Achieving Fast Boot Time and Efficient I/O Multiplexing for Image grabber in Embedded Linux System

Shiuh-Ku Weng[#], Yen-Ju Lin[*]

[#*]*Department of Information Engineering, Chung-Cheng Institute of Technology, National Defense University*

[#]*skw@ndu.edu.tw*

[*]*andrea_a267@yahoo.com.tw*

Abstract—**The purpose of this paper is to reduce system boot time and to improve I/O multiplexing performance for image player of the embedded Linux system. The boot time is reduced by bypassing some redundant codes. About the I/O multiplexing, the performance comparison between the select, poll and epoll system calls is made. The experimental results show that using the epoll I/O multiplexing in implementation of image player is able to get higher performance efficiently.**

Keywords—Embedded Linux system, I/O multiplexing

I. Introduction

The advance of computing technology has made the Internet of Things (IoT) devices own many functions possibly. Many IoTs embedded microprocessors with operating systems allow them to be flexible and to meet a wide range of end-user needs. The embedded systems have become popular in our daily life.

The key characteristic of the embedded system is dedicated to handle a particular task. Owing to the embedded system dedicated to specific tasks, the design methods can be optimized by increasing the reliability and performance [1][2].

A real-time system is determined to complete its functions in a period of time and makes the right response to the event system. Usually, in a real-time system, the I/O is a bottleneck. Accordingly, it is important to get the efficient I/O multiplexing to improve performance of real-time systems.

II. Reducing the Boot Time of Linux

By analyzing the boot process and the architecture of the Linux kernel, some methods are proposed to speed up the boot time embedded systems.

During U-Boot [3] in initial processes, the initialization of console device is separated into two functions: *console_init_f* and *console_init_r*. After executing the two functions sequentially, the console device will be initialized as a fully console device. However, we don't need U-Boot to provide a fully console device during boot. Therefore fully initialization of console device is redundant. After realizing the U-Boot source code and doing experiment, we know that the function *console_init_r* is useless during boot. Therefore, we skip the execution of *console_init_r*. The execution time of *console_init_r* can be saved. Although we skip the execution of *console_init_r*, U-boot is successful and the output messages can be shown by console after the function *console_init_f* done in the first stage initialization of console.

U-Boot provides some functions to print the information of devices; the information is useful during development and debug. Therefore, in the booting period, most of device will spend much time in initialization and output information on the console by serial port. We made the environment to be silent by changing the environment parameter (set env silent) to achieve a quiet console. [2]

In addition, in choosing the file system, according to the results in measuring the performance of several file systems, we find that the RAMdisk file system would be the best in the boot process. Therefore, the RAMdisk is adopted in our system to speed up boot time.

III. The Evaluation of I/O Multiplexing Mode

In Linux systems, there are serval I/O multiplexing mode system calls which are *select()*, *poll()* and *epoll()* (event *po*ll). *epoll()* is initially introduced in Linux 2.5.44 of kernel mainline [3]. Their functions are to monitor multiple file descriptors to watch if I/O is possible on any of them or not.

Select() is almost supported in all platforms. Its advantage is that it's also supported in cross-platform. But, its drawback is that *Select()* in one process can only monitor 1024 file descriptors maximally. Furthermore, as the number of the file descriptors increases, the copy time will grow since that it requires repeating to copy all of file descriptors into the kernel evens which the file descriptors have been already in the kernel. The image grabber system (v4l2grab Version 0.3) [5] of the Linux V4L2 [6] is still using this mode.

In the functions, *poll()* and *select*() have no difference. But, *poll()* is not designed to limit the maximum file descriptor. The drawback of *poll()* is the same with the *select*() that the copy time gains rapidly as the number of file descriptors is large.

To make comparison between *epoll()* and the previous two system calls, *epoll()* has much more advantages than those of *select()* and *poll()*. The most important one is that *epoll()* saves the copy time which the file descriptors have kept already in the kernel. It also only informs those file descriptors which are ready. While *epoll_wait( )* is being used to get those file descriptors which are ready, the number of the file descriptors instead of the actual descriptors will be got. Therefore, by using the *epoll()* to obtain the corresponding number of file descriptors, it will eliminate the copy time while using the system call for those file descriptors. In addition, *epoll()* uses the *epoll_ctl()* in advance to register a file descriptor. Once a file descriptor is ready, the kernel will quickly start the file descriptor and the kernel will get informed while the process is calling the *epoll_wait( )*. However, in *select()* and *poll()*, the process will only have the method which kernel scans for all of the file descriptors which are monitored.

From the above analysis, obviously, the data structure and algorithm of *epoll()* are more streamlined than *select()* and *poll()*.

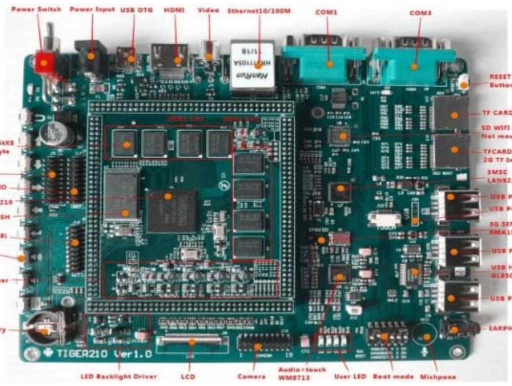


Fig. 1 The developing board – QT210

IV. The Experimental Results

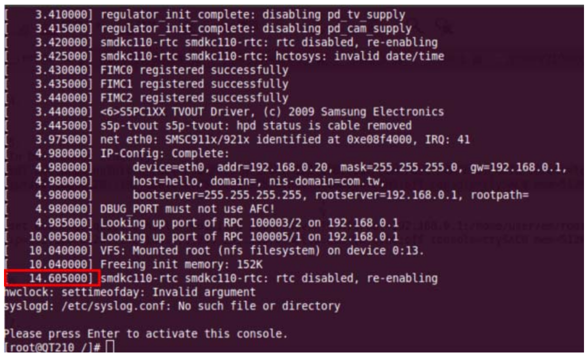
The QT210 developing board [7] is used as our embedded system. The appearance of the board is shown in Fig. 1. It is the developing board with the LCD monitor removed. The specifications of the hardware of the developing board are shown in Table 1.

To achieve the fast boot time, Fig 2 and 3 demonstrate the original boot time and the result of the fast one, respectively. The original boot time costs about 14.6 seconds, however, the fast one does it only in 0.86 seconds. The result gets about 17-time performance improved.

About the I/O multiplexing performance improvement, two systems are implemented to illustrate the results. In the first system, the image grabber system (v4l2grab Version 0.3) [5] of the Linux V4L2 [6] is modified with three different system calls to test their performance respectively. Fig. 4 displays the executing time of every system call for grabbing 1,000 images and saving them to disk. The performance of *epoll()* can be significantly distinguished from the other two ones. However, the *select()* is still used in the Linux V4L2 image grabber system.

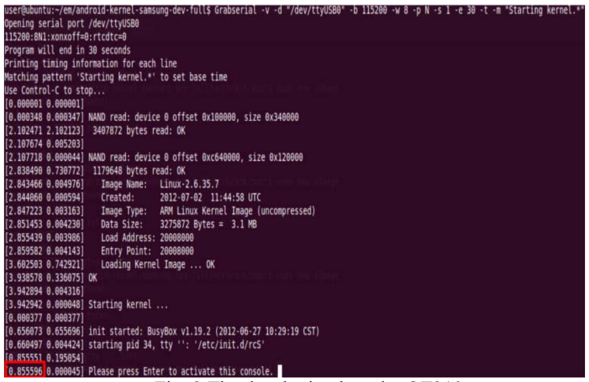
TABLE I
THE SPECIFICATIONS OF DEVELOPMENT BOARD

The type of CPU	The core of CPU	The frequency of CPU	RAM	Nand Flash
S5PV210	ARM Cortex-A8	1GHz	DDR2 1GB	256MB



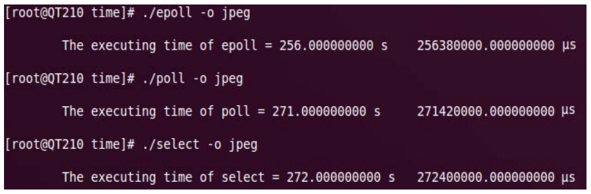
```
[    3.410000] regulator_init_complete: disabling pd_tv_supply
[    3.415000] regulator_init_complete: disabling pd_cam_supply
[    3.420000] smdkc110-rtc smdkc110-rtc: rtc disabled, re-enabling
[    3.425000] smdkc110-rtc smdkc110-rtc: hctosys: invalid date/time
[    3.430000] FIMC0 registered successfully
[    3.435000] FIMC1 registered successfully
[    3.440000] FIMC2 registered successfully
[    3.440000] <6>S5PC1XX TVOUT Driver, (c) 2009 Samsung Electronics
[    3.445000] s5p-tvout s5p-tvout: hpd status is cable removed
[    3.975000] net eth0: SMSC911x/921x identified at 0xe08f4000, IRQ: 41
[    4.980000] IP-Config: Complete:
[    4.980000]      device=eth0, addr=192.168.0.20, mask=255.255.255.0, gw=192.168.0.1,
[    4.980000]      host=hello, domain=, nis-domain=com.tw,
[    4.980000]      bootserver=255.255.255.255, rootserver=192.168.0.1, rootpath=
[    4.980000] DBUG_PORT must not use AFC!
[    4.985000] Looking up port of RPC 100003/2 on 192.168.0.1
[   10.005000] Looking up port of RPC 100005/1 on 192.168.0.1
[   10.040000] VFS: Mounted root (nfs filesystem) on device 0:13.
[   10.040000] Freeing init memory: 152K
[   14.605000] smdkc110-rtc smdkc110-rtc: rtc disabled, re-enabling
hwclock: settimeofday: Invalid argument
syslogd: /etc/syslog.conf: No such file or directory

Please press Enter to activate this console.
[root@QT210 /]# 
```

Fig. 2 The original boot time for QT210 developing board



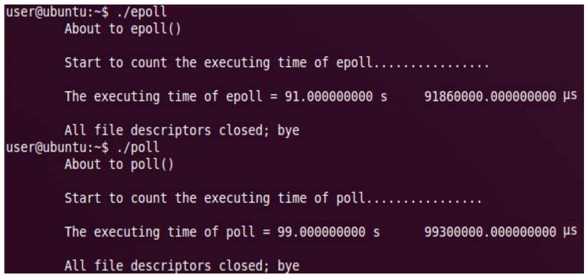
```
user@ubuntu:~/em/android-kernel-samsung-dev-full$ Grabserial -v -d "/dev/ttyUSB0" -b 115200 -w 8 -p N -s 1 -e 30 -t -m "Starting kernel.*"
Opening serial port /dev/ttyUSB0
115200:8N1:xonxoff=0:rtcdtc=0
Program will end in 30 seconds
Printing timing information for each line
Matching pattern 'Starting kernel.*' to set base time
Use Control-C to stop...
[0.000001 0.000001]
[0.000348 0.000347] NAND read: device 0 offset 0x100000, size 0x340000
[2.102471 2.102123]  3407872 bytes read: OK
[2.107674 0.005203]
[2.107718 0.000044] NAND read: device 0 offset 0xc640000, size 0x120000
[2.838490 0.730772]  1179648 bytes read: OK
[2.843466 0.004976]    Image Name:   Linux-2.6.35.7
[2.844060 0.000594]    Created:      2012-07-02  11:44:58 UTC
[2.847223 0.003163]    Image Type:   ARM Linux Kernel Image (uncompressed)
[2.851453 0.004230]    Data Size:    3275872 Bytes =  3.1 MB
[2.855439 0.003986]    Load Address: 20008000
[2.859582 0.004143]    Entry Point:  20008000
[3.602503 0.742921]    Loading Kernel Image ... OK
[3.938578 0.336075] OK
[3.942894 0.004316]
[3.942942 0.000048] Starting kernel ...
[0.000377 0.000377]
[0.656073 0.655696] init started: BusyBox v1.19.2 (2012-06-27 10:29:19 CST)
[0.660497 0.004424] starting pid 34, tty '': '/etc/init.d/rcS'
[0.855551 0.195054]
[0.855596 0.000045] Please press Enter to activate this console.
```

Fig. 3 The developing board – QT210



```
[root@QT210 time]# ./epoll -o jpeg

        The executing time of epoll = 256.000000000 s    256380000.000000000 µs

[root@QT210 time]# ./poll -o jpeg

        The executing time of poll = 271.000000000 s     271420000.000000000 µs

[root@QT210 time]# ./select -o jpeg

        The executing time of select = 272.000000000 s   272400000.000000000 µs
```

Fig. 4 The experimental results of testing the image grabbers with the *epoll()*, *poll()* and *select()* system calls



```
user@ubuntu:~$ ./epoll
        About to epoll()

        Start to count the executing time of epoll................

        The executing time of epoll = 91.000000000 s     91860000.000000000 µs

        All file descriptors closed; bye
user@ubuntu:~$ ./poll
        About to poll()

        Start to count the executing time of poll................

        The executing time of poll = 99.000000000 s      99300000.000000000 µs

        All file descriptors closed; bye
```

Fig. 5 The experimental results of testing the image grabbers with the *epoll()*, *poll()* and *select()* system calls

The second one is designed to handle 20,000 image file descriptors with the two different system calls, *epoll()* and *poll()*, respectively. Fig. 5 shows the test results of opening 20,000 image files for monitoring at the same time. The *select()* system call is not listed since the maximal number of the file descriptors it can handle is set in 1024. The experiment is using 20,000 file descriptors. It exceeds the limit. According to the above experimental results, the *epoll( )* has the best performance. The Table 2 lists the comparison.

TABLE IIIII
THE COMPARED TABLE OF MULTIPLEXING MODE SYSTEM CALLS – *EPOLL()*, *POLL()* AND *SELECT()*

Experiment / I/O multiplexing mode system	20,000 file descriptors	1,000 images captured and saved to disk
epoll()	91.86 Seconds	256.38 Seconds
poll()	99 Seconds	271.42 Seconds
select()	* Null	272.4 Seconds

*The maximal number of the file descriptors is 1024.

V. CONCLUSIONS

This study is mainly aimed at achieving fast boot time of the embedded system and analysing the performance of I/O multiplexing mode for implementing a grabber image system.

From this experiment, the *epoll()* gets the best performance in comparison with *select()* and *poll()*. Since *epoll()*is designed by several mechanisms and data structure, it is able to get the performance improved. Especially, in an image processing system, it frequently required handling a lot of image frames. To use an efficient I/O multiplexing mode is useful to enhance the system performance.

REFERENCES

[1] G. Singh, K. Bipin, and R. Dhawan, "Optimizing the Boot Time of Android on Embedded Systems," Proc. IEEE Int'l Symp. Consumer Electronics (ISCE), 2011, pp. 503–508.

[2] X. Yang, N. Sang and J. Alves-Foss, 'Shortening the Boot Time of Android OS', IEEE Computer, Vol. 47, No. 7, pp. 53–58, 2014.

[3] 'U-Boot Development', https://www.denx.de/wiki/U-Boot, accessed September 2018.

[4] Robert Love, Linux Kernel Development 3rd Edition, Pearson Education Inc., 2010.

[5] 'v4l2grab Version 0.3', https://github.com/twam/v4l2grab, accessed May. 2018.

[6] ' Video4Linux', https://www.linuxtv.org/docs.php, accessed May. 2018.

[7] 'QT210', http://twarm.com/commerce/product_info.php?Path=105_79_153&products_id=1632, accessed August 2018.