CS356 Operating System Projects Spring 2016

Project 2: Android Scheduler

Report

Name : Zhengtian Xu StudentID: 5140309178

1. Objectives

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

2. Assumption

- Assume that the output of processtest.apk is millisecond。

3. Task1: Change the Scheduler of Processes

3.1 SCHED_NORMAL

- **Description**

SCHED_NORMAL isn’t a real time policy in Linux Kernel, it’s a default universal time sharing scheduling policy which is used by most threads. The policy uses time slicing, which means that each thread runs for a limited time period, after which the next thread is allowed to run.

- **Result**

Since the default policy of the processtest.apk is SCHED_NORMAL, I test five times respectively for several different input. The average time of the process is listed in the following table.

Table 1

The Average Time in SCHED_NORMAL

Input	Average Time	Input	Average Time
10	817.6	80	5794.6
20	1429.8	90	6416.8
30	2236.2	100	7183.8
40	3189.4	150	10485.8
50	3785.8	200	13573.8
60	4511.6	250	17170.4
70	5240.0	300	20373.6

After linear regression, I get the function between the input number and the average time when using SCHED_NORMAL scheduler.

$$y = 67.01 \times x + 359.9$$

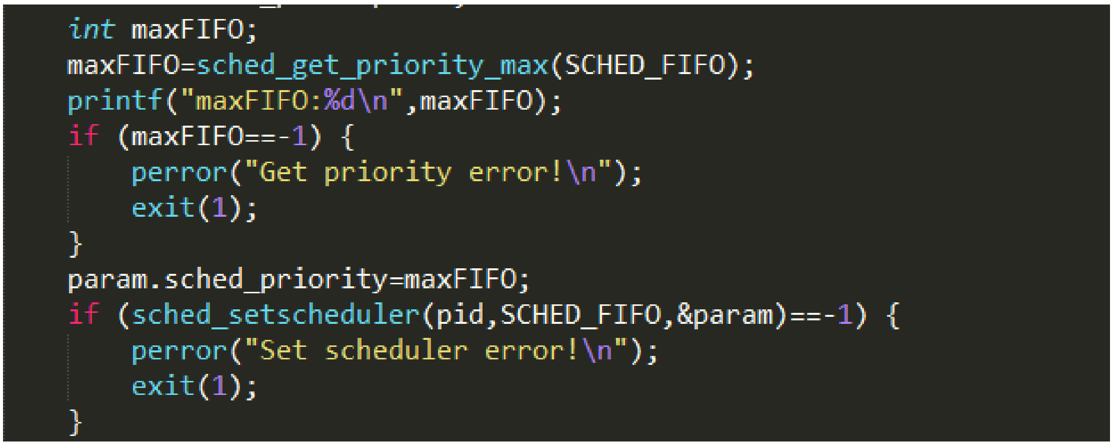
3.2 SCHED_FIFO

- **Target**

  Change the scheduler of test applications to SCHED_FIFO, and compare the executing time of them with the time using SCHED_NORMAL. The priorities of them should be same.

- **Implementation**

In order to implement this problem, I use s function which is called sched_setscheduler( ), the role of this function is changing the scheduling policy of a process. In this task, I use the max priority of SCHED_FIFO to show the main difference between these three scheduling policy. The following is the main code to change the scheduling policy to SCHED_FIFO.



```
    int maxFIFO;
    maxFIFO=sched_get_priority_max(SCHED_FIFO);
    printf("maxFIFO:%d\n",maxFIFO);
    if (maxFIFO==-1) {
        perror("Get priority error!\n");
        exit(1);
    }
    param.sched_priority=maxFIFO;
    if (sched_setscheduler(pid,SCHED_FIFO,&param)==-1) {
        perror("Set scheduler error!\n");
        exit(1);
    }
```

- **Result**

  The current policy of the processtest.apk is SCHED_FIFO, I also test five times respectively for several different input. The average time of the process is listed in the following table.

Table 2

The Average Time in SCHED_FIFO

Input	Average Time	Input	Average Time
10	650.2	80	6518.6
20	1372.8	90	7215.6
30	2187.6	100	8006.6
40	3190.4	150	12143.2
50	4015.4	200	16142.8
60	4804.8	250	20173.0
70	5513.4	300	24141.4

After linear regression, I get the function between the input number and the average time when using SCHED_FIFO scheduler.

$$y = 81.18 \times x - 116.5$$

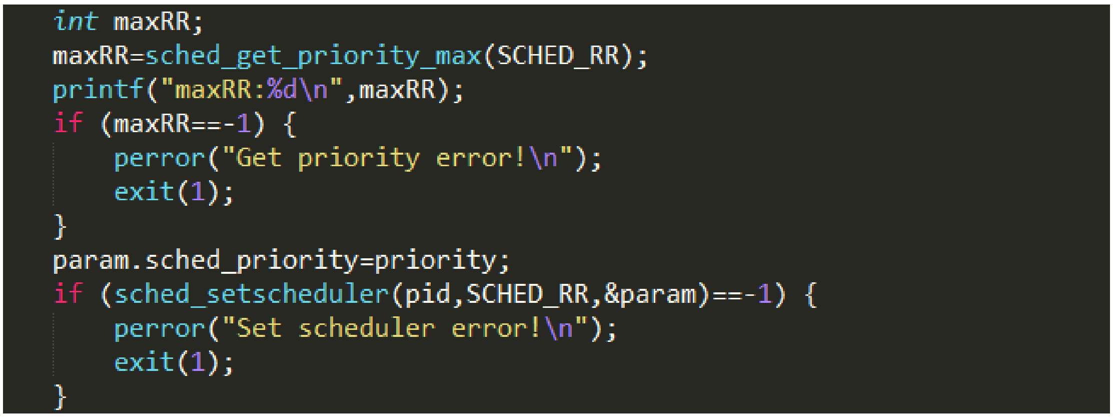
3.3 SCHED_RR

- **Target**

  Change the scheduler of test applications to SCHED_RR, and compare the executing time of them with the time using SCHED_NORMAL. The priorities of them should be same.

- **Implementation**

The principle of this change is the same with SCHED_FIFO. In this task, I use the maximal priority of SCHED_RR and the minimal priority of SCHED_RR to show the difference and influence when I use SCHED_RR scheduling policy. The following is the main code to change the scheduling policy to SCHED_RR.



```
int maxRR;
maxRR=sched_get_priority_max(SCHED_RR);
printf("maxRR:%d\n",maxRR);
if (maxRR==-1) {
    perror("Get priority error!\n");
    exit(1);
}
param.sched_priority=priority;
if (sched_setscheduler(pid,SCHED_RR,&param)==-1) {
    perror("Set scheduler error!\n");
    exit(1);
}
```

- **Result**

The current policy of the processtest.apk is SCHED_RR, I also test five times respectively for several different input. The average time of the process is listed in the following table.

Table 3

The Average Time in SCHED_RR in Maximal Priority

Input	Average Time	Input	Average Time
10	654.2	80	6340.8
20	1462.4	90	7178.2
30	2301.4	100	7894.6
40	3153.0	150	12120.2
50	3973.0	200	16186.4
60	4704.4	250	20119.6
70	5525.0	300	24119.2

After linear regression, I get the function between the input number and the average time when using SCHED_RR scheduler in maximal priority.

$$y = 81.07 \times x - 129.4$$

Table 4

The Average Time in SCHED_RR in Minimal Priority

Input	Average Time	Input	Average Time
10	655.2	80	6434.6
20	1446.0	90	7267.8
30	2250.8	100	8021.8
40	3104.0	150	12356.2
50	3930.2	200	16396.8
60	4795.2	250	20436.6
70	5674.0	300	24363.6

After linear regression, I get the function between the input number and the average time when using SCHED_RR scheduler in minimal priority.

$$y = 82.26 \times x - 153$$

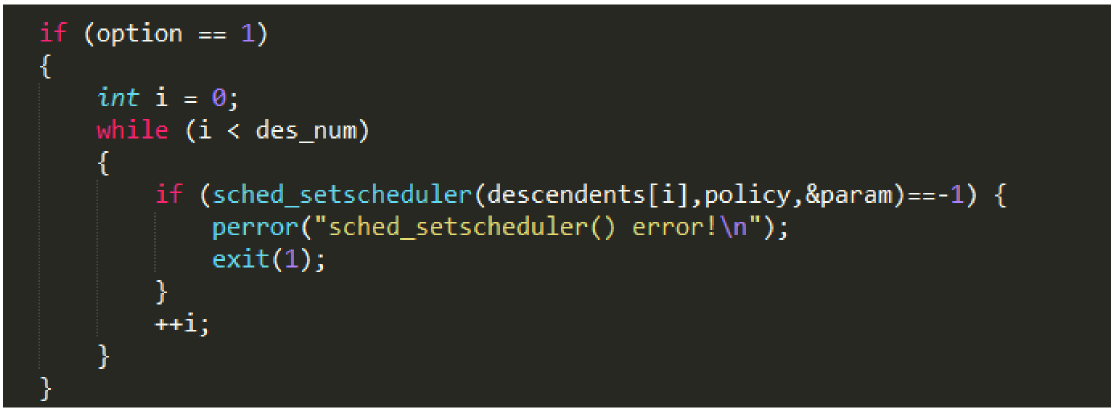
3.4 SCHED_RR for Zygote

- **Target**

Change the scheduler of all descendants of process zygote to SCHED_RR, and compare the executing time of them with the time using SCHED_NORMAL. The priority of any process exclude test application should be same.

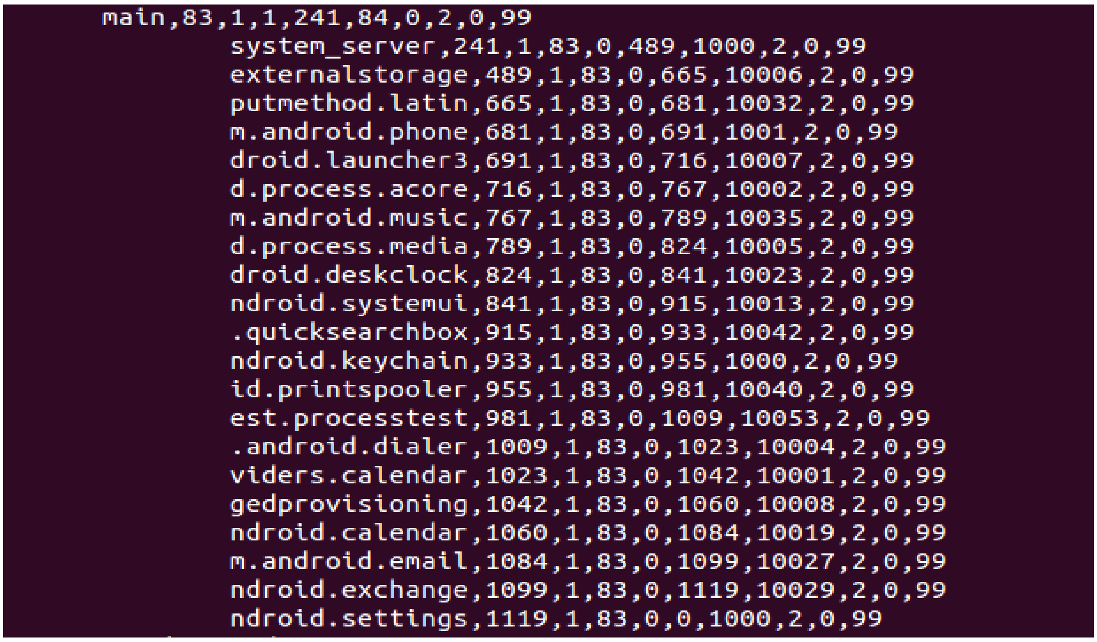
- **Implementation**

In this sub-problem, I use the system call which I have written in Project 1. The system call implements the function that it can find all the descendants of zygote. It is easy to get their pids through the buffer produced by system call. I use a matrix called descendants to store every descendant's pid of any process we want. Then I loop through the matrix and use the function sched_setscheduler( ), which will realize the function that it can change all the descendants' scheduling policy of any process to SCHED_RR or SCHED_FIFO with any priority. The following is the main code.



```
if (option == 1)
{
    int i = 0;
    while (i < des_num)
    {
        if (sched_setscheduler(descendents[i],policy,&param)==-1) {
            perror("sched_setscheduler() error!\n");
            exit(1);
        }
        ++i;
    }
}
```

- **Result**



```
      main,83,1,1,241,84,0,2,0,99
              system_server,241,1,83,0,489,1000,2,0,99
              externalstorage,489,1,83,0,665,10006,2,0,99
              putmethod.latin,665,1,83,0,681,10032,2,0,99
              m.android.phone,681,1,83,0,691,1001,2,0,99
              droid.launcher3,691,1,83,0,716,10007,2,0,99
              d.process.acore,716,1,83,0,767,10002,2,0,99
              m.android.music,767,1,83,0,789,10035,2,0,99
              d.process.media,789,1,83,0,824,10005,2,0,99
              droid.deskclock,824,1,83,0,841,10023,2,0,99
              ndroid.systemui,841,1,83,0,915,10013,2,0,99
              .quicksearchbox,915,1,83,0,933,10042,2,0,99
              ndroid.keychain,933,1,83,0,955,1000,2,0,99
              id.printspooler,955,1,83,0,981,10040,2,0,99
              est.processtest,981,1,83,0,1009,10053,2,0,99
              .android.dialer,1009,1,83,0,1023,10004,2,0,99
              viders.calendar,1023,1,83,0,1042,10001,2,0,99
              gedprovisioning,1042,1,83,0,1060,10008,2,0,99
              ndroid.calendar,1060,1,83,0,1084,10019,2,0,99
              m.android.email,1084,1,83,0,1099,10027,2,0,99
              ndroid.exchange,1099,1,83,0,1119,10029,2,0,99
              ndroid.settings,1119,1,83,0,0,1000,2,0,99
```

To get the result clearly, I also use the system call of ptree. As the picture aboved, we can see the first column is the name of process and ‘main’ represents zygote. I add three columns based on the first project and the meaning of the last three columns are respectively representing the scheduling policy, the priority and the rt_priority of the process. Specifically, 2 represents SCHED_RR, 1 represents SCHED_FIFO and 0 represents SCHED_NORMAL in the column of scheduling policy.

Since the current policy of the descendants of zygote is SCHED_RR, I also test five times respectively for several different input of processtest.apk. The average time of the process is listed in the following table.

Table 5

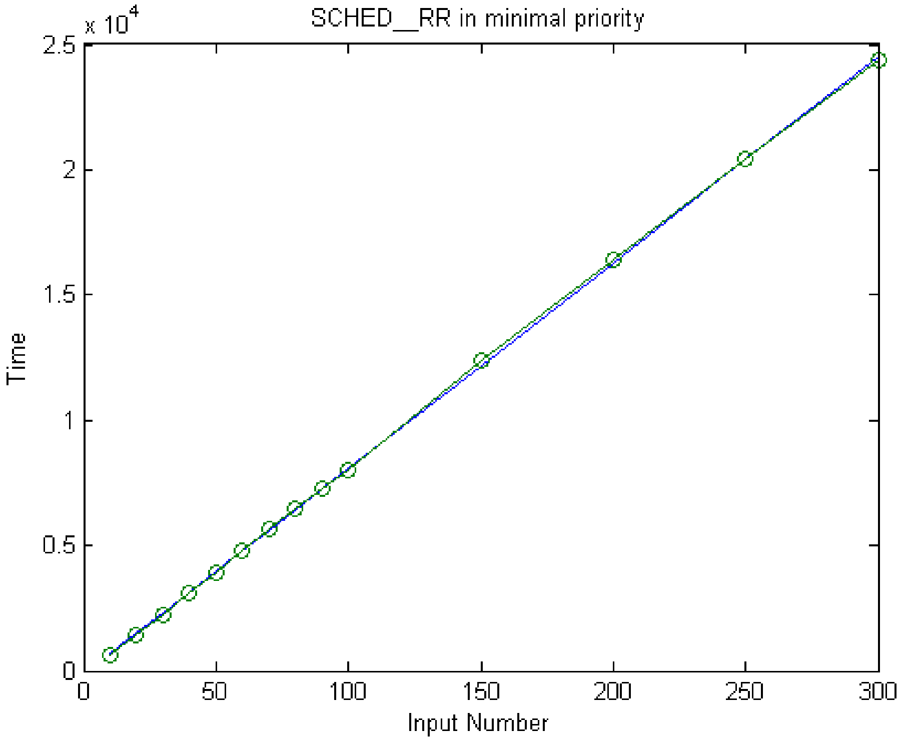
The Average Time of All SCHED_RR in Maximal Priority

Input	Average Time	Input	Average Time
10	659.8	80	6656.2
20	1508.0	90	7296.6
30	2169.4	100	7935.2
40	3211.8	150	12219.2
50	3862.6	200	16038.4
60	4733.2	250	20327.6
70	5493.8	300	24152.2

After linear regression, I get the function between the input number and the average time when all the descendants using SCHED_RR scheduler in maximal priority.

$$y = 81.31 \times x - 117$$

Then I get the trend of the average time with the changing of the input number.



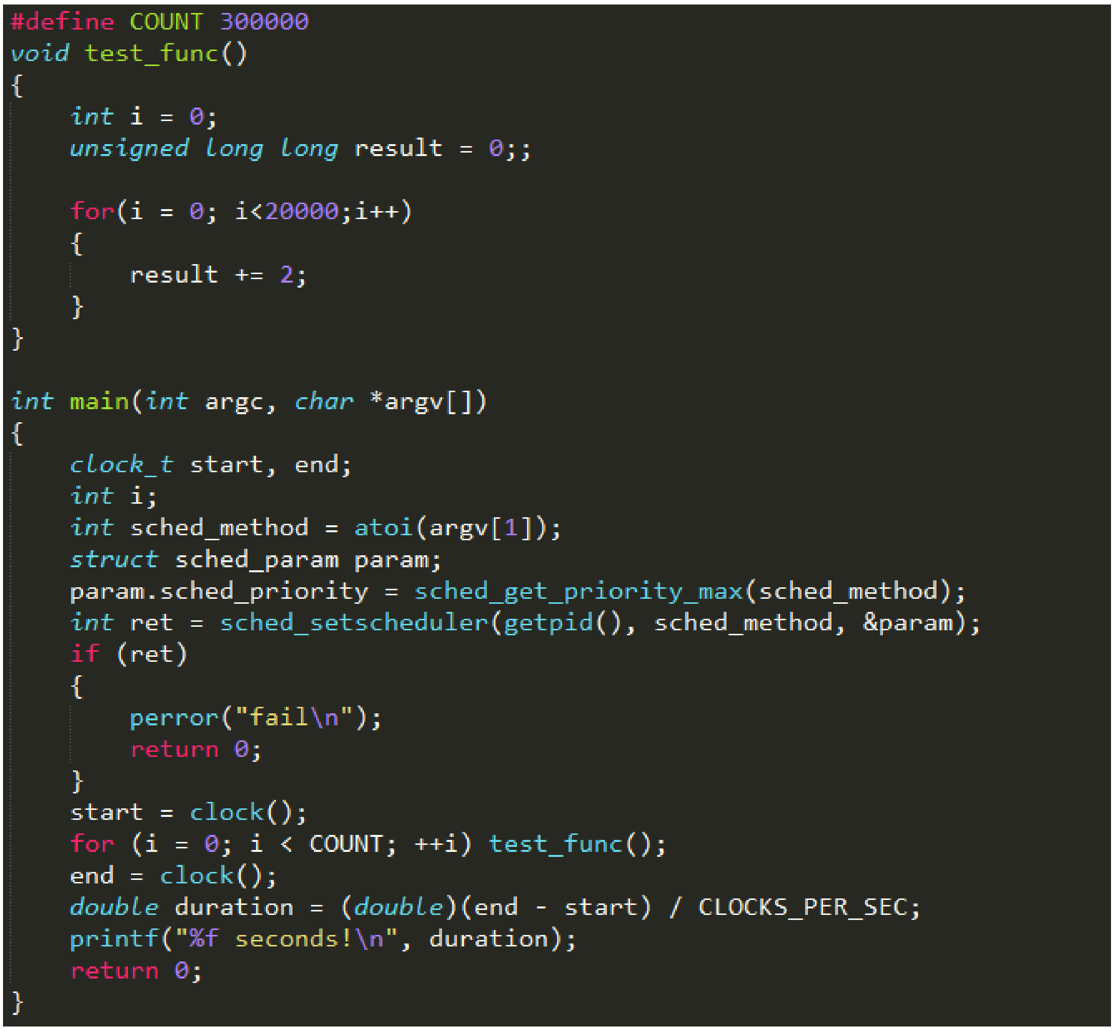
3.5 Compete with Linux Process with Three Different Policy

- **Target**

To test the new scheduler, you should set the priority of the android application as a certain number. Then, execute the two applications repeatedly to observe the difference between new scheduler and original scheduler.

- **Linux Process**

I write a linux test process to compete with the android application. The linux process will calculate the addition equation for many times. It can change its scheduling policy to any policy, which is realized by writing three similar test process. The following is the main code.



```
#define COUNT 300000
void test_func()
{
    int i = 0;
    unsigned long long result = 0;;

    for(i = 0; i<20000;i++)
    {
        result += 2;
    }
}

int main(int argc, char *argv[])
{
    clock_t start, end;
    int i;
    int sched_method = atoi(argv[1]);
    struct sched_param param;
    param.sched_priority = sched_get_priority_max(sched_method);
    int ret = sched_setscheduler(getpid(), sched_method, &param);
    if (ret)
    {
        perror("fail\n");
        return 0;
    }
    start = clock();
    for (i = 0; i < COUNT; ++i) test_func();
    end = clock();
    double duration = (double)(end - start) / CLOCKS_PER_SEC;
    printf("%f seconds!\n", duration);
    return 0;
}
```

- **Result**
  - **Independent running with default scheduling policy**

I also test five times respectively for the same input. The average time of the process and linux process is listed in the following table.

Running in SCHED_NORMAL

	Input	Average Time(s)
Android application	120	8.08
Linux test file		11.38

- **SCHED_NORMAL**

Then I run these two processes almost concurrently but with a little bit difference, that is which process runs first. I want to see whether the order of process will affect the running time. I test five times for these two situations and the result is in the following tables.

Android Application First Running

	Input	Average Time(s)
Android application	120	15.76
Linux test file		11.63

Linux Test File First Running

	Input	Average Time(s)
Android application	120	16.55
Linux test file		11.53

- **SCHED_FIFO**

Then I change the scheduling policy of both processes to SCHED_FIFO.I test five times for these two situations and the result is in the following tables.

Android Application First Running

	Input	Average Time(s)
Android application	120	9.69
Linux test file		11.17

Linux Test File First Running

	Input	Average Time(s)
Android application	120	9.78
Linux test file		11.16

- **SCHED_RR**

Then I change the scheduling policy of both processes to SCHED_RR.I test five times for these two situations and the result is in the following tables.

Android Application First Running

	Input	Average Time(s)
Android application	120	16.08
Linux test file		11.24

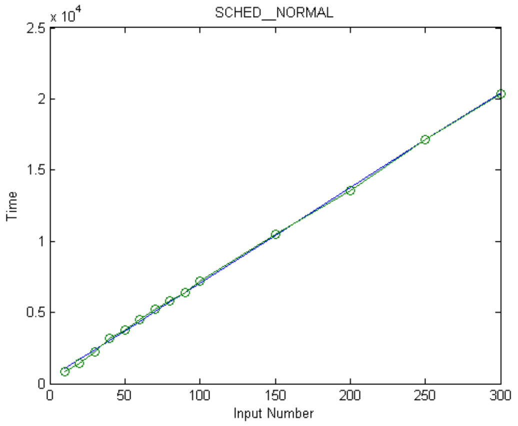
Linux Test File First Running

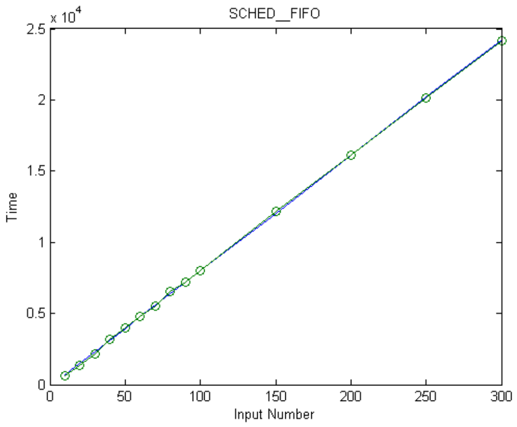
	Input	Average Time(s)
Android application	120	17.86
Linux test file		11.23

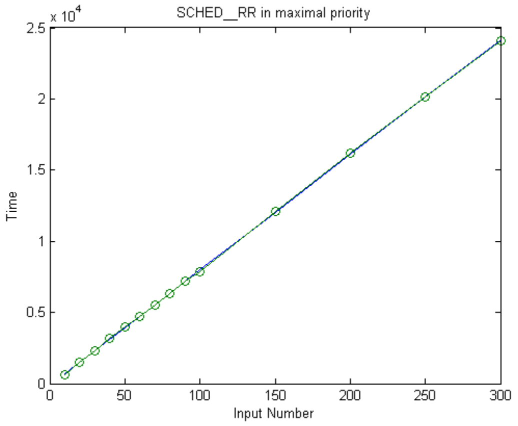
3.6 Comparison and Analysis

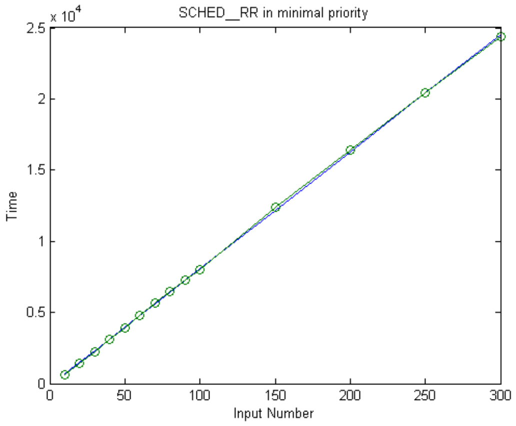
- **Comparison between Three Different Policy**

In the first sub-problem, I change the Android Application to three different scheduling policy respectively. We can draw the trend and relation of policies in these following pictures.









In these pictures, green line represents curve fitting by points I have recorded and blue line represents the fitting function.

Through these pictures we can find that in small time slice, using SCHED_RR policy and SCHED_FIFO policy will result in faster running time than SCHED_NORMAL policy. That is because SCHED_FIFO and SCHED_RR are real-time policy. If the current CPU running non-real-time process, the real-time process immediately preempt the non-real-time process.

But when we give the application a large time slice as its input number, the result is that using SCHED_RR policy and SCHED_FIFO policy will result in slower running time than SCHED_NORMAL policy. I individually think that SCHED_NORMAL is a completely fair scheduler. It can guarantee that every process will get the CPU resources fairly. So in a long term time slice, CFS scheduler will result in a great advantage over other policies. That's why CFS scheduler is the default scheduler.

When we compare SCHED_FIFO between SCHED_RR, we will find there is almost no difference between these two scheduling policy. The principle of

SCHED_FIFO use “first in, first out” scheduler, the CPU will always choose the process until it finishes its task. And when we use SCHED_RR as scheduler, CPU will give each process different time quantum based on their priority, it will be moved out if it is not finished in unit time, so they will get equally CPU resource.

When we compare different real-time priority with SCHED_RR, we will find that there is almost no difference. But CPU will give each process different time quantum based on their priority, if the real-time priority is larger, the time slice it gets is shorter.

- **Comparison between Two Applications**

These two applications cause different results when we use different scheduler . As for this result I tested, SCHED_FIFO is faster than SCHED_NORMAL, and SCHED_NORMAL is better than SCHED_RR.

The reason for the result is that CPU will choose these applications to execute with SCHED_FIFO scheduling policy and when we use SCHED_RR scheduling policy, every process will get equal time slice to execute and they will experience several context switches. The time caused by context switched may longer than the running time. So SCHED_FIFO is best and SCHED_RR is worst.

An interesting phenomenon is that the running order of the process may result in the difference of the executing time.

4. Task2: Modify Scheduler

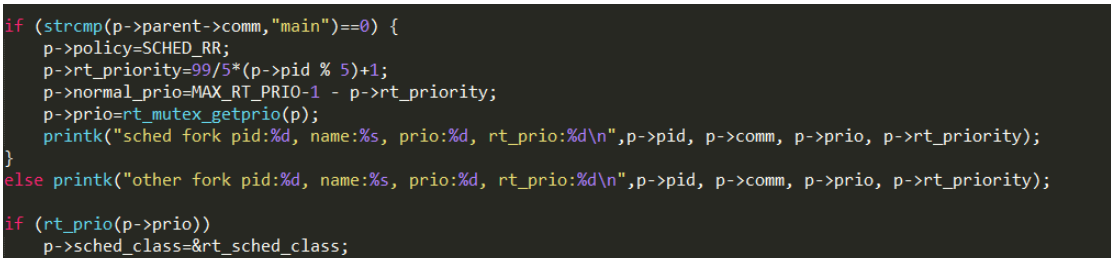
4.1 Change Default Scheduler

- **Target**

Default scheduler of all descendants of process zygote should be SCHED_RR. The priority of process should be (mαx priority of SCHED_RR)/5*(PID mod 5) + 1.

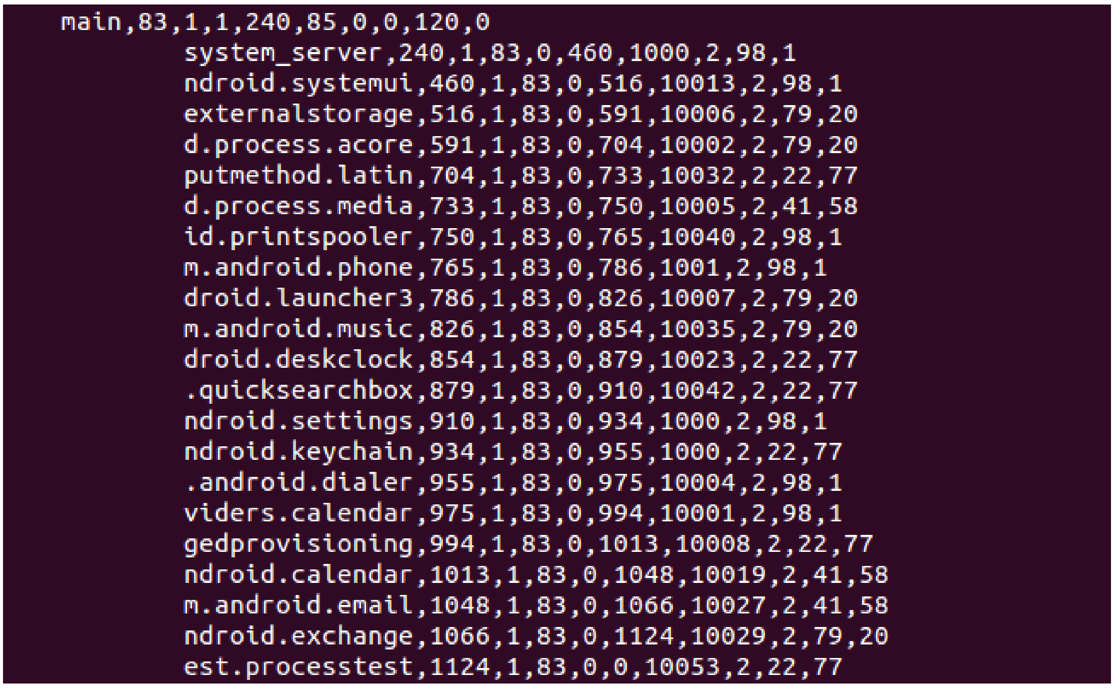
- **Implementation**

I modified kernel/fork.c in kernel to change the default scheduler and relevant priority of all descendants of process zygote. In the function do_fork( ), I add the following codes.



```
if (strcmp(p->parent->comm,"main")==0) {
    p->policy=SCHED_RR;
    p->rt_priority=99/5*(p->pid % 5)+1;
    p->normal_prio=MAX_RT_PRIO-1 - p->rt_priority;
    p->prio=rt_mutex_getprio(p);
    printk("sched fork pid:%d, name:%s, prio:%d, rt_prio:%d\n",p->pid, p->comm, p->prio, p->rt_priority);
}
else printk("other fork pid:%d, name:%s, prio:%d, rt_prio:%d\n",p->pid, p->comm, p->prio, p->rt_priority);

if (rt_prio(p->prio))
    p->sched_class=&rt_sched_class;
```

- **Result**



```
main,83,1,1,240,85,0,0,120,0
        system_server,240,1,83,0,460,1000,2,98,1
        ndroid.systemui,460,1,83,0,516,10013,2,98,1
        externalstorage,516,1,83,0,591,10006,2,79,20
        d.process.acore,591,1,83,0,704,10002,2,79,20
        putmethod.latin,704,1,83,0,733,10032,2,22,77
        d.process.media,733,1,83,0,750,10005,2,41,58
        id.printspooler,750,1,83,0,765,10040,2,98,1
        m.android.phone,765,1,83,0,786,1001,2,98,1
        droid.launcher3,786,1,83,0,826,10007,2,79,20
        m.android.music,826,1,83,0,854,10035,2,79,20
        droid.deskclock,854,1,83,0,879,10023,2,22,77
        .quicksearchbox,879,1,83,0,910,10042,2,22,77
        ndroid.settings,910,1,83,0,934,1000,2,98,1
        ndroid.keychain,934,1,83,0,955,1000,2,22,77
        .android.dialer,955,1,83,0,975,10004,2,98,1
        viders.calendar,975,1,83,0,994,10001,2,98,1
        gedprovisioning,994,1,83,0,1013,10008,2,22,77
        ndroid.calendar,1013,1,83,0,1048,10019,2,41,58
        m.android.email,1048,1,83,0,1066,10027,2,41,58
        ndroid.exchange,1066,1,83,0,1124,10029,2,79,20
        est.processtest,1124,1,83,0,0,10053,2,22,77
```

We can see that the last three columns of the descendants of zygote respectively represents its scheduling policy, priority and real-time priority, and they have been changed to SCHED_RR and the priority we want.

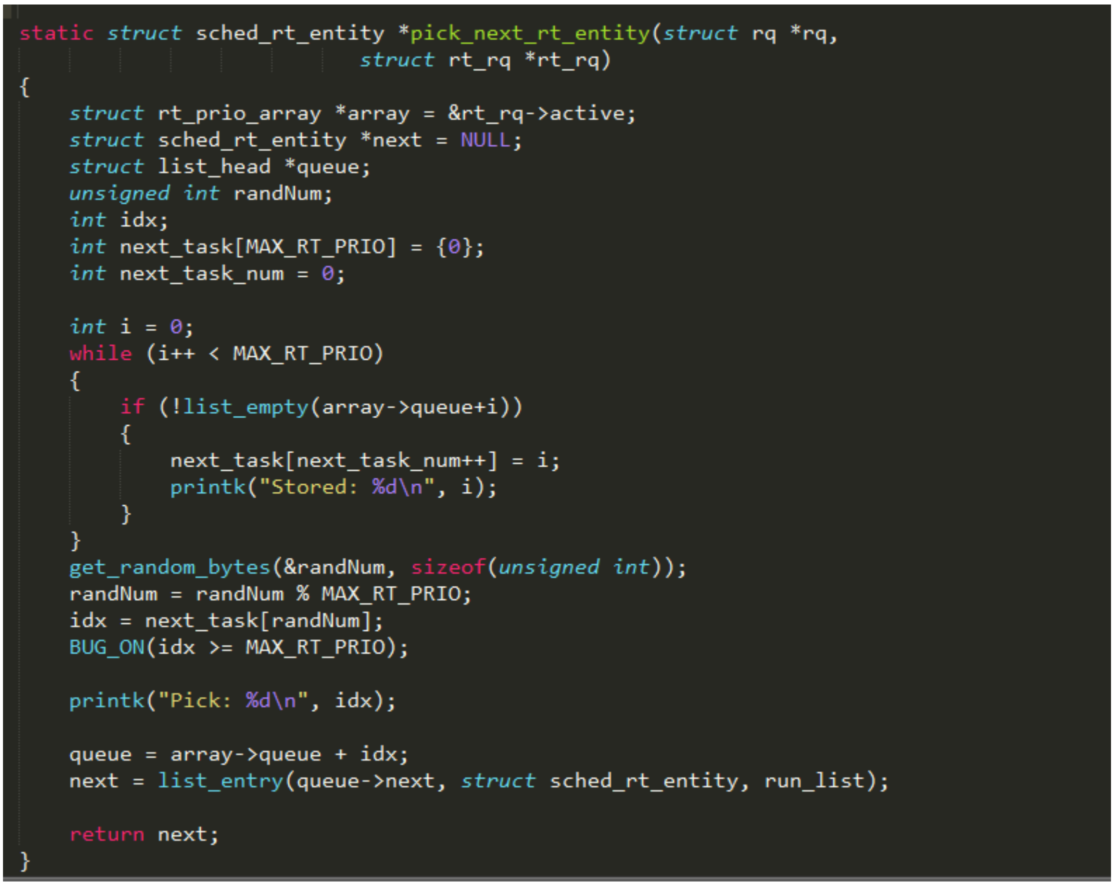
4.2 Pick the Next Process Randomly

- **Target**

Change the policy of SCHED_RR to pick the next process randomly.

- **Implementation**

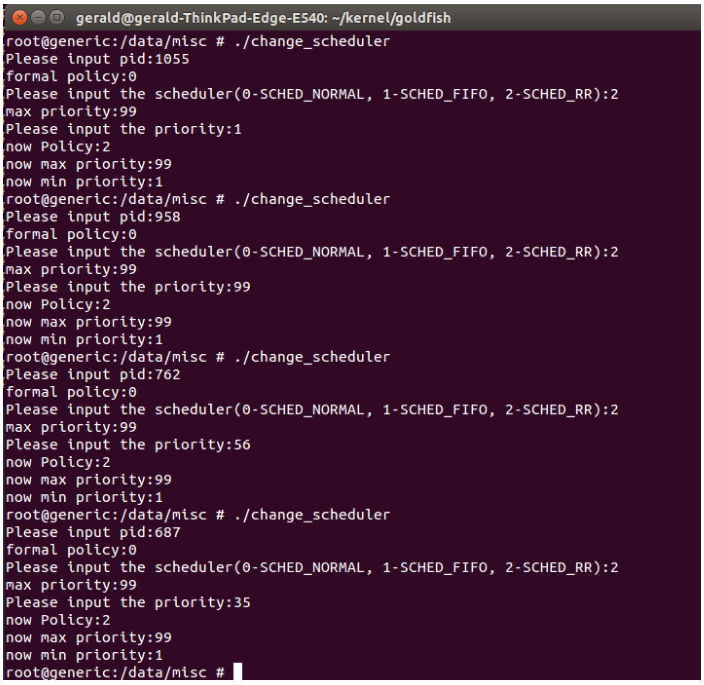
I modify kernel/sched/rt.c in kernel to change the policy of SCHED_RR to the next process randomly. In the function pick_next_rt_entity ( ), I add the following codes.



```
static struct sched_rt_entity *pick_next_rt_entity(struct rq *rq,
                                                   struct rt_rq *rt_rq)
{
    struct rt_prio_array *array = &rt_rq->active;
    struct sched_rt_entity *next = NULL;
    struct list_head *queue;
    unsigned int randNum;
    int idx;
    int next_task[MAX_RT_PRIO] = {0};
    int next_task_num = 0;

    int i = 0;
    while (i++ < MAX_RT_PRIO)
    {
        if (!list_empty(array->queue+i))
        {
            next_task[next_task_num++] = i;
            printk("Stored: %d\n", i);
        }
    }
    get_random_bytes(&randNum, sizeof(unsigned int));
    randNum = randNum % MAX_RT_PRIO;
    idx = next_task[randNum];
    BUG_ON(idx >= MAX_RT_PRIO);

    printk("Pick: %d\n", idx);

    queue = array->queue + idx;
    next = list_entry(queue->next, struct sched_rt_entity, run_list);

    return next;
}
```

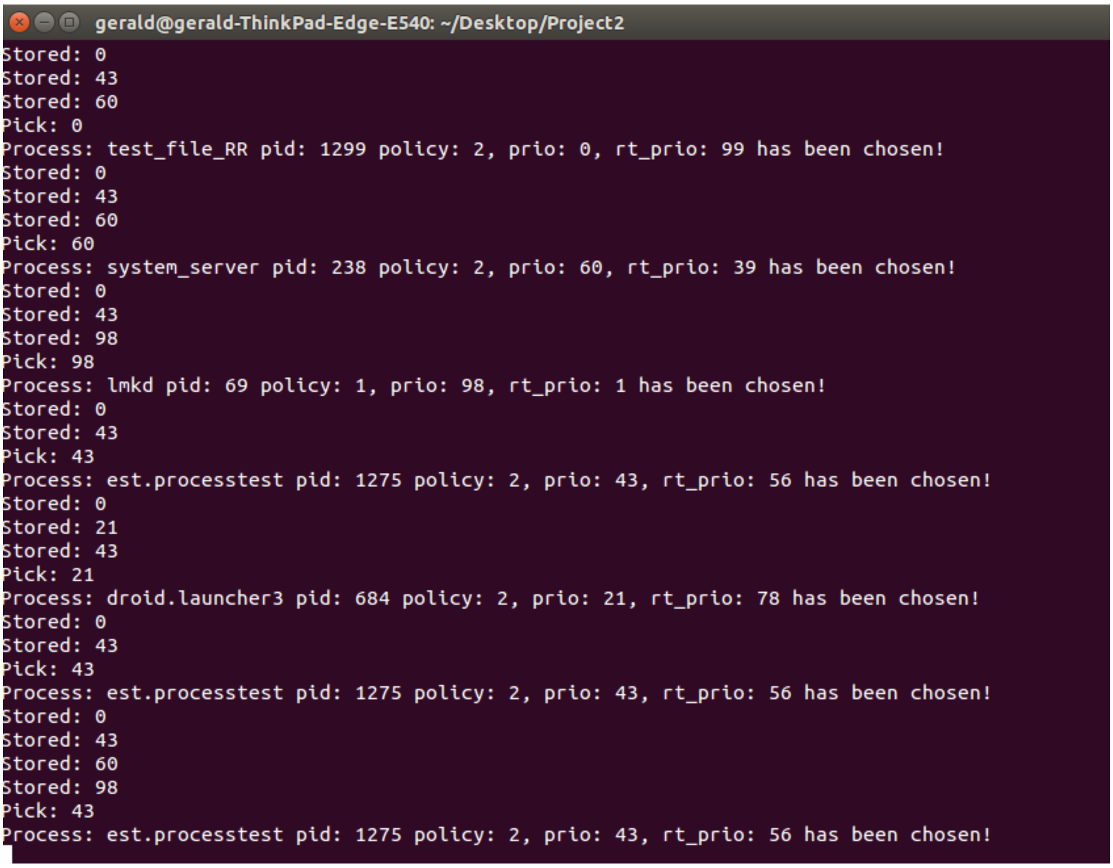
- **Result**

First, I change the scheduling policy several process including processtest.apk and test_file to SCHED_RR. Then I set different priority for different process. The following picture indicates my operation for some processes.



```
root@generic:/data/misc # ./change_scheduler
Please input pid:1055
formal policy:0
Please input the scheduler(0-SCHED_NORMAL, 1-SCHED_FIFO, 2-SCHED_RR):2
max priority:99
Please input the priority:1
now Policy:2
now max priority:99
now min priority:1
root@generic:/data/misc # ./change_scheduler
Please input pid:958
formal policy:0
Please input the scheduler(0-SCHED_NORMAL, 1-SCHED_FIFO, 2-SCHED_RR):2
max priority:99
Please input the priority:99
now Policy:2
now max priority:99
now min priority:1
root@generic:/data/misc # ./change_scheduler
Please input pid:762
formal policy:0
Please input the scheduler(0-SCHED_NORMAL, 1-SCHED_FIFO, 2-SCHED_RR):2
max priority:99
Please input the priority:56
now Policy:2
now max priority:99
now min priority:1
root@generic:/data/misc # ./change_scheduler
Please input pid:687
formal policy:0
Please input the scheduler(0-SCHED_NORMAL, 1-SCHED_FIFO, 2-SCHED_RR):2
max priority:99
Please input the priority:35
now Policy:2
now max priority:99
now min priority:1
root@generic:/data/misc #
```

When all the processes are set in SCHED_RR scheduling policy, the terminal will show which process is picked next. The following picture shows that the next process is randomly picked.



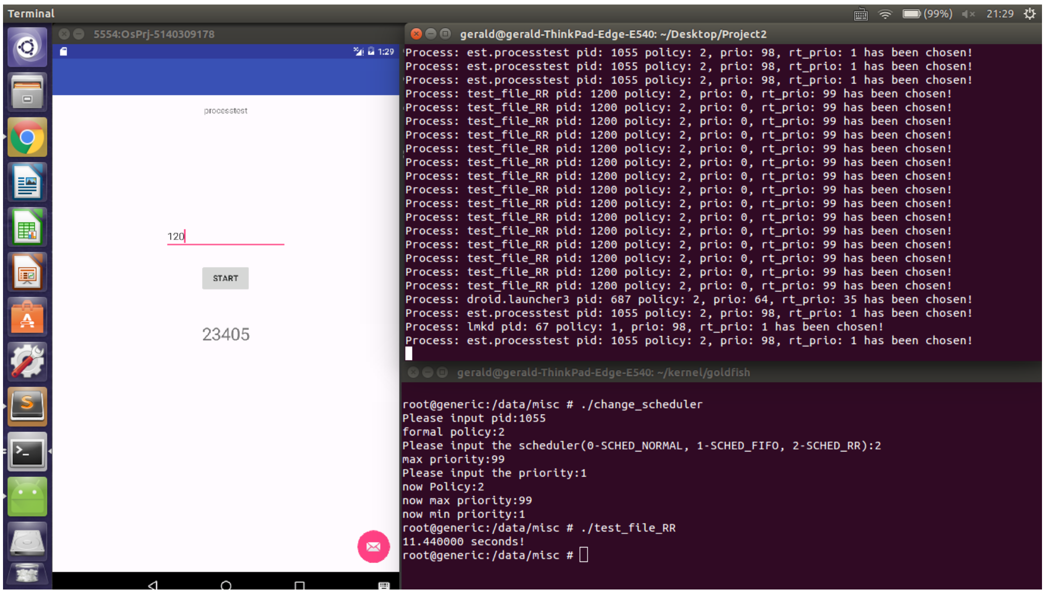
```
Stored: 0
Stored: 43
Stored: 60
Pick: 0
Process: test_file_RR pid: 1299 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Stored: 0
Stored: 43
Stored: 60
Pick: 60
Process: system_server pid: 238 policy: 2, prio: 60, rt_prio: 39 has been chosen!
Stored: 0
Stored: 43
Stored: 98
Pick: 98
Process: lmkd pid: 69 policy: 1, prio: 98, rt_prio: 1 has been chosen!
Stored: 0
Stored: 43
Pick: 43
Process: est.processtest pid: 1275 policy: 2, prio: 43, rt_prio: 56 has been chosen!
Stored: 0
Stored: 21
Stored: 43
Pick: 21
Process: droid.launcher3 pid: 684 policy: 2, prio: 21, rt_prio: 78 has been chosen!
Stored: 0
Stored: 43
Pick: 43
Process: est.processtest pid: 1275 policy: 2, prio: 43, rt_prio: 56 has been chosen!
Stored: 0
Stored: 43
Stored: 60
Stored: 98
Pick: 43
Process: est.processtest pid: 1275 policy: 2, prio: 43, rt_prio: 56 has been chosen!
```

5. Extension: The Relation Between Priority and Executing Time

5.1 The Situation with Default Setting

The scheduling policy of both applications must be set in SCHED_RR.

- **Different Priority**


Terminal
(99%)
21:29
5554:OsPrj-5140309178
1:29
processtest
120
START
23405
gerald@gerald-ThinkPad-Edge-E540: ~/Desktop/Project2
gerald@gerald-ThinkPad-Edge-E540: ~/kernel/goldfish

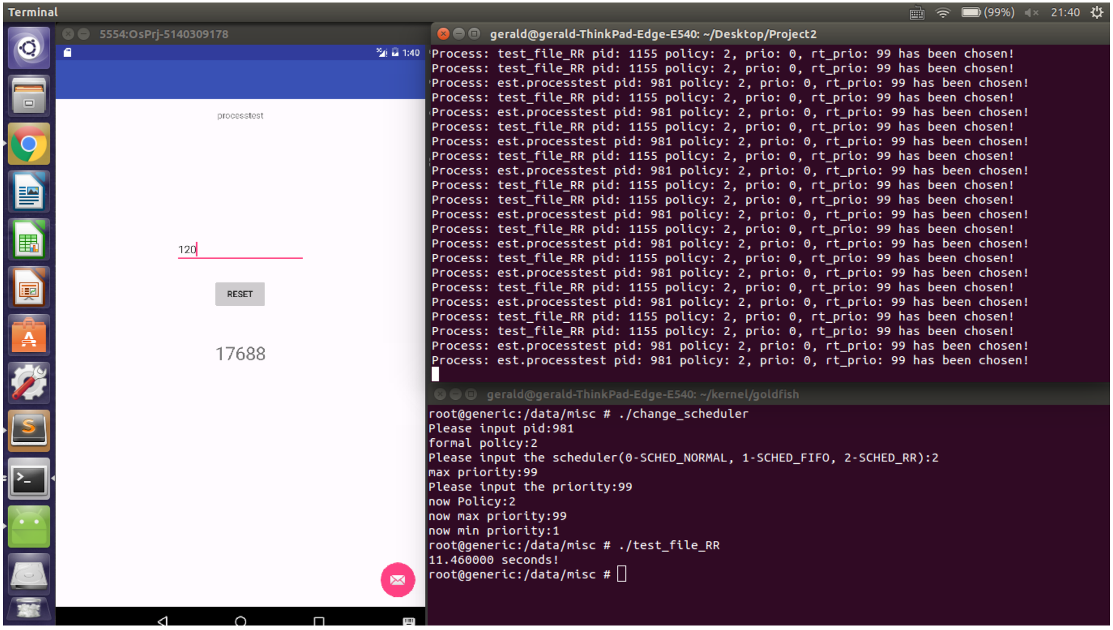
```
Process: est.processtest pid: 1055 policy: 2, prio: 98, rt_prio: 1 has been chosen!
Process: est.processtest pid: 1055 policy: 2, prio: 98, rt_prio: 1 has been chosen!
Process: est.processtest pid: 1055 policy: 2, prio: 98, rt_prio: 1 has been chosen!
Process: test_file_RR pid: 1200 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1200 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1200 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1200 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1200 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1200 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1200 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1200 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1200 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1200 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1200 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1200 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1200 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1200 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1200 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: droid.launcher3 pid: 687 policy: 2, prio: 64, rt_prio: 35 has been chosen!
Process: est.processtest pid: 1055 policy: 2, prio: 98, rt_prio: 1 has been chosen!
Process: lmkd pid: 67 policy: 1, prio: 98, rt_prio: 1 has been chosen!
Process: est.processtest pid: 1055 policy: 2, prio: 98, rt_prio: 1 has been chosen!
```

```
root@generic:/data/misc # ./change_scheduler
Please input pid:1055
formal policy:2
Please input the scheduler(0-SCHED_NORMAL, 1-SCHED_FIFO, 2-SCHED_RR):2
max priority:99
Please input the priority:1
now Policy:2
now max priority:99
now min priority:1
root@generic:/data/misc # ./test_file_RR
11.440000 seconds!
root@generic:/data/misc #
```

I execute android application and Linux test file at the same time, and I set the priority of Linux test file is the maximal priority of SCHED_RR and the priority of android application is the minimal priority of SCHED_RR. Then the result is shown as the above picture.

We can see that the Linux process always preempt the android application until it finish its process cycle. This experiment indicates that in default setting, CPU will choose the process with the higher priority as the next process to be executed. So the executing time of android application will be longer than other situation because it will be executed until Linux test file finish its task.

- **Same priority with Maximal Priority of SCHED_RR**


Terminal
(99%)
21:40
5554:OsPrj-5140309178
1:40
processtest
120
RESET
17688
gerald@gerald-ThinkPad-Edge-E540: ~/Desktop/Project2
gerald@gerald-ThinkPad-Edge-E540: ~/kernel/goldfish

```
Process: test_file_RR pid: 1155 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1155 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: est.processtest pid: 981 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1155 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: est.processtest pid: 981 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1155 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: est.processtest pid: 981 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1155 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: est.processtest pid: 981 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1155 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1155 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: est.processtest pid: 981 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1155 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: est.processtest pid: 981 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1155 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: est.processtest pid: 981 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1155 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: est.processtest pid: 981 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1155 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1155 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: est.processtest pid: 981 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: est.processtest pid: 981 policy: 2, prio: 0, rt_prio: 99 has been chosen!
```

```
root@generic:/data/misc # ./change_scheduler
Please input pid:981
formal policy:2
Please input the scheduler(0-SCHED_NORMAL, 1-SCHED_FIFO, 2-SCHED_RR):2
max priority:99
Please input the priority:99
now Policy:2
now max priority:99
now min priority:1
root@generic:/data/misc # ./test_file_RR
11.460000 seconds!
root@generic:/data/misc #
```

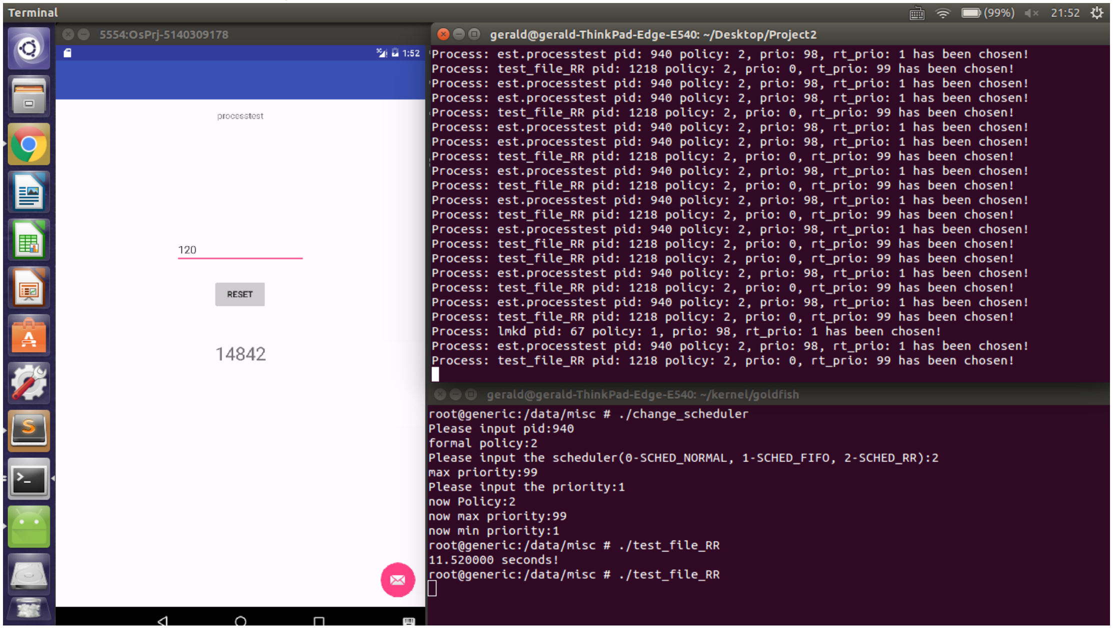
I execute android application and Linux test file at the same time, and I set the priority of Linux test file and android application are the maximal priority of SCHED_RR. Then the result is shown as the above picture.

We can see that android application and Linux test file get equal CPU resources. That is because they have the same priority. What’s more, it is obvious that the executing of android application is shorter than the last situation because it executes concurrently with Linux test file.

5.2 The Situation with Randomly Picking

The scheduling policy of both applications must be set in SCHED_RR.

- **Different Priority**


Terminal
(99%)
21:52
5554:OsPrj-5140309178
1:52
processtest
120
RESET
14842
gerald@gerald-ThinkPad-Edge-E540: ~/Desktop/Project2
gerald@gerald-ThinkPad-Edge-E540: ~/kernel/goldfish

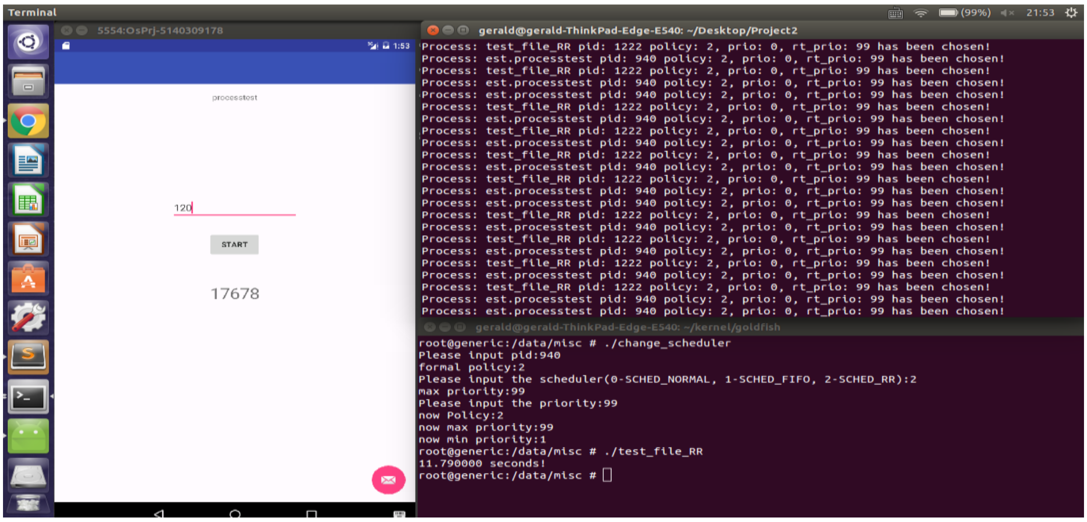
```
Process: est.processtest pid: 940 policy: 2, prio: 98, rt_prio: 1 has been chosen!
Process: test_file_RR pid: 1218 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: est.processtest pid: 940 policy: 2, prio: 98, rt_prio: 1 has been chosen!
Process: est.processtest pid: 940 policy: 2, prio: 98, rt_prio: 1 has been chosen!
Process: test_file_RR pid: 1218 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: est.processtest pid: 940 policy: 2, prio: 98, rt_prio: 1 has been chosen!
Process: est.processtest pid: 940 policy: 2, prio: 98, rt_prio: 1 has been chosen!
Process: test_file_RR pid: 1218 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: est.processtest pid: 940 policy: 2, prio: 98, rt_prio: 1 has been chosen!
Process: test_file_RR pid: 1218 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: est.processtest pid: 940 policy: 2, prio: 98, rt_prio: 1 has been chosen!
Process: test_file_RR pid: 1218 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: est.processtest pid: 940 policy: 2, prio: 98, rt_prio: 1 has been chosen!
Process: test_file_RR pid: 1218 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1218 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: est.processtest pid: 940 policy: 2, prio: 98, rt_prio: 1 has been chosen!
Process: test_file_RR pid: 1218 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: est.processtest pid: 940 policy: 2, prio: 98, rt_prio: 1 has been chosen!
Process: test_file_RR pid: 1218 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: lmkd pid: 67 policy: 1, prio: 98, rt_prio: 1 has been chosen!
Process: est.processtest pid: 940 policy: 2, prio: 98, rt_prio: 1 has been chosen!
Process: test_file_RR pid: 1218 policy: 2, prio: 0, rt_prio: 99 has been chosen!
```

```
root@generic:/data/misc # ./change_scheduler
Please input pid:940
formal policy:2
Please input the scheduler(0-SCHED_NORMAL, 1-SCHED_FIFO, 2-SCHED_RR):2
max priority:99
Please input the priority:1
now Policy:2
now max priority:99
now min priority:1
root@generic:/data/misc # ./test_file_RR
11.520000 seconds!
root@generic:/data/misc # ./test_file_RR
```

I execute android application and Linux test file at the same time, and I set the priority of Linux test file is the maximal priority of SCHED_RR and the priority of android application is the minimal priority of SCHED_RR. Then the result is shown as the above picture.

We can see that although the priority of Linux test file is greater than android application, they almost get equal CPU resources because of the randomly picking.

- **Same priority with Maximal Priority of SCHED_RR**


Terminal
(99%)
21:53
5554:OsPrj-5140309178
1:53
processtest
120
START
17678
gerald@gerald-ThinkPad-Edge-E540: ~/Desktop/Project2
gerald@gerald-ThinkPad-Edge-E540: ~/kernel/goldfish

```
Process: test_file_RR pid: 1222 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: est.processtest pid: 940 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1222 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: est.processtest pid: 940 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: est.processtest pid: 940 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1222 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: est.processtest pid: 940 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1222 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: est.processtest pid: 940 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1222 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: est.processtest pid: 940 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1222 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: est.processtest pid: 940 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: est.processtest pid: 940 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1222 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: est.processtest pid: 940 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1222 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: est.processtest pid: 940 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1222 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: est.processtest pid: 940 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: test_file_RR pid: 1222 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: est.processtest pid: 940 policy: 2, prio: 0, rt_prio: 99 has been chosen!
Process: est.processtest pid: 940 policy: 2, prio: 0, rt_prio: 99 has been chosen!
```

```
root@generic:/data/misc # ./change_scheduler
Please input pid:940
formal policy:2
Please input the scheduler(0-SCHED_NORMAL, 1-SCHED_FIFO, 2-SCHED_RR):2
max priority:99
Please input the priority:99
now Policy:2
now max priority:99
now min priority:1
root@generic:/data/misc # ./test_file_RR
11.790000 seconds!
root@generic:/data/misc #
```

I execute android application and Linux test file at the same time, and I set the priority of Linux test file and android application are the maximal priority of SCHED_RR. Then the result is shown as the above picture.

We can see that android application and Linux test file get equal CPU resources.

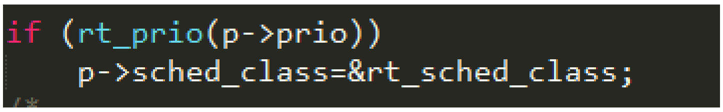
5.3 Problem and Analysis

In the randomly picking situation, theoretically, the executing time of android application is equal. But time in the second situation is longer than the first one. From my point of view, this may be caused by more context switch in the second situation. Since the fact that the higher the real-time priority is, the shorter the time slice is, when a process get high priority, it is obvious that the running cycle of this process includes many context switch, which will affect the overall time of the process.

6. Problem and Solution

6.1 Abnormal Startup

The second problem is modify default scheduling policy to SCHED_RR, but when I modify the sched_class of process from fair_sched_class to rt_sched_class. The following picture is relevant codes.

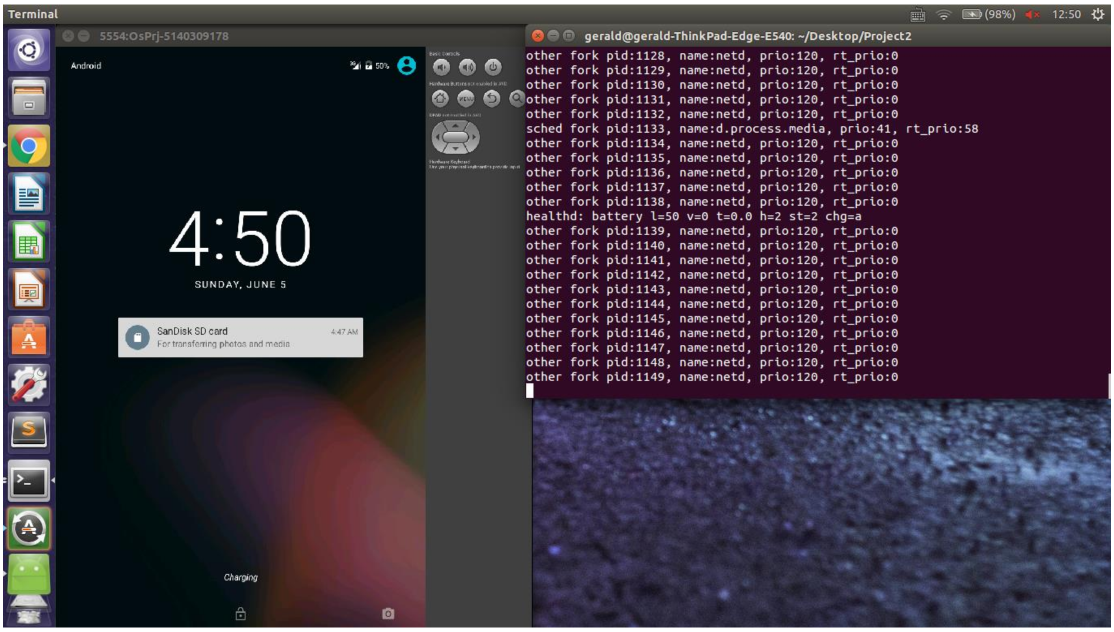


```
if (rt_prio(p->prio))
    p->sched_class=&rt_sched_class;
```

When the processes' sched_class have been changed, I observed that two situations may happen in the startup.

- **Normal Startup**

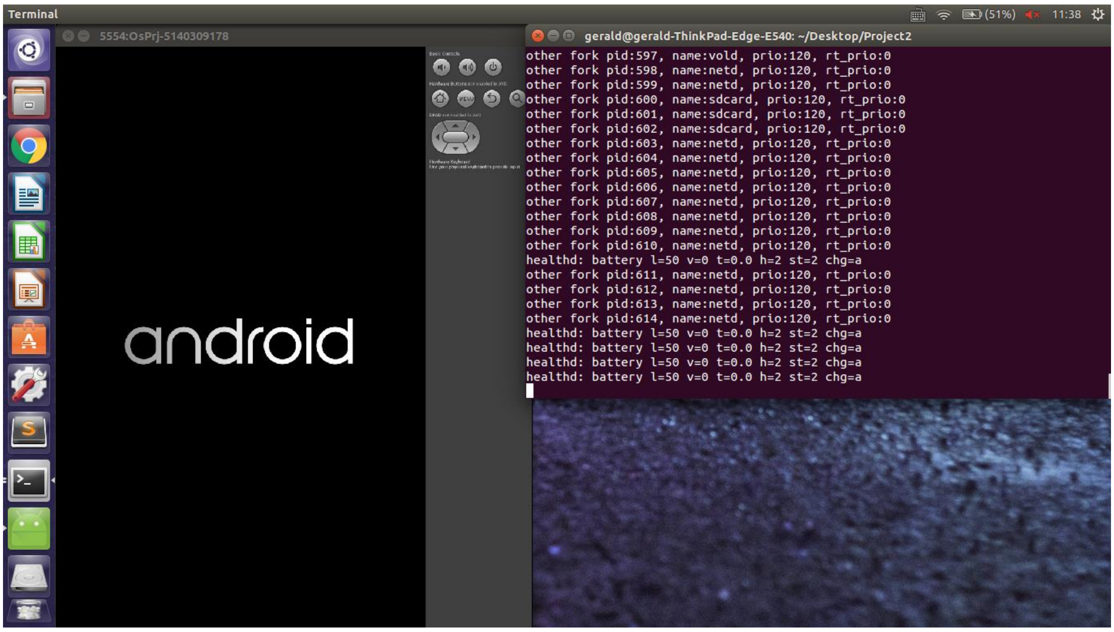
One situation is normal startup, and the following picture indicates the android virtual device can normally work.



```
other fork pid:1128, name:netd, prio:120, rt_prio:0
other fork pid:1129, name:netd, prio:120, rt_prio:0
other fork pid:1130, name:netd, prio:120, rt_prio:0
other fork pid:1131, name:netd, prio:120, rt_prio:0
other fork pid:1132, name:netd, prio:120, rt_prio:0
sched fork pid:1133, name:d.process.media, prio:41, rt_prio:58
other fork pid:1134, name:netd, prio:120, rt_prio:0
other fork pid:1135, name:netd, prio:120, rt_prio:0
other fork pid:1136, name:netd, prio:120, rt_prio:0
other fork pid:1137, name:netd, prio:120, rt_prio:0
other fork pid:1138, name:netd, prio:120, rt_prio:0
healthd: battery l=50 v=0 t=0.0 h=2 st=2 chg=a
other fork pid:1139, name:netd, prio:120, rt_prio:0
other fork pid:1140, name:netd, prio:120, rt_prio:0
other fork pid:1141, name:netd, prio:120, rt_prio:0
other fork pid:1142, name:netd, prio:120, rt_prio:0
other fork pid:1143, name:netd, prio:120, rt_prio:0
other fork pid:1144, name:netd, prio:120, rt_prio:0
other fork pid:1145, name:netd, prio:120, rt_prio:0
other fork pid:1146, name:netd, prio:120, rt_prio:0
other fork pid:1147, name:netd, prio:120, rt_prio:0
other fork pid:1148, name:netd, prio:120, rt_prio:0
other fork pid:1149, name:netd, prio:120, rt_prio:0
```

- **Stuck By a Process**

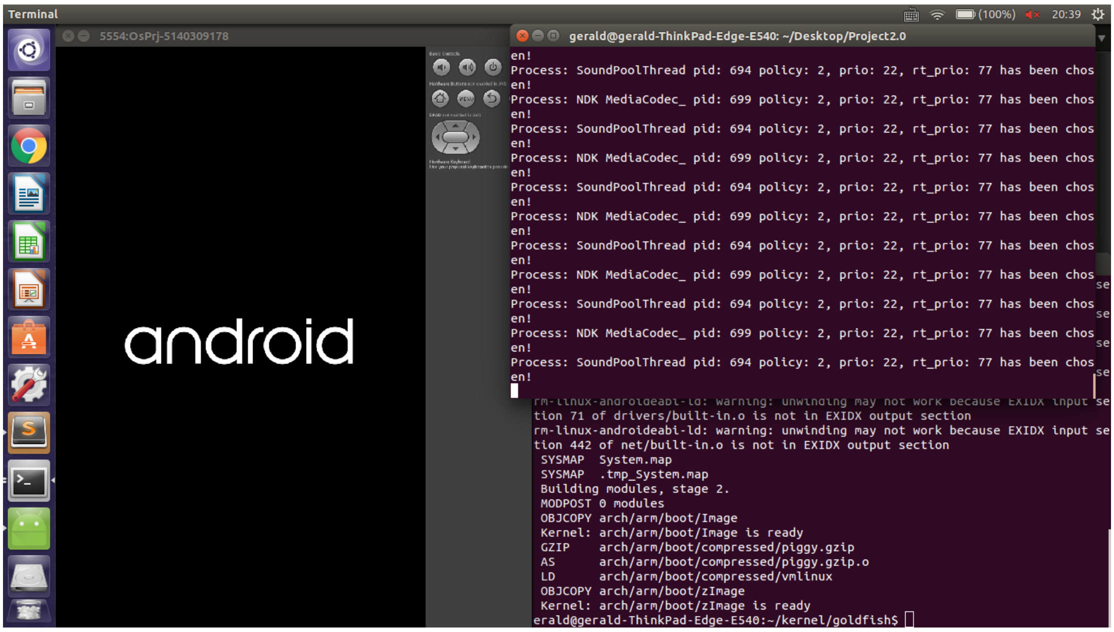
The second situation is that device is stuck by a process whose pid is over 600. The following picture indicates that device is stuck.



```
other fork pid:597, name:vold, prio:120, rt_prio:0
other fork pid:598, name:netd, prio:120, rt_prio:0
other fork pid:599, name:netd, prio:120, rt_prio:0
other fork pid:600, name:sdcard, prio:120, rt_prio:0
other fork pid:601, name:sdcard, prio:120, rt_prio:0
other fork pid:602, name:sdcard, prio:120, rt_prio:0
other fork pid:603, name:netd, prio:120, rt_prio:0
other fork pid:604, name:netd, prio:120, rt_prio:0
other fork pid:605, name:netd, prio:120, rt_prio:0
other fork pid:606, name:netd, prio:120, rt_prio:0
other fork pid:607, name:netd, prio:120, rt_prio:0
other fork pid:608, name:netd, prio:120, rt_prio:0
other fork pid:609, name:netd, prio:120, rt_prio:0
other fork pid:610, name:netd, prio:120, rt_prio:0
healthd: battery l=50 v=0 t=0.0 h=2 st=2 chg=a
other fork pid:611, name:netd, prio:120, rt_prio:0
other fork pid:612, name:netd, prio:120, rt_prio:0
other fork pid:613, name:netd, prio:120, rt_prio:0
other fork pid:614, name:netd, prio:120, rt_prio:0
healthd: battery l=50 v=0 t=0.0 h=2 st=2 chg=a
healthd: battery l=50 v=0 t=0.0 h=2 st=2 chg=a
healthd: battery l=50 v=0 t=0.0 h=2 st=2 chg=a
healthd: battery l=50 v=0 t=0.0 h=2 st=2 chg=a
```

6.2 Analysis

When I find this abnormal startup, I changed code and let the terminal show the process and why it is stuck. The following picture is the information on the terminal.



```
en!
Process: SoundPoolThread pid: 694 policy: 2, prio: 22, rt_prio: 77 has been chos
en!
Process: NDK MediaCodec_ pid: 699 policy: 2, prio: 22, rt_prio: 77 has been chos
en!
Process: SoundPoolThread pid: 694 policy: 2, prio: 22, rt_prio: 77 has been chos
en!
Process: NDK MediaCodec_ pid: 699 policy: 2, prio: 22, rt_prio: 77 has been chos
en!
Process: SoundPoolThread pid: 694 policy: 2, prio: 22, rt_prio: 77 has been chos
en!
Process: NDK MediaCodec_ pid: 699 policy: 2, prio: 22, rt_prio: 77 has been chos
en!
Process: SoundPoolThread pid: 694 policy: 2, prio: 22, rt_prio: 77 has been chos
en!
Process: NDK MediaCodec_ pid: 699 policy: 2, prio: 22, rt_prio: 77 has been chos
en!
Process: SoundPoolThread pid: 694 policy: 2, prio: 22, rt_prio: 77 has been chos
en!
Process: NDK MediaCodec_ pid: 699 policy: 2, prio: 22, rt_prio: 77 has been chos
en!
Process: SoundPoolThread pid: 694 policy: 2, prio: 22, rt_prio: 77 has been chos
en!
```

```
rm-linux-androideabi-ld: warning: unwinding may not work because EXIDX input se
tion 71 of drivers/built-in.o is not in EXIDX output section
rm-linux-androideabi-ld: warning: unwinding may not work because EXIDX input se
tion 442 of net/built-in.o is not in EXIDX output section
 SYSMAP  System.map
 SYSMAP  .tmp_System.map
 Building modules, stage 2.
 MODPOST 0 modules
 OBJCOPY arch/arm/boot/Image
 Kernel: arch/arm/boot/Image is ready
 GZIP    arch/arm/boot/compressed/piggy.gzip
 AS      arch/arm/boot/compressed/piggy.gzip.o
 LD      arch/arm/boot/compressed/vmlinux
 OBJCOPY arch/arm/boot/zImage
 Kernel: arch/arm/boot/zImage is ready
erald@gerald-ThinkPad-Edge-E540:~/kernel/goldfish$
```

Through the screenshot, we can find that CPU pick some different process as its next process. But they have the common priority. So the reason for that situation is that CPU causes starvation because CPU pick next process based on priority rather randomly picking. So when I finish the codes about randomly picking, the problem is solved.

7. Acknowledgement

My deepest gratitude goes first and foremost to Professor Fan Wu , my teacher for Operating System, for his teaching in this term.

Then I wish to express my sincere thanks to teaching assistant BO Wang and Jianping Xie for solving my problems patiently, whose generous help have made possible the successful running of my program.

In these two project, I have leant some knowledge beyond textbooks and have a good command of the ability to read English material and some English website such as stackoverflow.

Thanks are also due to a number of my friends who generously give me the benefit of their tremendous suggestions.

Hope this only a begin for me to my road for computer science.