Analyzing RTLinux/GPL Source Code for Education

She Kairui, Bai Shuwei, Zhou Qingguo, Nicholas Mc Guire, and Li Lian
Distributed and Embedded Systems Lab
School of Information Science and Engineering
Lanzhou University
Tianshui South Road 222,Lanzhou,P.R.China
{shekr06,baishw06}@lzu.cn
{zhouqg,mcguire}@lzu.edu.cn

Abstract

With decades development, RTLinux/GPL was widespread applied to both scientific research realm and industry. RTLinux/GPL has made a grate success in these realm, But has little affection in education, especially the undergraduate education. Most of the RTLinux/GPL documentation are focus on the practicability, other than emphasizing the basic RealTime Operatiing System theory. So RTLinux/GPL is very hard to applied to education realm. This article aimed at RTOS education, containing real time principle of RTLinux/GPL and the RTLinux/GPL modules implementation in detail. It would make the realtime operating system learning process easier with both theory and real source code analyze, and would attract more people to interested in RTLinux/GPL. This article presented RTLinux/GPL principle and implementation details, containning realtime clock, interrupt, realtime schedule strategy. We hope this article can help people to apprehend RTOS, especially the RTLinux/GPL.

1 Introduction

Nowadays, application of RTOS(RealTime Operating System) has becoming more and more prevalent, resulting that RTOS takes more and more important role in industry. So the research and development of RTOS is very active. But RTOS has little affection in education, especially the undergraduate education, for lack of appropriate document and teaching material. The RTOS educational instruction only focus on the essential RealTime System principle, without abundant RealTime case study.

RTLinux/GPL is distributed under GPL, This feature is worthful, students can get RTLinux/GPL source code freely, even they can modify the source code. But most document of RTLinux/GPL are introductory or about Application Programming Interface, which with little corresponding principle, which makes newcomers shrink back at the sight.

In order to further spread the RTOS in education realm, we take the RTLinux/GPL as an example, we will explain the essential principle of RTOS,such as RealTime schedule algorithm, and the concrete implementation in RTLinux/GPL. This article covers RealTime Operating System Architecture, RTLinux/GPL implement principle, clock, timer, interrupt management, RealTime schedule algorithm(RM,EDF), and the implementation dissection of these mentioned concepts in RTLinux/GPL.

2 RealTime OS and RTLinux/GPL

Generally, RTOS refer to Operating System with certain real time resource schedule and communication capability[1]. According to the real time capability, RTOS can classified as Hard RTOS and Soft RTOS. In this article the referred RealTime always indicate Hard RealTime,except for particular declaration.

During the RTOS design phase, designer must consider one basic conflict requirement, on the one hand the custom anticipate the target system support hard real time capability, on the other hand they also want the the system provide abundant function and feature like desktop PC system. There exists two solution to dilemma: Expand the exists RTOS,

*This research was supported in part by Cold and Arid Regions Environment and Engineering Research Institution, Chinese Academy of Sciences

or make a general purpose operating system Real-Time capable via add certain software layer. this article take RTLinux/GPL as an example, explaining the second solution implementation in detail.[2]

As general purpose OS, Linux system is a time sharing OS based on time slide Round Robin algorithm, further more, Linux kernel is nonpreemptive. generally, Linux time resolution is 10ms, and not satisfy the hard real time requirement.

RTLinux/GPL is a typical dual-kernel, one is Linux kernel, which provide various features of general purpose OS, other one is RTLinux kernel, which support hard real time capability. Figure 1 illustrate the RTLinux/GPL architecture.

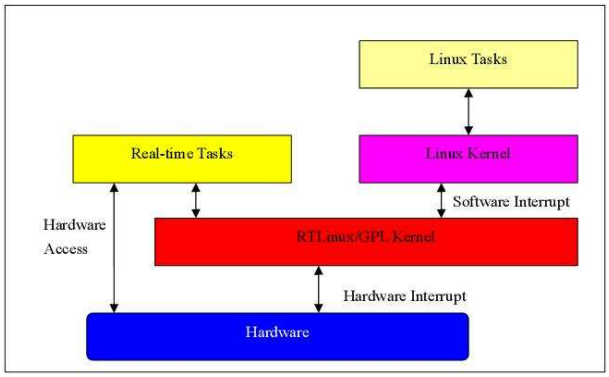

FIGURE 1: *RTLinux/GPL Runtime Model*

3 Clock and Timer

Clock is the hardware resource used for time management in computer. Timer is hardware or software facility that allows functions to be invoked at some future moment,after a given time interval has elapsed[3]. Generally speaking, hardware clock contain a hardware timer, but the only hardware timer is far from enough in multi-task system. So multi-task system need software timer to provide more timers.

This section will describe the RTLinux/GPL low level hardware clock management and soft timer management.

3.1 Clock

The current RTLinux/GPL version support two kinds of hardware timers, APIC and 8254, they are used for multi-processor system and uni-processor system respectively. RTLinux/GPL provide the identical clock control API to manage both hardware clock. We can use the API to achieve timer setting, read or write time,etc.

The RTLinux/GPL clock control APIs:

```
int init(struct rtl_clock *c);
void uninit(struct rtl_clock *c);
hrtime_t gethrtime(struct rtl_clock *c);
int sethrtime(struct rtl_clock *c, hrtime_t t);
int settimer(struct rtl_clock *c,
    hrtime_t interval);
int settimermode(struct rtl_clock *c, int mode);
void handler(struct pt_regs *r);
```

3.2 Timer

RTLinux/GPL can provide one or more soft timer for each real time task, and RTLinux/GPL use linked-list to management the soft timer[4]. Generally,all soft timer use the same low level hardware timer, operating system will select such soft timer in the linked-list based on certain timer schedule algorithm, then use the selected soft timer structure to set the hardware timer via timer control API. The Timer Interrupt will trigger task schedule at the special moment. How to operate the soft timer, such as create and destroy? RTLinux/GPL soft timer API implementation POSIX compliant.

Timer manage related API:

```
int timer_create(clockid_t clock_id,
    const struct sigevent *signal_specification,
    timer_t *timer_ptr);
int timer_gettime(timer_t timer_id,
    struct itimerspec *ts_set);
int timer_settime(timer_t timer_id, int flags,
    const struct itimerspec *new_setting,
    struct itimerspec *old\_setting);
int timer_delete(timer_t timer_id);
```

Figure 2 illustrate clock and timer hierarchy model.

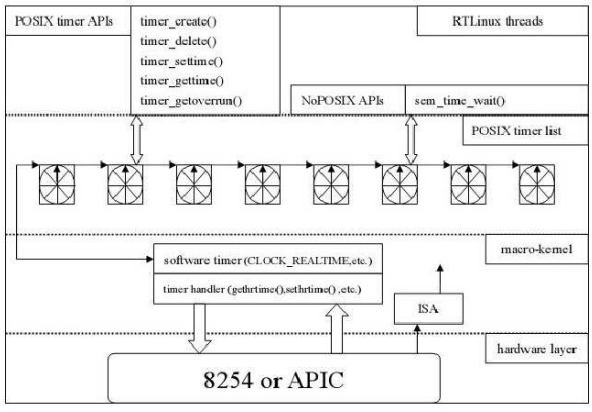

FIGURE 2: *Clock and Timer Hierarchy Model*

4 Interupt

In this section, we will start with Linux Interrupt, and then discuss RTLinux Interrupt implementation.

IBM compatible PC use two 8259A chips in cascade to make up of the PIC(Programmable Interrupt Controller), which can provide 15 IRQs. Each hardware device controller capable of issuing interrupt requests to the PIC, PIC can converts interrupt request into a corresponding interrupt vector and then store the vector. After PIC send a raised signal to the processor INTR pin, PIC will wait until CPU acknowledge. CPU would use this interrupt vector as index and get a corresponding interrupt service routine entry from IDT(Interrupt Description Table).

As for x86 architecture, CPU support 256 interrupt vector, IDT data structure is an array named idt_table, witch includes 256 entries.

interrupt initialization:

head.S is part of the uncompressed section of the vmlinuz image that is directly called by the bootloader - it is responsible for basic setup of the low level resources so that the hardware is actually accessable, it then calls the start_kernel symbol with is the entry point into the decompressed kerner prope.

call SYSTEM_NAME(start_kernel)

start_kernel is the compressed kernel entry, initialization of 8259A chips and interrupt gate is performed by the function call init_IRQ.

RTLinux/GPL running as modules after Linux kernel startup. RTLinux/GPL will takeover the Interrupt control when these modules was loded. RTLinux/GPL modules will patch the running linux kernel.

In linux each interrupt will come to the function common_interrupt, and then jump to do_IRQ(jmp do_IRQ). The primary role of patching is to change the jump destination to RTLinux interrupt entry(jmp rtl_intercept), so all interupt will be intercepted by RTLinux/GPL.

Figure 3[5] illustrate the RTLinux/GPL interrupt intercept

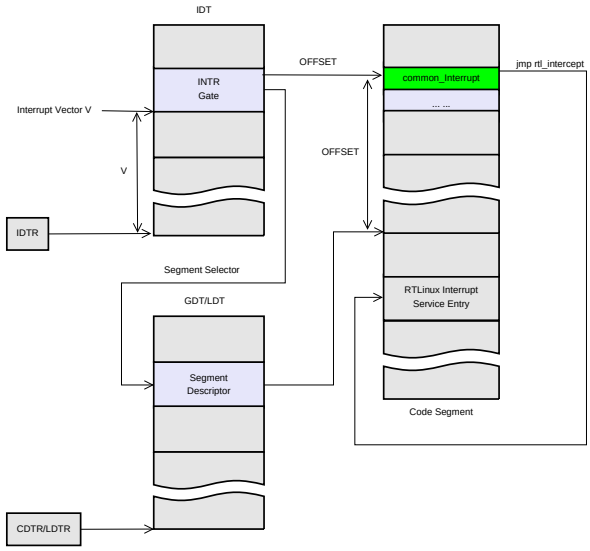

FIGURE 3: *RTLinux/GPL Interrupt Intercept*

RTLinux interrupt will be responsed immediately, as for Linux interrupt, if Linux interrupt is enable, Linux interrupt service routine will be called immediately, otherwise this Linux interrupt will be blocked until Linux interrupt is enabled. This process known as Soft Interrupt.

RTLinux interrupt API:

```
//add/remove real time interrupt handlers
int rtl_request_irq(unsigned int irq,
    unsigned int (*handler)(unsigned int irq,
        struct pt_regs *regs));
int rtl_free_irq(unsigned int irq);
//install/remove software interrupt handlers
int rtl_get_soft_irq(void (*handler)
    (int, void *, struct pt_regs *),
    const char * devname);
void rtl_free_soft_irq(unsigned int irq);
//schedule a Linux interrupt
void rtl_global_pend_irq(int irq);
```

The function schedule the interrupt specified by argument irq to be happen when the system enters the Linux mode.

5 Schedule Policies of Real-Time Operating System

Except for high resolution clock management and efficient interrupt process capability, task schedule policy is another critical factor directly affect the Real Time capability in the multi-task System. A multi-task System allows more than one task to be loaded into the executable memory at a time[6], the loaded

tasks share the CPU, so how to arrange the task execute sequence is important? We can choose appropriate schedule algorithm for corresponding requirement. The purpose of a real-time scheduling algorithm is to ensure critical timing constraints. RTLinux/GPL is a multi-task system with real time capability, and RTLinux/GPL implemented two priority based schedule algorithm,RM(Rate Monotonic) and EDF(Earliest Deadline First). RM schedule algorithm based on static priority, and EDF based on dynamic priority[7]. Static priority means each task was assigned a fixed priority. Generally, priority assigning was according to the task attribute, In RTLinux/GPL, realtime task always with higher priority than non-realtime task. Dynamic priority means the task priority is alterable in term of its resource demands, so dynamic priority schedule algorithm is more flexible for task schedule and resource assign.

5.1 RealTime Task, Priority, Preemptible

Before schedule policies presentation, I will explain several important concept firstly.

Real-Time task: The scheduler operational objects are real-time tasks, which require the tasks must be executed or finished at a given time(typically milli- or microseconds).

Priority: The priority is based on a predetermined assignment value, or importance to different types of tasks. In the RTLinux system, the Linux kernel has the lowest priority, and the real-time threads has the high priority(not less than 1000).

Preemptible: If the higher priority task could preempt CPU from the running task with lower priority by force, the system is preemptible, or non-preemptible. Such as, the old linux distributions are non-preemptible, but the RTLinux system is preemptible.

5.2 priority based Rate Monotonic algorithm

5.2.1 Rate Monotonic algorithm

Rate Monotonic algorithm based on fixed priority, system assign a priority for each task according to the task expected execution time, task with less execution time will assigned higher priority.

RM algorithm principle:

- schedule independent periodic task with fixed priority.
- priority assign policy: less cycle task will assign higher priority, assume cycle of non-periodic task is infinite.
- higher priority preemption.

following example would illustrate the the RM algorithm, considering three tasks: T1, T2, T3, T1 and T2 are periodic task, cycle of T1 is P1=5s, cycle of T2 is P2=10s, the task execution time is C1=C2=2s, T3 is non-periodic task, execution time is C3=5s, assume three tasks are ready at the same time, Figure 4 is the RM task schedule diagram.

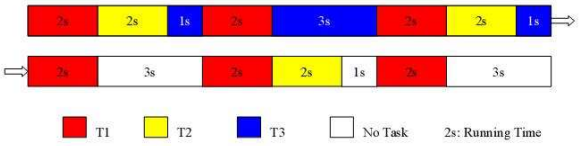

FIGURE 4: *RM Task Schedule Diagram*

The diagram reveal that the task with least cycle is T1, which was assigned highest priority. Task T1 was first scheduled before task T2 and T3, and T1 can preempt T3. T2 was scheduled after T1 and before T3, T3 was last scheduled.[8]

In order to work correctly, certain preemption must be satisfied :

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + ... + \frac{C_n}{T_n} \leq n(2^{1/2} - 1)$$

n is the total task number in the SystemCi($i \leq n$) stands for the longest execution time of task i, Ti($i \leq n$)stands for the cycle of task i.

5.3 RM algorithm implementation in RTLinux/GPL

considering following code fragment

```
257 if ((t->pending & ~t->blocked) &&
258  (!new_task || (cmp_prio(t, new_task)>0) ) ) {
259   #ifdef CONFIG_RTL_SRP
260    if (!current_sysceil ||
261      (cmp_preempt_level(&t->sched_param,
            current_sysceil) > 0))
262   #endif // CONFIG_RTL_SRP
263    new_task = t;
264  }
```

line 257 decide whether the process is blocked, line 258 select the task with highest priority. we don't care about the rest lines, which is resource storage style related.
the cmp_prio function:

```
51 static inline int cmp_prio(pthread_t A,
                             pthread_t B)
52 {
53 #ifdef CONFIG_RTL_SCHED_EDF
54 register int tmp;
55 if ( (tmp= (A->sched_param).sched_priority -
              B->sched_param).sched_priority) )
56 return tmp;
57 else
58 return ( (B->current_deadline) >
           (A->current_deadline) );
59 #else
60 return (A->sched_param).sched_priority >
         (B->sched_param).sched_priority;
61 \#endif
62 \}
```

line 53 to 59 EDF related, we will discuss it next section. considering line 60, if the return value is 0, then the task with highest priority will be selected , but context switch would not perform immediately. Scheduler will check the clock mode first, if clock mode is ONESHORT, scheduler will reset the hardware timer to issue interrupt at a specified moment, and then switch the context.

5.4 Earliest Deadline First algorithm

5.4.1 EDF algorithm

EDF schedule algorithm is based on dynamic priority. System assign a priority for each task according to the Deadline of the task dynamically, Task with earliest deadline will assigned highest priority[9]. In order to schedule correctly, certain preemption must be satisfied:

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + ... + \frac{C_n}{T_n} \leq 1$$

n is the total task number in the SystemCi($i \leq n$) stands for the longest execution time of task i, Ti($i \leq n$) stands for the cycle of task i.

Take an example, task T1, T2, T3, cycle: P1= 11s, P2=10s, P3=12s, execution time C1=4s, C2=5s, C3=5s. At moment t=0, each Deadline is: D1=P1-C1=7s, D2=P2-C2=5s, D3=P3-C3=10s, D2 < D1 < D3, so priority of task T2 is higher than T1 and T3, T2 will execute first. After task T2 finished, namely the moment t=4s, Deadline of each task is: D1=D1-t1=3s, D2=D2+ P2-C2=14s, D3=D3-t1=6s, now D1 < D3 < D2, so task T1 will be scheduled next, ...

Figure 5 illustrate the schedule flow:

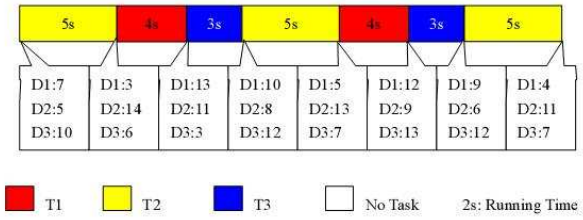

FIGURE 5: *EDF Schedule Algorithm*

5.4.2 EDF algorithm implementation in RTLinux/GPL

Each thread in RTLinux/GPL holds two deadline attribute: one is relative deadline(sched_deadline), other one is absolute deadline(current_deadline). RTLinux/GPL provide two function to set sched_deadline:

```
pthread_attr_setdeadline_np();
pthread_setdeadline_np();
```

there is no API to set the absolute deadline(current_deadline),
function pthread_make_periodic_np() will set the absolute deadline:

```
p->current_deadline = start_time +
    p->sched_deadline  p->period;
```

The core of RTLinux/GPL scheduler is function rtl_schedule(), which will search all the non-blocked thread, and the function call cmp_prio will select the task with highest priority and least absolute deadline(current_deadline).

if ((t->pending & t->blocked) &&(!new_task || (cmp_prio(t, new_task) > 0)))

```
static inline int cmp_prio(pthread_t A,
                           pthread_t B)
{
#ifdef CONFIG_RTL_SCHED_EDF
register int tmp;
if ( (tmp= (A->sched_param).sched_priority -
           (B->sched_param).sched_priority) )
return tmp;
else
return ( (B->current_deadline) >
         (A->current_deadline) );
#else
return (A->sched_param).sched_priority -
       (B->sched_param).sched_priority;
#endif
}
```

System will check the mode to decide whether need to call find_preemptor() to get the preempt time, and the use this time to set the timer

6 conclusion and prospection

This article focused on RTOS education, made an expatiation about RTOS principles, and take RTLinux/GPL as an Example, illustrate the concrete implementation. At present most of the content of RTOS education is about the essential RealTime System principle. It will make the RTOS learning process much easier via illustrate a typical RTOS implementation, and will promote the development of RTLinux/GPL indirectly. Due to time limitation,this article only describe most basic parts, and RealTime communication and process management need to be addressed.

References

[1] Victor Yodaiken and Michael Barabanov, 1997, *RTLinux Version TWO Design documentation about RTLinux in FSMLabs* `http://www.fsmlabs.com`h

[2] Der Herr Hofrat, 2002,*Introducing RTLinux/GPL*, p8

[3] Daniel P.Bovert & Marco Cesati, 2005, *Understanding the Linux Kernel*, O'Reilly & Associates,Inc., 0-596-00565-2.

[4] Yang Lifeng,*Embeded RTOS Development and Design*, `http://www.51kaifa.com/zxyd/read.php?ID=107`

[5] Mao Decao & Hu Ximing, 2001, *Linux Kernel Source Code Scene Analyze*, Zhejiang University Press, 7-308-02704-X/TP.209.

[6] Gary J.Nutt, 2000, *Operating Systems a modern perspective*, Addison Wesley Longman,Inc., 0-201-61251-8.

[7] Amit Choudhary, Nitin Shrivastav, Ramnath Venugopalan, 2002, *Implementation of EDF, PCEP and PIP in RT-Linux Under the guidance of Dr,Mueller*

[8] Patricia Balbastre, Ismael Ripoll, *Integrated Dynamic Priority Scheduler for RTLinux*

[9] Chu Fenmin, Dai Shenghua*Study of RTLinux Scheduling Policy*,MicroComputer Information, 2003, Issue 11.