = task_woken_wrr,
    .switched_from    = switched_from_wrr,
    .pre_schedule    = pre_schedule_wrr,
    .post_schedule   = post_schedule_wrr,
#endif

    .set_curr_task   = set_curr_task_wrr,
    .task_tick    = task_tick_wrr,

    .get_rr_interval = get_rr_interval_wrr,

    .prio_changed    = prio_changed_wrr,
    .switched_to     = switched_to_wrr,
};
```

2.2 Functions

- enqueue_task_wrr

```
{
    struct sched_wrr_entity *wrr_se;
    struct wrr_rq *wrr_rq;
    struct list_head *queue;

    wrr_se = &p->wrr;
    wrr_rq = &rq->wrr;
    queue = &wrr_rq->wrr_rq_list;
```

```


  list_add_tail(&wrr_se->run_list, queue);
  wrr_rq->wrr_nr_running++;

  inc_nr_running(rq);
}
```

- dequeue_task_wrr

```
static void dequeue_task_wrr(struct rq *rq, struct task_struct *p,
    int flags)
{
  struct sched_wrr_entity *wrr_se;
  struct wrr_rq *wrr_rq;

  wrr_se = &p->wrr;
  wrr_rq = &rq->wrr;

  list_del_init(&wrr_se->run_list);
  --wrr_rq->wrr_nr_running;
  dec_nr_running(rq);
}
```

- yield_task_wrr

```
static void yield_task_wrr(struct rq *rq)
{
  requeue_task_wrr(rq, rq->curr , 0);
}
```

```
static void requeue_task_wrr(struct rq *rq, struct task_struct *p,
    int flags)
{
  struct sched_wrr_entity *wrr_se;
  struct wrr_rq *wrr_rq;
  struct list_head *head;

  wrr_se = &p->wrr;
  wrr_rq = &rq->wrr;
  head = &wrr_rq->wrr_rq_list;

  if (sizeof(wrr_rq->wrr_nr_running) == 1)
    return;
```

```

    list_move_tail(&wrr_se->run_list, head);
}
```

- pick_next_task_wrr

```
static struct task_struct *pick_next_task_wrr(struct rq *rq)
{
    struct sched_wrr_entity *wrr_se;
    struct task_struct *p;
    struct wrr_rq *wrr_rq;
    struct list_head *queue;

    wrr_rq = &rq->wrr;

    if (!wrr_rq->wrr_nr_running)
        return NULL;

    queue = &wrr_rq->wrr_rq_list;
    wrr_se = list_entry(queue->next, struct sched_wrr_entity,
        run_list);
    return container_of(wrr_se, struct task_struct, wrr);\
}
```

- check_preempt_curr_wrr

```
static void check_preempt_curr_wrr(struct rq *rq,
            struct task_struct *p, int flags)
{
    if (p->prio < rq->curr->prio) {
        resched_task(rq->curr);
        return;
    }
}
```

- put_prev_task_wrr

```
static void put_prev_task_wrr(struct rq *rq, struct task_struct
    *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”

```
static void task_tick_wrr(struct rq *rq, struct task_struct *p, int
    queued)
{
   printk("task_tick_wrr pid:%d timeslice:
       %d\n",p->pid,p->wrr.time_slice);
   struct sched_wrr_entity *wrr_se;

   wrr_se = &p->wrr;
   if (--p->wrr.time_slice)
      return;
   //char *tmp=task_group_path(p->sched_task_group);

   //printk("tmp: %s",tmp);
   //printk("task_group_path:
       %s",task_group_path(p->sched_task_group));

   if(task_group_path(p->sched_task_group) [1] != 'b')
   {
      printk("WRR Fore Ground\n");
      p->wrr.time_slice = WRR_TIMESLICE_FG;
   }
   else
   {
      printk("WRR Back Ground\n");
      p->wrr.time_slice = WRR_TIMESLICE_BG;
   }

   if (wrr_se->run_list.prev != wrr_se->run_list.next) {
      requeue_task_wrr(rq, p, 0);
      set_tsk_need_resched(p);
      return;
   }
}
```

- switched_to_wrr

```
static void switched_to_wrr(struct rq *rq, struct task_struct *p)
{
   struct sched_wrr_entity *wrr_se;

   wrr_se = &p->wrr;
   if(task_group_path(p->sched_task_group) [1] != 'b'
      p->wrr.time_slice = WRR_TIMESLICE_FG;

```

```
  else
    p->wrr.time_slice = WRR_TIMESLICE_BG;

  INIT_LIST_HEAD(&wrr_se->run_list);
}
```

- init_wrr_rq

```
void init_wrr_rq(struct wrr_rq *wrr_rq, struct rq *rq)
{
  wrr_rq->wrr_nr_running = 0;
  INIT_LIST_HEAD(&wrr_rq->wrr_rq_list);
}
```

- get_rr_interval_wrr

```
{
  if (p == NULL)
    return -EINVAL;

  if(task_group_path(p->sched_task_group) [1] != 'b')
    return WRR_TIMESLICE_FG;
  else
    return WRR_TIMESLICE_BG;
}
```

3 Modifications in other files

3.1 rt.c

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

```
const struct sched_class rt_sched_class = {
   .next      = &wrr_sched_class,
   .enqueue_task   = enqueue_task_rt,
   .dequeue_task   = dequeue_task_rt,
   .yield_task   = yield_task_rt,

   .check_preempt_curr = check_preempt_curr_rt,

   .pick_next_task    = pick_next_task_rt,
   .put_prev_task   = put_prev_task_rt,
   ....
}
```

3.2 core.c

- Add INIT_LIST_HEAD for WRR in ___sched_fork

```
static void __sched_fork(struct task_struct *p)
{
	....

	INIT_LIST_HEAD(&p->rt.run_list);

	INIT_LIST_HEAD(&p->wrr.run_list);

	...
}
```

- Modify sched_fork

  We need to tell whether a task belongs to wrr_sched_class.

```
void sched_fork(struct task_struct *p)
{
	...

	if(rt_prio(p->prio))
	{
		if (p->policy == SCHED_WRR)
			p->sched_class = &wrr_sched_class; //wrr
		else
			p->sched_class = &rt_sched_class;
	}
	else
		p->sched_class = &fair_sched_class;

	...
}
```

- Modify ___setscheduler

  Same as sched_fork.

```
static void
__setscheduler(struct rq *rq, struct task_struct *p, int policy, int
    prio)
{
	...

	if(rt_prio(p->prio))
```

```
	{
		if (p->policy == SCHED_WRR)
			p->sched_class = &wrr_sched_class;
		else
			p->sched_class = &rt_sched_class;
	}

	...
}
```

- Add some printk information in function ___sched_setscheduler

```
static int __sched_setscheduler(struct task_struct *p, int policy,
				const struct sched_param *param, bool user)
{
	....
	printk("I AM IN __SCHED_SETSCHEDULER\n");
	printk("group=");
	printk("%s\n",task_group_path(p->sched_task_group));

	switch(policy)
	{
		case 0:
			printk("NORMAL entity,");break;
		case 1:
			printk("FIFO entity,");break;
		case 2:
			printk("RR entity,");break;
		case 6:
			printk("WRR entity,");break;
	}

	if(task_group_path(p->sched_task_group) [1] != 'b')
		printk("foregroup\t");
	else
		printk("backgroup\t");
	printk("pid=%d, ",p->pid);
	printk("proc=est.processtest,\n");
	return 0;
}
```

- Add init_wrr_rq

```
	for_each_possible_cpu(i) {
		struct rq *rq;
```

```
    ...
    init_cfs_rq(&rq->cfs);
    init_rt_rq(&rq->rt, rq);
    init_wrr_rq(&rq->wrr, rq); //wrr
    ...
  }
```

- Modify System Call 159: sched_get_priority_max

```
SYSCALL_DEFINE1(sched_get_priority_max, int, policy)
{
  int ret = -EINVAL;

  switch (policy) {
  case SCHED_FIFO:
  case SCHED_RR:
    ret = MAX_USER_RT_PRIO-1;
    break;
  case SCHED_NORMAL:
  case SCHED_BATCH:
  case SCHED_IDLE:
    ret = 0;
  case SCHED_WRR:
    ret = MAX_USER_WRR_PRIO-1;
    break;
  }
  return ret;
}
```

- Modify System Call 160: sched_get_priority_min

```
SYSCALL_DEFINE1(sched_get_priority_min, int, policy)
{
  int ret = -EINVAL;

  switch (policy) {
  case SCHED_FIFO:
  case SCHED_RR:
    ret = 1;
    break;
  case SCHED_NORMAL:
  case SCHED_BATCH:
  case SCHED_IDLE:
    ret = 0;
    break;
```

```
    case SCHED_WRR:
        ret = 1;
        break;
    }
    return ret;
}
```

- Other definitions

```
int sysctl_sched_wrr_runtime = 950000;
unsigned int sysctl_sched_wrr_period = 1000000;
```

3.3 debug.c

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

```
char group_path[PATH_MAX];

char *task_group_path(struct task_group *tg)
{
    ....
}
```

3.4 sched.h

- Declare a wrr_rq struct

```
struct cfs_rq;
struct rt_rq;
struct wrr_rq;
```

- Define a new struct wrr_rq

```
struct wrr_rq {
    unsigned long wrr_nr_running;
    struct list_head wrr_rq_list;
    struct task_struct* curr;
    raw_spinlock_t wrr_rq_lock;
};
```

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

```
    struct cfs_rq cfs;
    struct rt_rq rt;
```

```
  struct wrr_rq wrr;

#ifdef CONFIG_RT_GROUP_SCHED
  struct list_head leaf_rt_rq_list;
#endif

#ifdef CONFIG_WRR_GROUP_SCHED
  struct list_head leaf_wrr_rq_list;
#endif
```

- Declare some extern variables and functions

```
extern const struct sched_class wrr_sched_class;

extern void init_sched_rt_class(void);
extern void init_sched_fair_class(void);
extern void init_sched_wrr_class(void);

extern void print_cfs_stats(struct seq_file *m, int cpu);
extern void print_rt_stats(struct seq_file *m, int cpu);
extern void print_wrr_stats(struct seq_file *m, int cpu);

extern void init_cfs_rq(struct cfs_rq *cfs_rq);
extern void init_rt_rq(struct rt_rq *rt_rq, struct rq *rq);
extern void init_wrr_rq(struct wrr_rq *wrr_rq, struct rq *rq);
```

3.5 linux/sched.h

- Define SCHED_WRR

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

- Define sched_wrr_entity

```
struct sched_wrr_entity {
  struct list_head run_list;
  unsigned long timeout;
  unsigned int time_slice;
  int nr_cpus_allowed;
```

```

  struct sched_wrr_entity *back;
#ifdef CONFIG_WRR_GROUP_SCHED
  struct sched_wrr_entity *parent;
  /* rq on which this entity is (to be) queued: */
  struct wrr_rq   *wrr_rq;
  /* rq "owned" by this entity/group: */
  struct wrr_rq   *my_q;
#endif
};
```

- Define time slice for WRR

```
#define RR_TIMESLICE  (100 * HZ / 1000) //100ms

#define WRR_TIMESLICE_FG  (100 * HZ / 1000) //foreground 100ms
#define WRR_TIMESLICE_BG  (10 * HZ / 1000) //background 10ms
```

- Add a sched_wrr_entity varaible to task_struct

```
struct task_struct {
  ....
  unsigned int rt_priority;
  unsigned int wrr_priority;
  const struct sched_class *sched_class;
  struct sched_entity se;
  struct sched_rt_entity rt;
  struct sched_wrr_entity wrr;
  ...
}
```

- Declare a wrr_rq struct

```
struct seq_file;
struct cfs_rq;
struct wrr_rq;
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

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

- System call 157: CALL(sys_sched_getscheduler)

```
asmlinkage long sys_sched_getscheduler(pid_t pid);
```

- System call 159: CALL(sys_sched_get_priority_max)

```
asmlinkage long sys_sched_get_priority_max(int policy);
```

- System call 160: CALL(sys_sched_get_priority_min)

```
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:

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <sched.h>

#include <linux/sched.h>

#define SCHED_WRR 6

int main(void)
{
   struct sched_param pp;

   int prev,policy,pid,prio,prio_min,prio_max,flag=1;

   while(flag){
   printf("Please input the Choice of Scheduling algorithms
       (0-NORMAL,1-FIFO,2-RR,6-WRR):");
      scanf("%d",&policy);
      printf("Current scheduling algorithm is ");
      switch(policy) {
      case 0:
```

```
            printf("SCHED_NORMAL\n");
            flag=0;
            break;
        case 1:
            printf("SCHED_FIFO\n");
            flag=0;
            break;
        case 2:
            printf("SCHED_RR\n");
            flag=0;
            break;
        case 6:
            printf("SCHED_WRR\n");
            flag=0;
            break;
        }
        if(flag)
            printf("Invalid input! Please input policy again!\n");
    }

    printf("Please input the id of the testprocess :");
    scanf("%d",&pid);

    prio_min=syscall(160,SCHED_WRR);
    prio_max=syscall(159,SCHED_WRR);
    printf("Set Process's priority (%d-%d) :",prio_min,prio_max);
    scanf("%d",&prio);

    printf("current scheduler's priority is : %d\n",prio);
    prev=syscall(157,pid); //get scheduler

    printf("pre scheduler:");

    switch(prev) {
        case 0:
            printf("SCHED_NORMAL\n");
            break;
        case 1:
            printf("SCHED_FIFO\n");
            break;
        case 2:
            printf("SCHED_RR\n");
            break;
        case 6:
```

```
        printf("SCHED_WRR\n");
        break;
    }


    pp.sched_priority=prio;

    prev=syscall(156,pid,policy,&pp); //set scheduler & priority
    printf("set scheduler result: %d\n",prev);

    prev=syscall(157,pid); //get scheduler
    printf("current scheduler: %d\n", prev);

    return 0;
}
```

5 Results

5.1 Foreground

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

5.2 Background

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

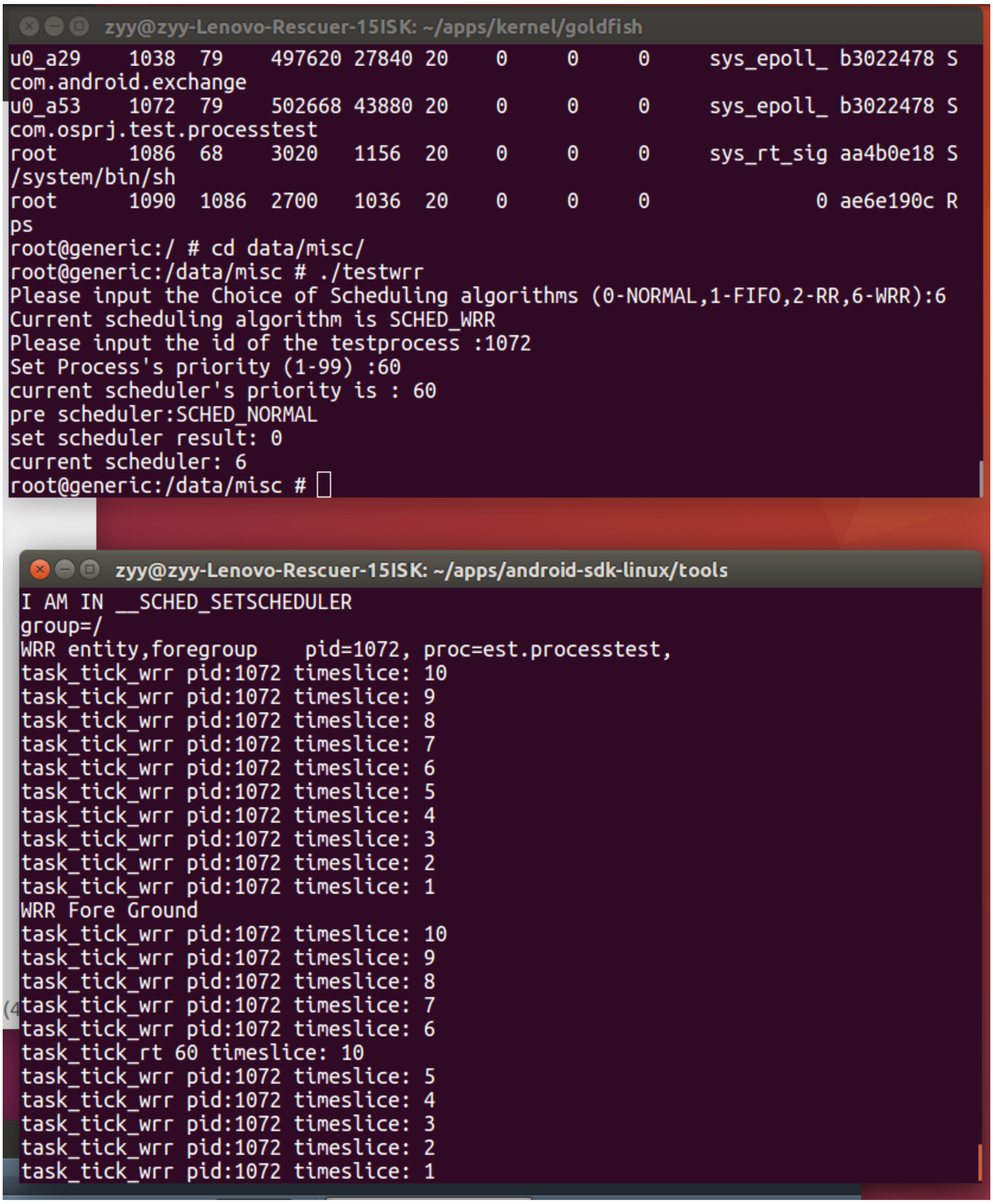

图 1: Foreground WRR

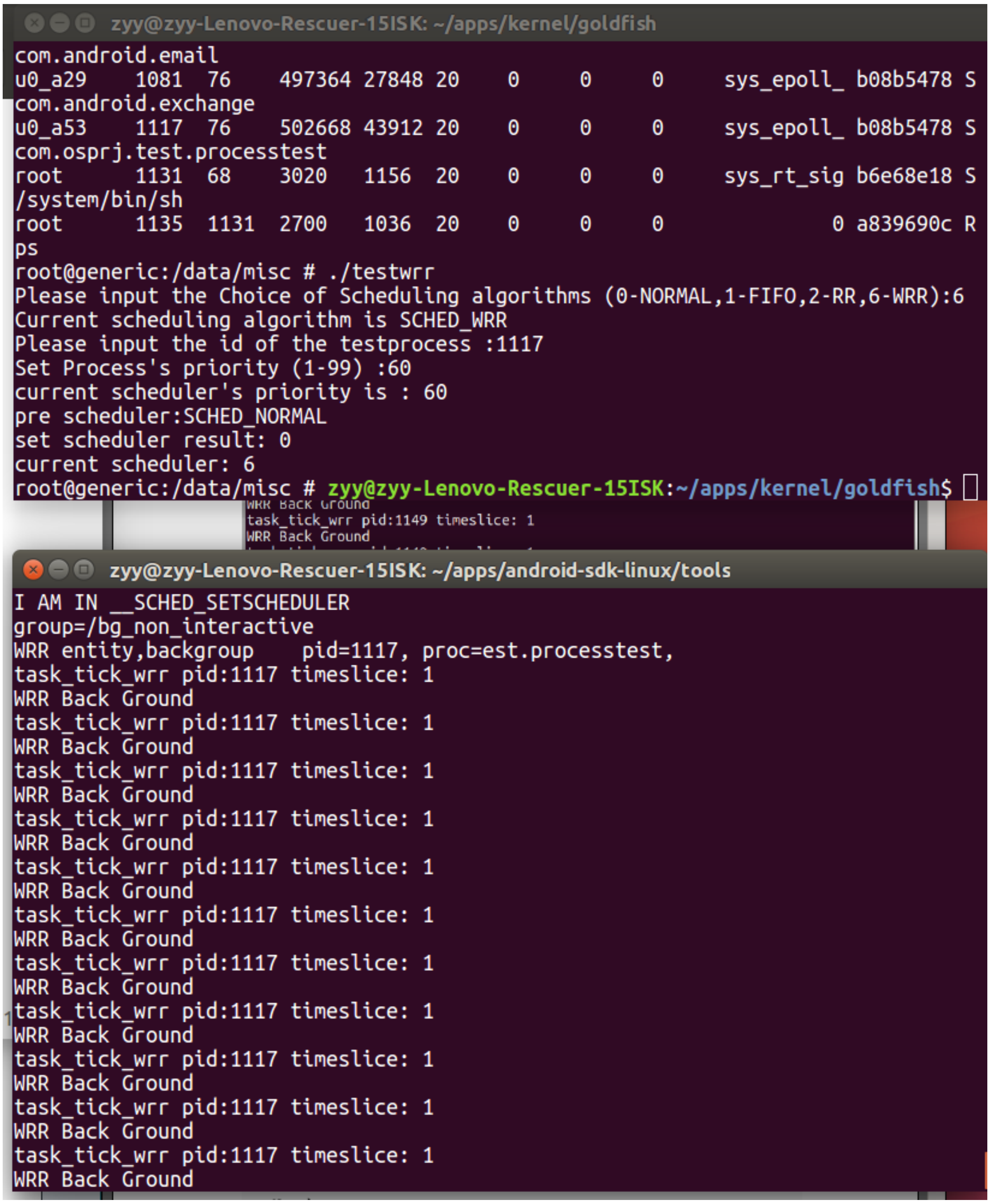

图 2: Background WRR

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

```
#include "sched.h"
#include <linux/sched.h>

extern char *task_group_path(struct task_group *tg);

static void dequeue_task_wrr(struct rq *rq, struct task_struct *p, int
    flags)
{
  struct sched_wrr_entity *wrr_se;
  struct wrr_rq *wrr_rq;

  wrr_se = &p->wrr;
  wrr_rq = &rq->wrr;

  list_del_init(&wrr_se->run_list);
  --wrr_rq->wrr_nr_running;
  dec_nr_running(rq);
}



static void enqueue_task_wrr(struct rq *rq, struct task_struct *p, int
    flags)
{
  struct sched_wrr_entity *wrr_se;
  struct wrr_rq *wrr_rq;
  struct list_head *queue;

  wrr_se = &p->wrr;
  wrr_rq = &rq->wrr;
  queue = &wrr_rq->wrr_rq_list;


  list_add_tail(&wrr_se->run_list, queue);
  wrr_rq->wrr_nr_running++;

  inc_nr_running(rq);
  //printk("enqueue_task_wrr 2 %d\n",p->pid);
}

static void requeue_task_wrr(struct rq *rq, struct task_struct *p, int
```

```
    flags)
{
  //printk("requeue_task_wrr 1 %d\n",p->pid);
  struct sched_wrr_entity *wrr_se;
  struct wrr_rq *wrr_rq;
  struct list_head *head;

  wrr_se = &p->wrr;
  wrr_rq = &rq->wrr;
  head = &wrr_rq->wrr_rq_list;

  if (sizeof(wrr_rq->wrr_nr_running) == 1)
    return;

  list_move_tail(&wrr_se->run_list, head);

  //printk("requeue_task_wrr 1 %d\n",p->pid);
}

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

static struct task_struct *pick_next_task_wrr(struct rq *rq)
{
  //printk("pick_next_task_wrr 1\n");
  struct sched_wrr_entity *wrr_se;
  struct task_struct *p;
  struct wrr_rq *wrr_rq;
  struct list_head *queue;

  wrr_rq = &rq->wrr;

  if (!wrr_rq->wrr_nr_running)
    return NULL;

  queue = &wrr_rq->wrr_rq_list;
  wrr_se = list_entry(queue->next, struct sched_wrr_entity, run_list);
  return container_of(wrr_se, struct task_struct, wrr);
  //printk("pick_next_task_wrr 2\n");
}

static void put_prev_task_wrr(struct rq *rq, struct task_struct *prev)
```

```
{

}

static void task_tick_wrr(struct rq *rq, struct task_struct *p, int queued)
{
	printk("task_tick_wrr pid:%d timeslice: %d\n",p->pid,p->wrr.time_slice);
	struct sched_wrr_entity *wrr_se;

	wrr_se = &p->wrr;
	if (--p->wrr.time_slice)
		return;
	//char *tmp=task_group_path(p->sched_task_group);

	//printk("tmp: %s",tmp);
	//printk("task_group_path: %s",task_group_path(p->sched_task_group));

	if(task_group_path(p->sched_task_group) [1] != 'b')
	{
		printk("WRR Fore Ground\n");
		p->wrr.time_slice = WRR_TIMESLICE_FG;
	}
	else
	{
		printk("WRR Back Ground\n");
		p->wrr.time_slice = WRR_TIMESLICE_BG;
	}

	if (wrr_se->run_list.prev != wrr_se->run_list.next) {
		requeue_task_wrr(rq, p, 0);
		set_tsk_need_resched(p);
		return;
	}
}

static void set_curr_task_wrr(struct rq *rq)
{
	struct task_struct *p;

	p = rq->curr;
	p->se.exec_start = rq->clock_task;

}

```

```
static void check_preempt_curr_wrr(struct rq *rq,
          struct task_struct *p, int flags)
{
   if (p->prio < rq->curr->prio) {
      resched_task(rq->curr);
      return;
   }
}

static void switched_to_wrr(struct rq *rq, struct task_struct *p)
{
   struct sched_wrr_entity *wrr_se;

   wrr_se = &p->wrr;
   if(task_group_path(p->sched_task_group) [1] != 'b')
      p->wrr.time_slice = WRR_TIMESLICE_FG;

   else
      p->wrr.time_slice = WRR_TIMESLICE_BG;

   INIT_LIST_HEAD(&wrr_se->run_list);
}

static void prio_changed_wrr(struct rq *rq, struct task_struct *p, int old)
{ }


static unsigned int get_rr_interval_wrr(struct rq *rq, struct task_struct
    *p)
{
   if (p == NULL)
      return -EINVAL;

   if(task_group_path(p->sched_task_group) [1] != 'b')
      return WRR_TIMESLICE_FG;
   else
      return WRR_TIMESLICE_BG;
}

void init_wrr_rq(struct wrr_rq *wrr_rq, struct rq *rq)
{
   wrr_rq->wrr_nr_running = 0;
   INIT_LIST_HEAD(&wrr_rq->wrr_rq_list);
}
```

```


const struct sched_class wrr_sched_class = {
    .next      = &fair_sched_class,
    .dequeue_task   = dequeue_task_wrr,
    .enqueue_task   = enqueue_task_wrr,
    .yield_task   = yield_task_wrr,
    .check_preempt_curr = check_preempt_curr_wrr,
    .pick_next_task    = pick_next_task_wrr,
    .put_prev_task   = put_prev_task_wrr,

#ifdef CONFIG_SMP
    .select_task_rq    = select_task_rq_wrr,
    .rq_online    = rq_online_wrr,
    .rq_offline   = rq_offline_wrr,
    .task_woken   = task_woken_wrr,
    .switched_from   = switched_from_wrr,
    .pre_schedule   = pre_schedule_wrr,
    .post_schedule   = post_schedule_wrr,
#endif

    .set_curr_task   = set_curr_task_wrr,
    .task_tick    = task_tick_wrr,

    .get_rr_interval = get_rr_interval_wrr,

    .prio_changed   = prio_changed_wrr,
    .switched_to    = switched_to_wrr,
};
```