Improvement on Buffer Management Strategy to Enhance System Performance

Chen-Ying Wang，Shyh-In Hwang，Te-neng Liu
Institute of Computer Science and Engineering，Yuan-Ze University
fjdi.rod@msa.hinet.net，shyhin@saturn.yzu.edu.tw，earthman@mmlab.cse.yzu.edu.tw

Abstract

This paper proposed the issues of system performance and power consumption that come from the management of Direct Memory Access（DMA） Buffer Resource via the "up" Semaphore release mechanism in Linux Kernel and submitted the "up_trigger Method" to improve the existing problems.

"up_trigger Method" is a new way to manage the Semaphores in Linux Kernel that can reduce the Context Switching（CS）to improve the system performance for buffer management strategy of the embedded system. Besides, it can reduce the power consumptions due to the decreasing of Context Switching.

1.Introduction

Two important missions for Embedded Systems are performance and power consumptions. The portable feature of embedded system limits the product specification, especially; the size of product is concerned for carry. Because of this issue, the component performance of embedded system is less than the PC systems'. And, it comes out a difficult mission of power consumption for product development due to the portable issue.

We will probe into the defect of DMA buffer managed via the "up" Semaphore release mechanism in Linux Kernel（hereafter called "Kernel up Method"）based on Linux-2.6.10 in the platform of TI Omap1710 to find out the way for system performance improvement and power consumptions reduction.

To observe the behavior of DMA buffer management by Semaphore in Kernel, the audio playback will be used. From Figure 1, it divides into 2 properties: CPU and DMA. It obtains the result, as Table 1 due to the executing speed of CPU is much faster than DAM transferring the audio data to audio chip. Table 1 shows each time about the data written into Kernel DMA buffer by executing of audio playback. The testing conditions are : (1).Sample Rate 44100 Hz，(2).Channel Number 2，(3).Bit Per Sample 16，(4).DMA Buffer Number 8，(5).DMA Buffer Size 1024 Bytes.

From Table 1, the test result tells it gains Semaphore from 1st to 8th of Audio Playback executing. The spend time is "Buffer Copy Latency", as Figure 1 showed. Since the 9th time, audio playback task goes to sleep due to it cannot acquire Semaphore. Then, it will be waked up by DMA interrupt to release Semaphore after the data of DMA buffer is moved to audio chip by DMA controller. It contains the "Buffer Copy Latency" and task sleep times. We infer the usage of DMA buffer will be as Figure 2 showed when executes Audio Playback and we learn that it is not an efficient method due to it causes the system CS after DMA buffer finish the audio data transferring each time, and it will also increase power consumptions during CS. For these two issues,

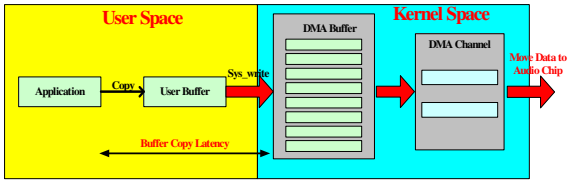


Figure1. Audio Playback Behavior

we submit the "up_trigger Method" to improve them.

Times	Sec
1	0.000124000
2	0.000054000
3	0.000048000
4	0.000052000
5	0.000043000
6	0.000041000
7	0.000040000
8	0.000046000
9	0.005581000
10	0.005734000
11	0.005783000
12	0.005796000
13	0.005792000

Write Buffer Time

(Write Buffer Time + Sleep Time)
DMA Buffer Write Time
+ Play 1024 Bytes Spend Time
≅0.0058 Sec

Table 1. Audio Playback Timing

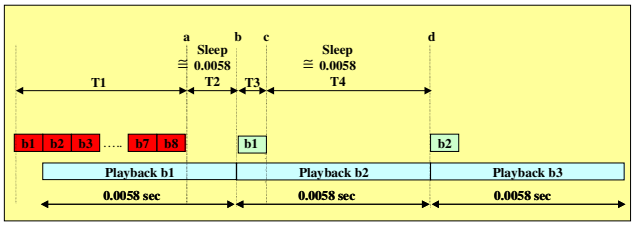


Figure2. Audio Playback DMA Buffer Usage

2.up_trigger Method

We hereby propose the "up_trigger Method" to reduce CS for system performance enhancement and to manage DMA buffer more flexibly.

Figure 3 shows the "up_trigger Method" will accumulate the numbers and release more Semaphores each time. Task can write data to DMA buffer as same numbers of Semaphores getting after it wakes up. Comparing to the " Kernel up Method " , it will decrease the times of CS. For the "up_trigger Method", there are several parts need to be amended and added in the Linux Kernel.

(1) Add two fields for trigger_level and trigger_count in the "struct semaphore". At first, trigger_level will be setting from 0 to (DMA number-2); trigger_count will be used to accumulate Semaphore when it arrives to (trigger_level+1). Then, task will be waked up and get Semaphores as same numbers of (trigger_level+1). Comparing to the " Kernel up Method " , the task can write (trigger_level+1) number of DMA buffer more than one. See Figure 3 for details.

(2) Add sema_init_trigger function to set up the requested trigger level and trigger count.

(3) Add up_trigger function to replace " Kernel up Method" .

The #CS (Context Switches) relationships between "up_trigger Method" and " Kernel up Method" can be defined as Equation (1).

CS(up) : CS (up_trigger) = 1 : (trigger_level+1)..(1)

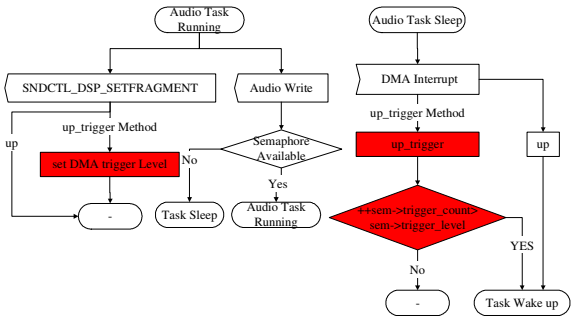


Figure 3. "up_trigger Method" SDL Flow

2.1 up_trigger Level 3

The "up_trigger Method" trigger_level=3 is submitted to explain the relationship between audio task and DMA buffer when executes audio playback. The testing results are as Table 2 showed. The "up_trigger Method" will accumulate 4 times of Semaphore when the trigger_level=3, the task in sleep will be around 4 times period of "Kernel up Method" (called Long Sleep), and it can write 4 DMA buffer in each Wake up. Figure 4 is the relationship between " Kernel up Method" and "up_trigger Method" trigger_level=3, it shows the usage of DMA buffer in details. The main difference is "up_trigger Method" can save 3/4 #CS than " Kernel up Method" to let other task keep going to improve the system performance. Besides, it can lower the power consumption due to the reduction of #CS. For the Power Saving, the efficiency is not good than the popular Shutdown Mechanism [2][3][4] and Kernel Scheduler [1] [5] caused they are based on $P \propto V^2 f$ theory. As to the power saving which "up_trigger Method" proposed is to reduce Task #CS then lower the switch between task's stack and register. So, the effect of power saving is another benefit except performance improvement.

Times	Sec	
1	0.000117000	Write Buffer Time
2	0.000049000	
3	0.000043000	
4	0.000043000	
5	0.000041000	
6	0.000042000	
7	0.000042000	
8	0.000045000	
9	0.022964999	Trigger level = 3
10	0.000052000	
11	0.000048000	
12	0.000053000	
13	0.023016000	Write Buffer Time + Long Sleep Time = (DMA Buffer Write Time + 4*Play 1024 bytes spend Time) ≅ 0.0058 * 4 = 0.0232 Long Sleep = Sleep * 4
14	0.000049000	

Table 2. up_trigger level 3 Audio Playback Timing

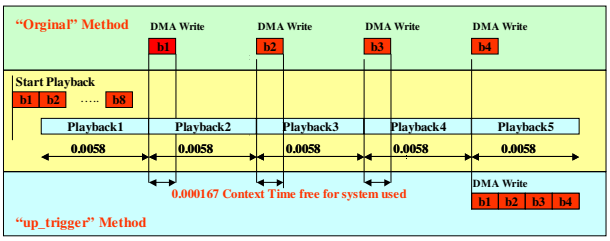


Figure 4.Relationship Between "Original Method" and "up_trigger Method Level =3" Mechanism

2.2 Trigger Level and Task Deadline

Section 2.1 is a discussion on trigger_level=3 to verify the difference of execution between "up_trigger Method" and "Kernel up Method". In fact, the Trigger level of "up_trigger Method" could be adjusted dynamically. For example, the range of trigger_level is from 0 to (the number of DMA Buffer – 2), as Figure 5 shows the trigger_level is from 0 to 6. When the system is busy, it can lower the trigger_level to prevent any miss deadline if the task cannot wake up on time; vice versa, it can raise the trigger_level to reduce the numbers of CS for performance improvement when the system is not busy.

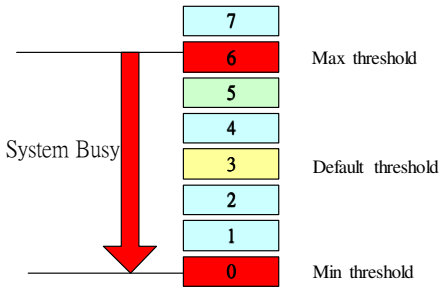


Figure 5. trigger_level and System Status

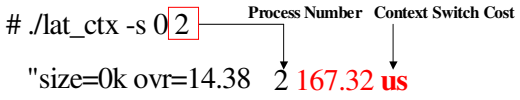
3.Performance Evaluation Method

In Section 3, it's an introduction for the evaluation of system performance cost. The difference of performance and power consumption between "up_trigger Method" and " Kernel up Method" will be implemented and be verified based on the Omap1710 platform.

3.1 Context Switching Cost Benchmark

We will use the "lat_ctx" of Linux Open Source lmbench to measure the CS Cost. The main working theory is as Figure 6 (refer to (7)).The lat_ctx will fork "N" of child process from parent and combine together through "pipe-ring". Parent will send "token" via "pipe-ring" to forward it in each process sequentially. All processes are in sleep except it received token. After last child task received token and send to parent, the cost of CS will be calculated by subtracting the needed transaction time.

We can run lat_ctx benchmark as under showed. The testing frequency is 192MHz, and it takes 167.32us for CS.



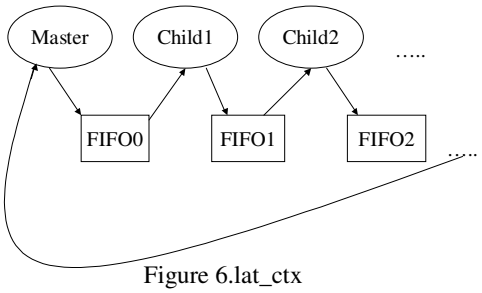


Figure 6.lat_ctx

3.2 Context Switching Number

The numbers of CS can be got from /proc/stat. We can cat it in console mode as below showed.

```
# cat /proc/stat
ctxt xxxxx
⇩
rq->nr_switches
```

To open /proc/stat file, read the file context and find out string of "ctxt", the numbers of System CS will be learned after the string of "ctxt".

3.3 Buffer Copy Latency Cost

It is not objective enough due to the " Kernel up Method" only can get the "Buffer Copy Latency" cost from the first 8th times, as Table 1 showed. So, we defined the "Buffer Copy Latency" from the average executing numbers of times. As Table 3 shows, it is "up_trigger Method" trigger_level=3 subtracting the timing of task long Sleep, i.e. the result of 45.79us is from the average of 100 times.

Times	Sec
1	0.000117000
2	0.000049000
3	0.000043000
4	0.000043000
5	0.000041000
6	0.000042000
7	0.000042000
8	0.000045000
9	0.022964999
10	0.000052000
11	0.000048000
12	0.000053000
13	0.023016000
14	0.000049000

Task long Sleep Discard

Task long Sleep Discard

+

Table 3. Buffer Copy Latency Cost

4.Implementation

Following the test architecture as Figure 7 shows, it can measure the difference of performance and power consumption between "up_trigger Method" and "Kernel up Method". Program of "Start" will run first and then forks 3 child. The functions of child task are as under:

(1) Playback：

Playback Task has the function to start the "up_trigger Method" and "Calculating Buffer Copy Latency"…etc.

(2) get_ctx：

Divide the length of timing into 60 equal parts, each part is period of timeout. It will get the numbers of CS from the file of /proc/stat when times out.

(3) File_rw：

Test the difference of system performance between "up_trigger Method" and "Kernel up Method".

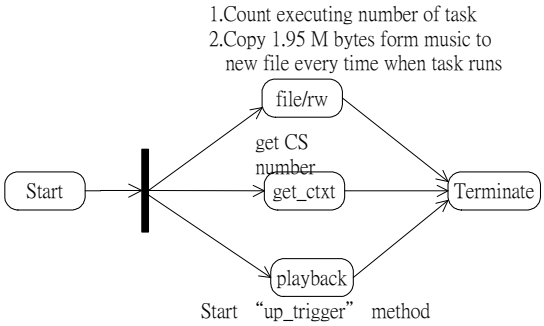


Figure 7.Testing Architecture

4.1 Performance Testing

The performance testing divided into 3 different conditions: (1) Pure Kernel Running, (2) Kernel up Method, (3) up_trigger Method Level = 3. The results are as Figure 8 shows.

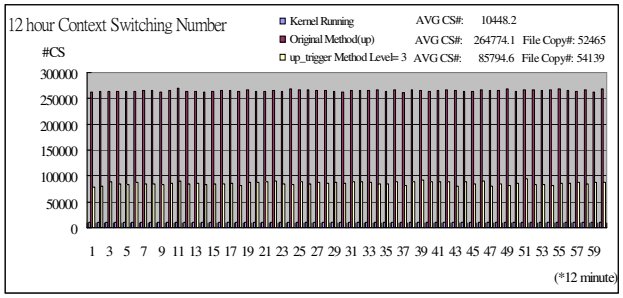


Figure 8.#CS & File Copy testing

Kernel up Method total #CS：16513340

up_trigger Method Level total #CS ： 5774571

4.2 Power Consumption Testing

Based on Figure 7 test architecture, the testing result of power consumption in different conditions are as Figure 9 ~ Figure 11 show.

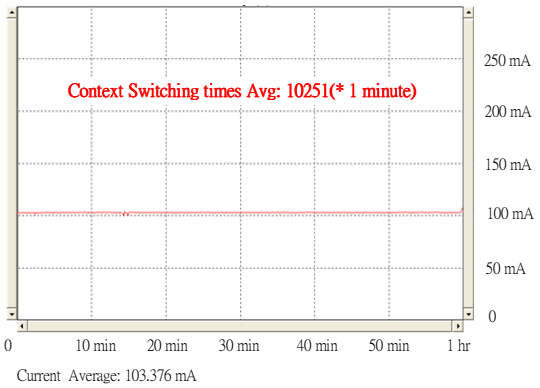


Figure 9.File_rw + get_ctxt Power Consumption

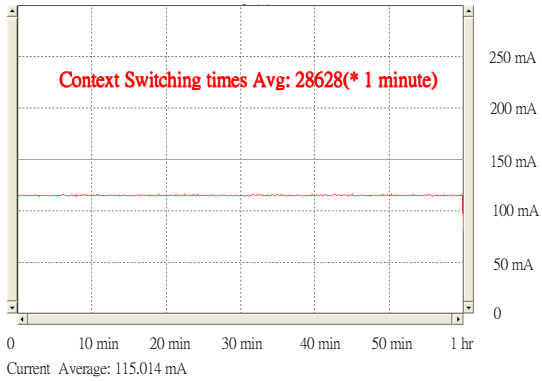


Figure 10. Kernel up Method Power Consumption

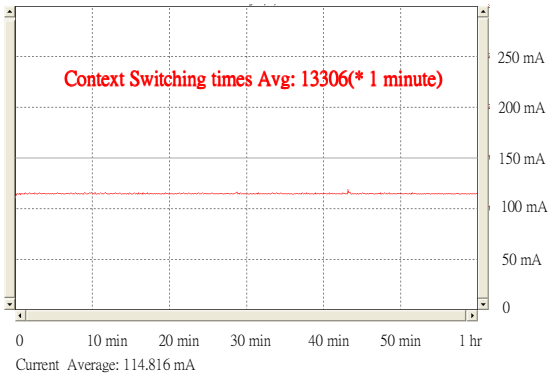


Figure 11. up_trigger Method Level =3 Power Consumption

4.3 Testing Result

Regarding the performance and power consumption of "up_trigger Method" trigger_level=3 and "Kernel up Method", it can be discussed as followings.

(1) #CS(Context Switching) reduction

From Section4.1, it describes the #CS reduce 65% when executes 12 hours testing (see Figure 8)
(16513340 – 5774571) / 16513340 = 65 %

(2) Performance Enhancement

From File Copy#, see Figure 8
up_trigger Method：Kernel up Method＝
54139：52465 (times)
In the same execution period, the "up_trigger Method" increase 1674 times file copy to improve the system performance: (54139-52465) / 52465 = 3.19 %

(3) System Power Saving

From Figure 9~ Figure 11
Kernel up Method= (115.014 - 103.376) = 11.664
up_trigger level 3 = (114.816 - 103.376) = 11.44
Power Saving：(11.664 - 11.44)/ 11.664 = 1.92%

5.Conclusion

The "up_trigger Method" we submitted has outstanding aid for system performance improvement. Even though many researches based on $P \propto V^2 f$ are better than the way we proposed for the power saving. But, "up_trigger Method" doesn't need any extra effort to get the effect via reducing the quantity of CS.

The "up_trigger Method" compares with "Kernel up Method", it has following advantages:

1. System performance enhancement
2. Power consumption decrease
3. Easy implement for "up_trigger Method" (see Section 2 for details)

References

〔1〕 David Tam,Winnie Tsang,Catalin Drula. "Dynamic Voltage Scaling in Mobile Devices", CSC2228 Project Final Report December 15, 2003.

〔2〕 Luca Benini, Robin Hodgson, Polly Siege1. "System-level Power Estimation And Optimization", in Proc. Symp. Low Power Electronics, pages 173-178, Oct. 1998.

〔3〕 Xue Wu. "Dynamic Power Management Technique and Its Application on A Multiprocessor Architecture". http://www.cs.umd.edu/~wu/paper/DPM_Final.doc

〔4〕 Wanghong Yuan，Klara Nahrstedt. "Practical Voltage Scaling for Mobile Multimedia Devices", in Proc. of ACM Multimedia 2004, New York, NY, October, 2004.

〔5〕 Wanghong Yuan, Klara Nahrstedt. "Energy-Efficient Soft Real-Time CPU Scheduling for Mobile Multimedia Systems", in Proc. of 19th ACM Symposium on Operating Systems Principles (SOSP'03), Bolton Landing, NY, October, 2003.

〔6〕 Understanding the Linux 2.6.8.1 CPU Scheduler.

〔7〕 Hyok-Sung Choi, Hee-Chul Yun. "Context Switchinging and IPC Performance Comparison between uClinux and Linux on the ARM9 based Processor" ,http://opensrc.sec.samsung.com/document/uc-linux-04_sait.pdf.

〔8〕 Murali Vilayannur , Robert B. Ross, Philip H. Carns , Rajeev Thakur , Anand Sivasubramaniam , Mahmut Kandemir. "On the Performance of the POSIX I/O Interface to PVFS", in Parallel, Distributed and Network-Based Processing, 2004. Proceedings. 12th Euromicro Conference on, pages 332- 339, 11-13 Feb. 2004.

〔9〕 SuSE Linux AG, Nuremberg, Germany. "Sound Systems on Linux: From the Past To the Future", Linux 2003 Conference, Edinburgh, Scotland.

〔10〕 Selim Gurun , Chandra Krintz. "AutoDVS: An Automatic, General Purpose , Dynamic Clock Scheduling System for HandHeld Devices" , in Proceedings of the Fifth ACM International Conference on Embedded Software (EMSOFT'05), pages 218-226, September 2005.

〔11〕 MontaVista Linux Dynamic Power Management for OMAP 161x GSM/GPRS Software Development Platform

〔12〕 Dynamic Power Management for Embedded Systems IBM and MontaVista Software, White Paper, November 2002. http://www.mvista.com/

〔13〕 Bishop Brock, Karthick Rajamani. "Dynamic Power Management for Embedded Systems", in Proceedings of the IEEE Int'l SoC Conference （SoCC 2003）, September 2003.

〔14〕 Yung-Hsiang Lu, Giovanni De Micheli. "Comparing System-Level Power Management Policies" , IEEE Design Test Computers,2001,18(2):10-19.

〔15〕 Bill Weinberg. "MontaVista Software Building Intelligent Devices with MontaVista Linux Consumer Electronics Edition". http://www.linuxpundit.com/cv/docs/wp_cee.pdf