Rockchip Linux 实时性能问题排查

文件标识：RK-KF-YF-A04

发布版本：V1.0.0

日期：2023-05-19

文件密级：□绝密 □秘密 ■内部资料 □公开

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园A区18号

网址： www.rock-chips.com

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： fae@rock-chips.com

前言

概述

本文主要举例介绍了Linux系统下实时性问题的排查，提供问题排查的思路和建议，帮助开发者更好地排查问题。

产品版本

芯片名称	内核版本
ALL	ALL

读者对象

本文档（本指南）主要适用于以下工程师：

技术支持工程师
软件开发工程师

修订记录

版本号	作者	修改日期	修改说明
V1.0.0	Hans Yang	2023-05-19	初始版本

目录

1. 实时性能介绍

Linux系统的实时性能指的是其对实时任务的响应和处理能力。实时任务是那些具有严格时间要求的任务，需要在特定的时间限制内完成，并提供准确和可预测的响应时间。

Linux实时性能的主要特征包括以下几个方面：

1. 硬实时性（Hard Real-Time）：硬实时性要求实时任务在严格的时间限制内完成，任何违反时间约束的情况都被认为是错误的。Linux的硬实时性是通过实时调度器和相关机制来实现的。
2. 软实时性（Soft Real-Time）：软实时性要求实时任务在大多数情况下能够按时完成，但偶尔的时间延迟可以被接受。Linux的软实时性是通过实时调度器的预测性能来实现的。
3. 可预测性：Linux系统的实时性能应具有可预测性，即实时任务的响应时间和执行时间应该是可预测的。这样可以使开发人员能够更好地规划任务和调度，以满足实时要求。
4. 低延迟：实时任务的响应时间应该尽可能地低。Linux系统通过减少上下文切换时间、优化调度算法等方式来降低延迟。
5. 实时性保证：Linux系统应提供一些机制来确保实时任务的实时性能。例如，可以使用优先级调度、亲和性调度、实时扩展等技术来确保实时任务得到适当的处理和调度。

需要注意的是，Linux系统并非一个专门设计用于硬实时任务的操作系统，而是一个通用的操作系统。尽管Linux在实时性能方面有一定的改进和增强，但它可能无法满足某些特定领域对硬实时性能的严格要求。对于对实时性能要求非常高的应用，可能需要考虑使用专门设计的实时操作系统。

以下我们就以一个串口读取实时性能的问题举例分析，进一步的对此类问题的分析提供方案和优化建议。

2. 实时性能问题分析

2.1 问题说明

2.1.1 问题描述

测试时遇到了RK3588与MCU串口通信帧数据存在较大延时，MCU每隔10ms（100Hz周期）向RK3588 UART0口发送一次数据，数据长度不定，RK3588以阻塞方式read串口(/dev/ttyS0)数据，测试实际的读取数据的时间间隔会>10+ms，在30ms左右。

2.1.2 测试拓扑

```
RK3588（UART0）<------(Baudrate:115200)------>(UART)MCU
```

2.1.3 问题影响

该问题导致RK3588无法及时收取到MCU所发送的数据，影响其业务功能正常工作。

2.2 问题排查思路

2.2.1 确定MCU发送数据的间隔时间

可以通过示波器或逻辑分析仪，抓取MCU数据发送的间隔时间，是否有大于10ms周期发送数据的情况。

2.2.1.1 实测分析

MCU以10ms周期间隔往串口发送数据，通过ttyS0中断间隔时间查看，MCU有概率地会超过13ms发送数据，此作为已知问题，MCU部分去排查。

2.2.2 确认MCU发送数据有无丢帧

可以通过对MCU发送的数据帧，加上帧序列序号，在接收端对接收的数据进行采集和校验，确认传输过程中有无丢帧。

2.2.2.1 实测分析

通过帧序列的校验，确认数据传输过程没有丢帧，这部分不存在问题。

2.2.3 确定串口中断的延迟

可以在串口中断的处理函数中统计间隔两次之间的中断延迟，来确定，是否中断本身有延迟。

举例伪代码逻辑参考（可加到串口中断处理函数内统计）：

```
static unsigned long prev_time;

irqreturn_t serial_interrupt(int irq, void *dev_id)
{
    unsigned long current_time = jiffies;

    unsigned long elapsed_time = current_time - prev_time;
```

```
    printk(KERN_INFO "Interrupt interval: %lu jiffies\n", elapsed_time);

    prev_time = current_time;

    // 处理串口中断

    return IRQ_HANDLED;
}
```

2.2.3.1 实测分析

串口两次中断间隔最长的不会超过11ms，这部分不存在问题。

2.2.4 确定串口中断到flush函数的时间

串口中断的调用流程如下：

```
1)drivers/tty/serial/8250/8250_port.c
-->serial8250_rx_chars()    #rx中断处理

2)kernel/drivers/tty/tty_buffer.c
-->tty_flip_buffer_push(&port->state->port) #调用Flush函数
    -->tty_schedule_flip(struct tty_port *port)
        -->flush_to_ldisc()     #起work_queue线程，调用flush_to_ldisc()函数
```

2.2.4.1 实测分析

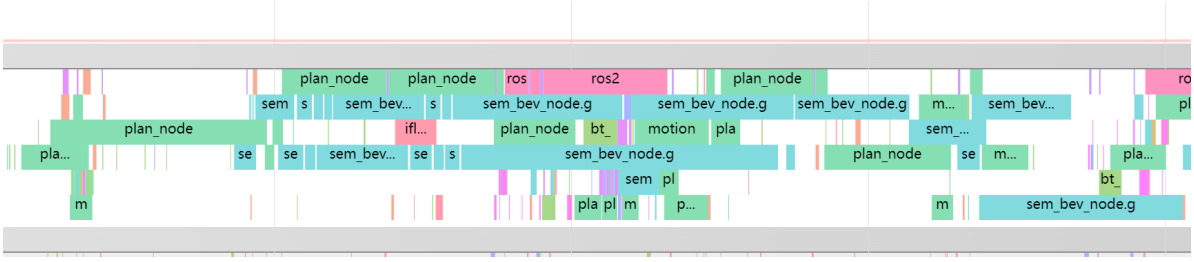
串口从中断到调用完`flush_to_ldisc()`存在延时，因此确认串口数据读取的延时来自于调度的延迟。

2.2.5 抓异常时刻Trace数据

根据 Trace抓取章节抓取Trace数据，进行分析。

2.2.5.1 实测分析

UART0 的中断 和 kwork0都是绑定在CPU0 上，出问题的时候， sem_bev_node.g-1531 持续跑了9.5ms，所以导致kwork0 没有机会调度到。所以需要等`sem_bev_node.g-1531`主动放弃CPU0，通过Trace数据查看，看到`kwork_queue` 7ms没有调度到， 要等上一个任务主动放弃CPU。



2.2.6 排查系统实时调度配置

当前系统内核配置：

```
#这个配置选项启用了 Linux 内核的自愿抢占特性。自愿抢占允许进程在执行期间主动放弃 CPU，以便其他进程可以运行。这种抢占方式是基于任务之间的协作，而不是强制性的时间片轮转。通过启用这个配置，可以提高系统的响应性能，尤其在多任务环境下
CONFIG_PREEMPT_VOLUNTARY
#这个配置选项定义了内核时钟滴答的频率。HZ 是一个内核常量，表示每秒中断的次数，用于调度和计时等操作。将 HZ 设置为 300 意味着每秒会有 300 次时钟中断。较高的 HZ 值可以提高系统的精确性和响应性能
CONFIG_HZ=300
```

调整优化后的内核配置：

```
#这个配置选项启用了 Linux 内核的抢占特性。抢占允许内核在执行期间中断任务，并立即切换到更高优先级的任务。通过启用这个配置，内核可以更及时地响应紧急事件和高优先级任务，提高系统的实时性能
CONFIG_PREEMPT

#这个配置选项定义了内核时钟滴答的频率。HZ 是一个内核常量，表示每秒中断的次数，用于调度和计时等操作。将 HZ 设置为 1000 意味着每秒会有 1000 次时钟中断。较高的 HZ 值可以提高系统的精确性和响应性能
CONFIG_HZ=1000:
```

修改目的是为了打开内核抢占特性，一个任务不会长时间占用CPU，减小延迟；同时修改减小内核心跳间隔，提高任务切换响应。

2.2.7 排查串口中断绑定情况

2.2.7.1 实测分析

UART0 的中断 和 kwork0都是绑定在CPU0 上，实际CPU0有很大的负载，对于串口中断响应的实时性也会有影响。

通过`cat /proc/interrupts`可以查看中断响应的情况，从下面信息可以看到ttyS0有比较多的中断，且分配到CPU0

```
root@RK3582:/# cat /proc/interrupts
           CPU0       CPU1       CPU2       CPU3       CPU4       CPU5
 13:     170960     189521     227141     291827     209204     178688     GICv3
 26 Level     arch_timer
 14:      13261      18274      13784       6986      25954      22381     GICv3
321 Level     rk_timer
 15:          0          0          0          0          0          0     GICv3
 23 Level     arm-pmu
 16:         10          0          0          0          0          0     GICv3
105 Level     dmc
 17:          0          0          0          0          0          0     GICv3
247 Level     ehci_hcd:usb1
 18:          0          0          0          0          0          0     GICv3
248 Level     ohci_hcd:usb2
 19:          0          0          0          0          0          0     GICv3
425 Level     rockchip_usb2phy
```

```
 20:          0          0          0          0          0          0     GICv3
423 Level     rockchip_usb2phy
 21:       2405          0          0          0          0          0     GICv3
349 Level     fd880000.i2c
 22:      36061          0          0          0          0          0     GICv3
363 Level     ttyS0
 23:          0          0          0          0          0          0     GICv3
142 Level     fdab9000.iommu, fdab0000.npu
 24:          0          0          0          0          0          0     GICv3
143 Level     fdab9000.iommu, fdab0000.npu
 25:          0          0          0          0          0          0     GICv3
144 Level     fdab9000.iommu, fdab0000.npu
 26:          0          0          0          0          0          0     GICv3
151 Level     fdb50400.vdpu
 27:          0          0          0          0          0          0     GICv3
150 Level     fdb50800.iommu
 28:       1718          0          0          0          0          0     GICv3
146 Level     fdb60f00.iommu, rga3_core0
 29:          0          0          0          0          0          0     GICv3
147 Level     fdb70f00.iommu, rga3_core1
 30:          0          0          0          0          0          0     GICv3
148 Level     rga2
 31:          0          0          0          0          0          0     GICv3
161 Level     fdb90000.jpegd
 32:          0          0          0          0          0          0     GICv3
162 Level     fdb90480.iommu
 33:          0          0          0          0          0          0     GICv3
154 Level     fdba0000.jpege-core
 34:          0          0          0          0          0          0     GICv3
153 Level     fdba0800.iommu
 35:          0          0          0          0          0          0     GICv3
156 Level     fdba4000.jpege-core
 36:          0          0          0          0          0          0     GICv3
155 Level     fdba4800.iommu
 37:          0          0          0          0          0          0     GICv3
158 Level     fdba8000.jpege-core
 38:          0          0          0          0          0          0     GICv3
157 Level     fdba8800.iommu
 39:          0          0          0          0          0          0     GICv3
160 Level     fdbac000.jpege-core
 40:          0          0          0          0          0          0     GICv3
159 Level     fdbac800.iommu
 41:          0          0          0          0          0          0     GICv3
149 Level     fdbb0800.iommu, fdbb0000.iep
 42:          0          0          0          0          0          0     GICv3
133 Level     fdbd0000.rkvenc-core
 43:          0          0          0          0          0          0     GICv3
131 Level     fdbdf000.iommu
 44:          0          0          0          0          0          0     GICv3
132 Level     fdbdf000.iommu
 45:          0          0          0          0          0          0     GICv3
141 Level     fdca0000.iommu
 46:          0          0          0          0          0          0     GICv3
163 Level     rkisp_hw
 47:          0          0          0          0          0          0     GICv3
165 Level     rkisp_hw
 48:          0          0          0          0          0          0     GICv3
166 Level     rkisp_hw
```

```
 49:          0          0          0          0          0          0     GICv3
164 Level     fdcb7f00.iommu
 50:       6920          0          0          0          0          0     GICv3
167 Level     rkisp_hw
 51:       3460          0          0          0          0          0     GICv3
169 Level     rkisp_hw
 52:          0          0          0          0          0          0     GICv3
170 Level     rkisp_hw
 53:          0          0          0          0          0          0     GICv3
168 Level     fdcc7f00.iommu
 54:      20575          0          0          0          0          0     GICv3
187 Level     rkcifhw
 55:          0          0          0          0          0          0     GICv3
145 Level     fdce0800.iommu
 56:          0          0          0          0          0          0     GICv3
175 Level     rockchip-mipi-csi2
 57:          0          0          0          0          0          0     GICv3
176 Level     rockchip-mipi-csi2
 58:          0          0          0          0          0          0     GICv3
177 Level     rockchip-mipi-csi2
 59:          0          0          0          0          0          0     GICv3
178 Level     rockchip-mipi-csi2
 60:          5          0          0          0          0          0     GICv3
179 Level     rockchip-mipi-csi2
 61:          0          0          0          0          0          0     GICv3
180 Level     rockchip-mipi-csi2
 62:          2          0          0          0          0          0     GICv3
181 Level     rockchip-mipi-csi2
 63:          0          0          0          0          0          0     GICv3
182 Level     rockchip-mipi-csi2
 64:      34865          0          0          0          0          0     GICv3
235 Level     dw-mci
 65:      44775          0          0          0          0          0     GICv3
237 Level     mmc0
 67:          0          0          0          0          0          0     GICv3
212 Level     i2s
 68:          0          0          0          0          0          0     GICv3
213 Level     i2s
 69:      13808          0          0          0          0          0     GICv3
118 Level     fea10000.dma-controller
 70:          0          0          0          0          0          0     GICv3
119 Level     fea10000.dma-controller
 71:          0          0          0          0          0          0     GICv3
120 Level     fea30000.dma-controller
 72:          0          0          0          0          0          0     GICv3
121 Level     fea30000.dma-controller
 73:        115          0          0          0          0          0     GICv3
351 Level     feaa0000.i2c
 74:      15171          0          0          0          0          0     GICv3
353 Level     feac0000.i2c
 75:        235          0          0          0          0          0     GICv3
354 Level     fead0000.i2c
 76:       2523          0          0          0          0          0     GICv3
360 Level     feb20000.spi
 78:     147561          0          0          0          0          0     GICv3
371 Level     ttyS8
 79:          5          0          0          0          0          0     GICv3
372 Level     ttyS9
```

```
 80:          0          0          0          0          0          0     GICv3
429 Level     rockchip_thermal
 81:       3896          0          0          0          0          0     GICv3
430 Level     fec10000.saradc
 82:          1          0          0          0          0          0     GICv3
355 Level     fec80000.i2c
 83:        209          0          0          0          0          0     GICv3
357 Level     feca0000.i2c
 84:        294          0          0          0          0          0     GICv3
122 Level     fed10000.dma-controller
 85:          0          0          0          0          0          0     GICv3
123 Level     fed10000.dma-controller
 91:          0          0          0          0          0          0     GICv3
455 Edge      debug-signal
 92:          0          0          0          0          0          0     GICv3
365 Level     debug
 93:          0          0          0          0          0          0     GICv3
140 Level     av1d-master
 96:          0          0          0          0          0          0
rockchip_gpio_irq   7 Level     rk806
 97:          0          0          0          0          0          0     rk806
  0 Edge      rk805_pwrkey_fall
 98:          0          0          0          0          0          0     rk806
  1 Edge      rk805_pwrkey_rise
 99:          0          0          0          0          0          0     rk806
  7 Level     rk806_vb_low
100:     220559          0          0          0          0          0     GICv3
252 Level     dwc3
101:      44527          0          0          0          0          0
rockchip_gpio_irq   4 Edge      csk_resume_core
102:          0          0          0          0          0          0
rockchip_gpio_irq  28 Edge      wifi config GPIO
103:          0          0          0          0          0          0
rockchip_gpio_irq   0 Edge      rtw_wifi_gpio_wakeup
IPI0:    250042     354926     384199     423915     276318     285196
Rescheduling interrupts
IPI1:     62962      68359      67971      73474     171214     183095
Function call interrupts
IPI2:         0          0          0          0          0          0       CPU
stop interrupts
IPI3:         0          0          0          0          0          0       CPU
stop (for crash dump) interrupts
IPI4:       878       2299       1474        833       4717       4447
Timer broadcast interrupts
IPI5:       370        104        108        203        146         83       IRQ
work interrupts
IPI6:         0          0          0          0          0          0       CPU
wake-up interrupts
```

2.2.7.2 中断绑定空闲核心

可以看到CPU4响应的中断数量较小，且CPU4为大核，性能更强，可以把对实时性要求较高的UART0绑定至CPU4，加快对UART0中断响应的时间。

以下为UART0中断绑定CPU4的修改补丁：

```
diff --git a/drivers/tty/serial/8250/8250_core.c
b/drivers/tty/serial/8250/8250_core.c
index 00f6dc7e9477..b4b9c985f977 100644
--- a/drivers/tty/serial/8250/8250_core.c
+++ b/drivers/tty/serial/8250/8250_core.c
@@ -176,6 +176,7 @@ static int serial_link_irq_chain(struct uart_8250_port *up)
    struct hlist_node *n;
    struct irq_info *i;
    int ret;
+   struct cpumask cpumask;

    mutex_lock(&hash_mutex);

@@ -214,6 +215,12 @@ static int serial_link_irq_chain(struct uart_8250_port *up)
                  up->port.irqflags, up->port.name, i);
        if (ret < 0)
            serial_do_unlink(i, up);
+
+       if (up->port.line == 0)
+           cpumask_clear(&cpumask);
+           cpumask_set_cpu(4, &cpumask);
+           irq_set_affinity(up->port.irq, &cpumask);
+       }
    }
```

修改后可以通过`cat /proc/interrupts`查看中断绑定核心的情况。

2.2.8 排查系统是否有实时线程

2.2.8.1 实时线程说明

在 Linux 中，实时线程是指具有实时调度属性的线程。实时调度是一种能够确保任务在特定时间约束下得到及时执行的调度方式。与普通的时间片轮转调度不同，实时调度会根据任务的优先级和时间限制进行调度，以确保实时任务能够在规定的时间内完成。

Linux 提供了两种类型的实时调度策略：FIFO（First-In-First-Out）和 RR（Round Robin）。FIFO 调度策略根据优先级进行调度，具有最高优先级的任务先被执行，直到任务结束或被其他更高优先级的任务抢占。RR 调度策略则按照轮转的方式进行调度，每个任务被分配一个时间片，在时间片用完之前会被挂起，然后切换到下一个任务。

实时线程通常用于处理对时间敏感的任务，如实时控制、数据采集、音视频处理等。它们需要满足严格的时间约束，以确保任务的响应性和实时性。为了创建实时线程，需要使用特定的函数和调用适当的参数，以设置线程的实时调度属性和优先级。

2.2.8.2 排查

若系统里有实时线程的话，可能会影响普通线程的调度，所以需要排查系统是否存在实时线程，我们可以根据 实时线程扫描脚本章节抓取系统上是否有实时线程进行分析。

2.2.8.3 实测分析

通过脚本的扫描，或是trace文件的抓取，可以发现系统有实时线程，且优先级配置为`prio=0`，就存在实时线程的高优先级会抢占普通线程甚至内核线程，这部分要求客户进行调整修改。

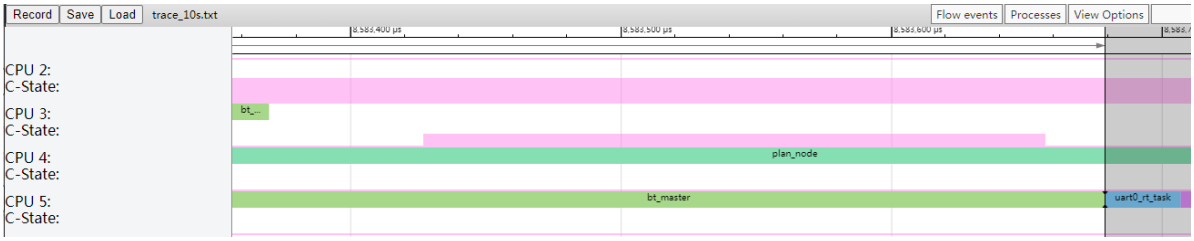
通过如下信息可以看到，有很多应用线程设为了实时线程，prio=0，且绑定到了CPU4、CPU5。

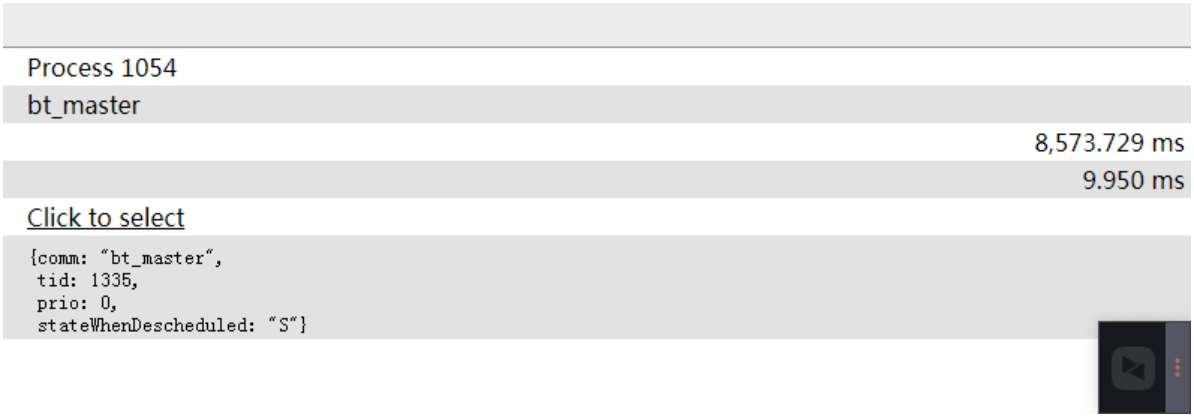
```
map_fusion-1207    (   1152) [004] dNh3    97.095981: sched_wakeup:
comm=bt_master pid=1335 prio=0 target_cpu=004
plan_node-1279    (   1036) [004] dNh3    97.106023: sched_wakeup: comm=bt_master
pid=1335 prio=0 target_cpu=004
motion-1074    (   1035) [000] d.s5    97.115596: sched_wakeup: comm=bt_master
pid=1338 prio=0 target_cpu=005
relocation_node-1148    (   1148) [004] dNh5    97.116059: sched_wakeup:
comm=bt_master pid=1335 prio=0 target_cpu=004
<...>-1031    (   1031) [004] dNh3    97.137900: sched_wakeup: comm=bt_master
pid=1335 prio=0 target_cpu=004
relocation_node-1148    (   1148) [000] d.s5    97.147671: sched_wakeup:
comm=bt_master pid=1338 prio=0 target_cpu=005
call_robot-1052    (   1052) [004] dNh3    97.147932: sched_wakeup:
comm=bt_master pid=1335 prio=0 target_cpu=004
```

根据trace图形化信息可以看到：

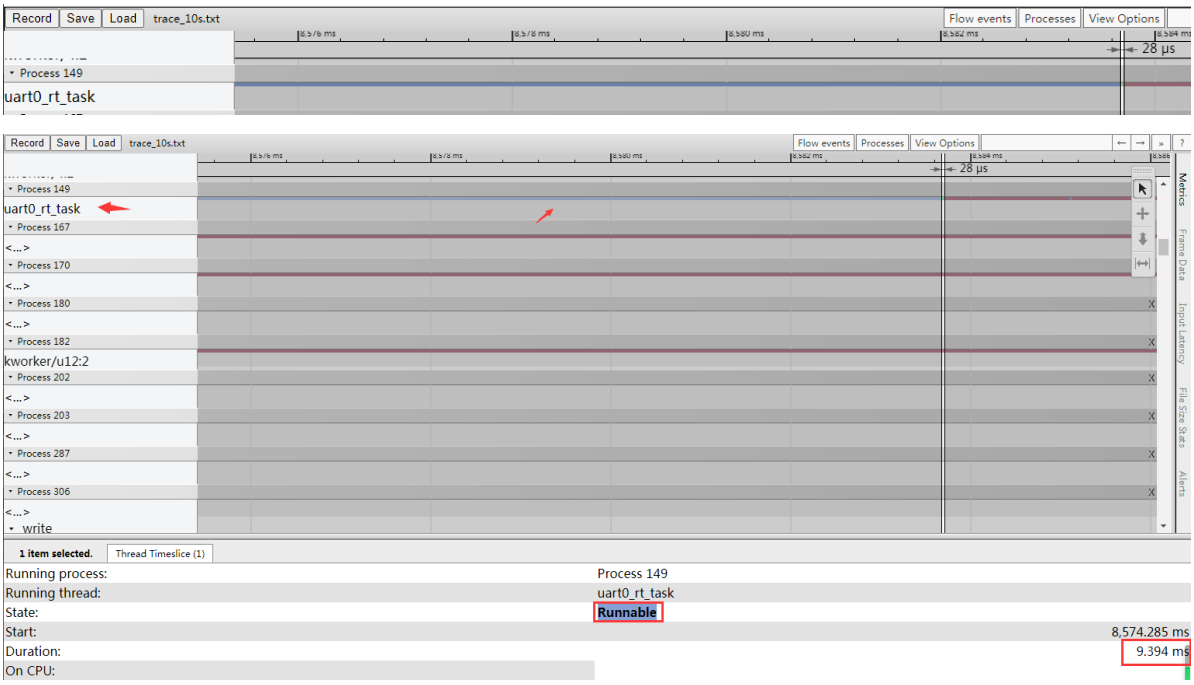
1.每次UART数据上报的流程：ttyS0中断->唤醒内核uart0-rt-task线程->唤醒应用读取ttyS0线程，可以看到第二次ttyS0中断来了之后，唤醒uart0-rt-task间隔了11.25ms以上

2.放大第二次uart0-rt-task调度部分，可以看到uart0-rt-task前被bt_master阻塞，且bt_master的prio=0，优先级最高





3.查看此uart0_rt_task的调度情况，其中蓝色部分为Runnable即等待被调度，绿色部分为Running及调度到执行。可以看到该线程等待调度等了9.394ms，运行时间只有28us。可以得出内核线程被其他更高优先级任务（应用bt_master线程优先级配为0）阻塞。



2.2.8.4 优化

2.2.8.4.1 应用实时线程优先级调整

应用线程优先级设为0，相当于是设为了实时线程，且优先级最高。看这些应用线程都耗时非常高。建议只能把优先级高的，操作时间短的任务拆成实时线程，且要保证其他任务不会依赖实时线程的资源释放。

2.2.8.4.2 内核实时线程

为了提升内核 kwork_queue 的实时性，调整 kwork_queue 的调度策略：根据特定应用场景和实时需求，可以考虑调整 kwork_queue 的调度策略。例如，使用实时线程调度策略，以更好地满足实时任务的响应要求。

以下为讲内核UART0（与MCU统信）数据上报改成RT线程的补丁参考：

```
diff --git a/drivers/tty/tty_buffer.c b/drivers/tty/tty_buffer.c
index 5bbc2e010b48..1c08b666e050 100644
--- a/drivers/tty/tty_buffer.c
+++ b/drivers/tty/tty_buffer.c
```

```
@@ -543,12 +543,36 @@ static inline void tty_flip_buffer_commit(struct tty_buffer
*tail)
  * held off and retried later.
  */

+#include <linux/kthread.h>
+#include <uapi/linux/sched/types.h>
+
+static int tty_rt_task(void *data)
+{
+   struct work_struct *work = data;
+
+   set_task_comm(current, "uart0_rt_task");
+
+   while (!kthread_should_stop()) {
+       set_current_state(TASK_RUNNING);
+       flush_to_ldisc(work);
+       set_current_state(TASK_UNINTERRUPTIBLE);
+       schedule();
+   }
+
+   return 0;
+}
+
 void tty_flip_buffer_push(struct tty_port *port)
 {
    struct tty_bufhead *buf = &port->buf;

    tty_flip_buffer_commit(buf->tail);
-   queue_work(system_unbound_wq, &buf->work);
+   if (port->tty && port->tty->driver->subtype == SERIAL_TYPE_NORMAL && port-
>tty->index == 8) {
+       if (port->uart_task) {
+           wake_up_process(port->uart_task);
+       }
+   } else {
+       queue_work(system_unbound_wq, &buf->work);
+   }
 }
 EXPORT_SYMBOL(tty_flip_buffer_push);

@@ -594,6 +618,7 @@ int tty_insert_flip_string_and_push_buffer(struct tty_port
*port,
 void tty_buffer_init(struct tty_port *port)
 {
    struct tty_bufhead *buf = &port->buf;
+   struct sched_param param = { .sched_priority = MAX_RT_PRIO - 51 };

    mutex_init(&buf->lock);
    tty_buffer_reset(&buf->sentinel, 0);
@@ -603,6 +628,8 @@ void tty_buffer_init(struct tty_port *port)
    atomic_set(&buf->mem_used, 0);
    atomic_set(&buf->priority, 0);
    INIT_WORK(&buf->work, flush_to_ldisc);
+   port->uart_task = kthread_create(tty_rt_task, &buf->work, "tty_task");
+   sched_setscheduler_nocheck(port->uart_task, SCHED_FIFO, &param);
    buf->mem_limit = TTYB_DEFAULT_MEM_LIMIT;
 }
```

```
diff --git a/include/linux/tty.h b/include/linux/tty.h
index bd76f2f1cde0..9cd68af8baba 100644
--- a/include/linux/tty.h
+++ b/include/linux/tty.h
@@ -255,7 +255,7 @@ struct tty_port {
                                set to size of fifo */
     struct kref     kref;       /* Ref counter */
     void            *client_data;
-
+    struct task_struct  *uart_task;
     ANDROID_KABI_RESERVE(1);
 };
```

修改完可以查看线程的优先级来确认是否修改有效：

```
# 找一下这个任务的pid
ps -A | grep uart0_rt_task
# 通过$pid查看优先级，我们配置的是50
cat /proc/$pid/sched | grep prio
```

2.2.8.4.3 应用实时线程

同时我们也可以把应用的需要实时性操作的线程，改为实时线程，如应用层读取串口节点的线程，我们将其改为实时线程。

要将Linux应用线程设为实时线程（RT线程），可以参考以下代码，我们配置的优先级比内核的优先级小一点即可：

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sched.h>
#include <unistd.h>

void* threadFunc(void* arg) {
    // 实时线程的具体逻辑代码

    return NULL;
}

int main() {
    pthread_t thread;
    pthread_attr_t attr;
    struct sched_param param;

    // 初始化线程属性
    pthread_attr_init(&attr);

    // 设置线程调度策略为实时策略
    pthread_attr_setinheritsched(&attr, PTHREAD_EXPLICIT_SCHED);
    pthread_attr_setschedpolicy(&attr, SCHED_FIFO);

    // 设置线程优先级
    param.sched_priority = sched_get_priority_max(SCHED_FIFO) - 60;  // 优先级范围一般为1-99，99为最高优先级
    pthread_attr_setschedparam(&attr, &param);
```

```
    // 创建实时线程
    if (pthread_create(&thread, &attr, threadFunc, NULL) != 0) {
        perror("Failed to create thread");
        return 1;
    }

    // 等待线程结束
    if (pthread_join(thread, NULL) != 0) {
        perror("Failed to join thread");
        return 1;
    }

    return 0;
}
```

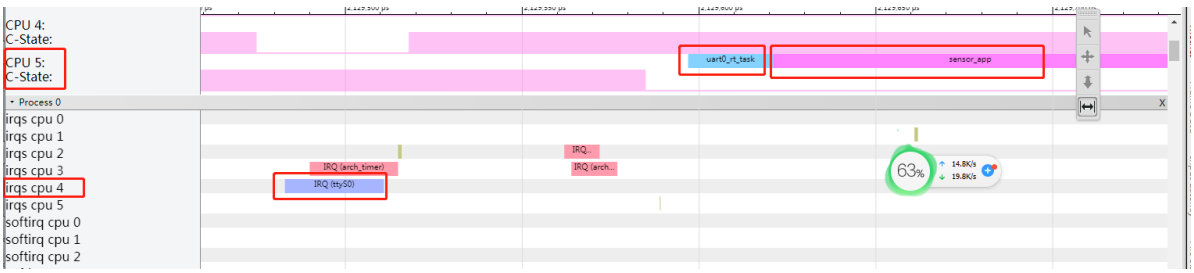
2.2.8.4.4 实时线程注意事项

在实现 Linux 实时线程时，有一些注意事项需要考虑，以确保实时任务的正确性和可靠性。以下是一些常见的注意事项：

1. 优先级设置：正确设置实时线程的优先级是关键。较高优先级的任务会优先执行，但过高的优先级可能导致系统不稳定或其他任务无法得到充分执行。因此，需要根据实际需求和系统负载来合理设置优先级。
2. 避免优先级反转：在使用实时线程时，需要小心处理优先级反转问题。当一个低优先级任务持有某个资源时，如果一个高优先级任务等待该资源，会导致高优先级任务无法及时执行，从而产生优先级反转。为避免这种情况，可以使用互斥锁、信号量等机制进行资源访问的同步和保护。
3. 避免资源争用：实时线程之间可能存在资源争用的问题，特别是共享资源的访问。如果多个实时线程同时访问共享资源，可能导致竞争条件和数据不一致。在设计实时线程的逻辑时，需要考虑合适的同步机制，如互斥锁、条件变量等，以确保对共享资源的安全访问。
4. 实时任务的时间约束：实时任务通常有严格的时间约束，需要在规定的时间内完成。因此，在实现实时线程时，需要对任务的执行时间有清晰的认识，并合理安排任务的调度和处理逻辑，以确保任务能够及时响应和完成。不能在实时线程里处理太多耗时操作。尽量要小于1ms。
5. 避免长时间阻塞：实时线程应尽量避免长时间的阻塞操作，例如等待外部事件或资源。长时间阻塞可能导致任务无法按时执行，影响实时性能。可以使用非阻塞的方式进行等待，并合理处理超时情况。
6. 周期性任务的设计：如果涉及到周期性任务，例如定时触发的实时任务，需要根据实际需求和时间约束来设计任务的周期和触发机制。可以使用定时器、周期性定时器或硬件定时器等来实现周期性任务的触发。

2.2.9 实时线程绑定核心

由于UART TX会在中断中处理发送数据，将数据填入硬件FIFO，中断的优先级会比实时线程来的高，会打断RX实时线程，因为此前已将串口ttyS0中断绑定到CPU4上，现在需要把实时线程绑定到CPU5，让中断不打断实时线程。



2.2.9.1 内核实时线程绑定核心

在这个示例中，`bind_kernel_thread_to_cpu5()` 函数将当前内核线程绑定到 CPU 5。我们创建一个新的 CPU 核心掩码 `new_mask`，并使用 `cpumask_set_cpu()` 将 CPU 5 添加到 `new_mask` 中。然后，我们将新的 CPU 核心掩码应用到当前内核线程，使其绑定到 CPU 5。

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

void bind_kernel_thread_to_cpu5(void)
{
    struct task_struct *task = current;  // 获取当前内核线程的 task_struct

    cpumask_t new_mask;
    cpumask_clear(&new_mask);
    cpumask_set_cpu(5, &new_mask);  // 设置 CPU 5

    set_cpus_allowed_ptr(task, &new_mask);  // 将新的 CPU 核心掩码应用到内核线程
}
```

2.2.9.2 应用线程绑定核心

要将应用程序线程绑定到 CPU5，可以使用 `pthread_setaffinity_np()` 函数来实现。这个函数允许您设置线程允许运行的 CPU 核心集合。

以下是使用 `pthread_setaffinity_np()` 函数将应用程序线程绑定到 CPU 5 的示例代码：

```
#include <pthread.h>
#include <sched.h>
#include <stdio.h>

void bind_application_thread_to_cpu5(void)
{
    pthread_t thread = pthread_self();  // 获取当前线程的 ID

    cpu_set_t cpuset;
    CPU_ZERO(&cpuset);          // 清空 CPU 核心集合
    CPU_SET(5, &cpuset);        // 设置 CPU 5

    int result = pthread_setaffinity_np(thread, sizeof(cpu_set_t), &cpuset);  //
将线程绑定到 CPU 5
    if (result != 0) {
        fprintf(stderr, "Failed to bind thread to CPU 5. Error code: %d\n",
result);
        // 错误处理
        // 可以根据错误码使用 pthread_strerror(result) 来获取错误信息
    }
}
```

2.2.10 实时内核补丁

Linux 发行版提供了实时内核补丁（如 PREEMPT-RT 补丁），可以显著提高内核的实时性能。可以考虑使用这些补丁来增强 kwork_queue 的实时特性。

RK有提供专门带RT-Linux功能支持的内核版本，内部仓库：

```
https://10.10.10.29/admin/repos/rk/kernel-rt
```

2.2.11 系统工作频率

CPU和DDR频率可以对系统的实时性能产生影响，影响的具体程度取决于系统的具体配置和工作负载。以下是它们可能对实时性的影响的一些方面：

1. 响应时间：较高的CPU频率可以加快任务的处理速度，从而缩短任务的响应时间。这对于实时系统非常重要，因为它们需要在规定的时间内对事件做出响应。同样，较高的DDR频率可以提供更快的内存访问速度，有助于减少延迟并提高任务执行效率。
2. 中断处理：实时系统通常需要及时处理中断事件。较高的CPU频率可以加快中断处理程序的执行速度，确保及时处理重要的中断事件。较高的DDR频率可以加快内存中断数据的读取和写入速度，提高中断处理的效率。
3. 多任务处理：实时系统可能需要同时处理多个任务。较高的CPU频率可以提供更大的计算能力，使系统能够更快地切换和执行多个任务。较高的DDR频率可以加快任务之间的数据传输，有助于提高多任务处理的效率。

2.2.11.1 查看系统CPU和DDR频率

通过查看CPU和DDR频率，来确认是否有必要抬高频率后再进行实时性测试。

```
cat /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
cat /sys/devices/system/cpu/cpu*/cpufreq/scaling_cur_freq
cat /sys/class/devfreq/dmc/governor
cat /sys/class/devfreq/dmc/cur_freq
```

3. 实时性能问题总结

经过分析，最终RK3588读取串口节点数据慢的解决方案如下：

1. CONFIG_PREEMPT 打开（抢占打开，一个任务不会长时间占用CPU，减小延迟）
2. CONFIG_HZ=1000（减小内核心跳间隔，提高任务切换响应）
3. UART0中断绑定到大核CPU4上（提高UART0的中断响应速度）
4. 单独把内核UART0（与MCU通信）数据上报改成RT线程（保证UART0线程唤醒的时候，可以马上得到CPU，不用等待其他普通任务）
5. 应用读取串口UART0节点线程改为RT线程（保证读取UART0节点的应用线程唤醒时，可以马上得到CPU，不用等待其他普通任务）
6. 在RT线程中去掉耗时和阻塞的操作（避免影响实时线程的调度）
7. 内核UART0数据上报实时线程和应用读取串口UART0节点实时线程，都绑定CPU5，与串口中断绑定的CPU4隔离开（否则TX数据发送时，会打断RX实时线程）

8. 需要合理调整应用实时线程的优先级，已排查到很多应用线程设为RT线程，需要结合产品应用进行修改
9. MCU有概率地会超过13ms发送数据，此作为已知问题，MCU部分去排查

4. 实时性能分析工具

4.1 Trace抓取

Linux Trace是一种用于抓取和分析系统运行时信息的工具。允许开发人员跟踪和分析系统的各种活动，以便深入了解系统的性能、行为和故障。下面是关于Linux Trace数据抓取的说明：

1. 数据抓取范围：Linux Trace可以抓取各种级别的数据，包括系统级别、进程级别和内核级别的数据。可以选择抓取特定进程、线程或系统整体的数据，以满足您的分析需求。
2. 数据抓取方法：Linux Trace提供了多种抓取方法，包括静态跟踪、动态跟踪和事件触发跟踪。可以根据需要选择合适的方法进行数据抓取。
3. 抓取的数据类型：Linux Trace可以抓取各种类型的数据，如系统调用、函数调用、硬件事件、中断、进程状态、网络流量等。这些数据可以帮助您了解系统的运行状态、性能瓶颈和资源利用情况。
4. 数据分析和可视化：抓取的数据可以通过专门的工具进行分析和可视化。可以使用Chrome浏览器进行可视化的分析。这些工具可以帮助识别系统瓶颈、优化性能，并提供可视化的结果。
5. 性能分析和故障排查：通过抓取和分析Linux Trace数据，可以进行性能分析和故障排查。可以识别系统中的性能瓶颈，发现潜在的问题，并定位故障的根本原因。这有助于改进系统的性能和可靠性。
6. 实时监控和调试：Linux Trace还可以用作实时监控和调试工具。您可以在系统运行时实时监视和分析各种活动，以便及时响应和调试问题。

4.1.1 Trace配置

kernel打开如下配置

```
CONFIG_IRQSOFF_TRACER=y
CONFIG_PREEMPT_TRACER=y
CONFIG_SCHED_TRACER=y
```

4.1.2 Trace抓取脚本

如下为Trace抓取脚本，命名为captrue.sh，示例中抓取10s的数据：

```
#!/bin/sh

# 禁用调度程序中的 NO_TTWU_QUEUE 功能
echo NO_TTWU_QUEUE > /sys/kernel/debug/sched_features

# 进入事件跟踪的目录
cd /sys/kernel/debug/tracing/events

# 启用特定的事件类别进行跟踪
```

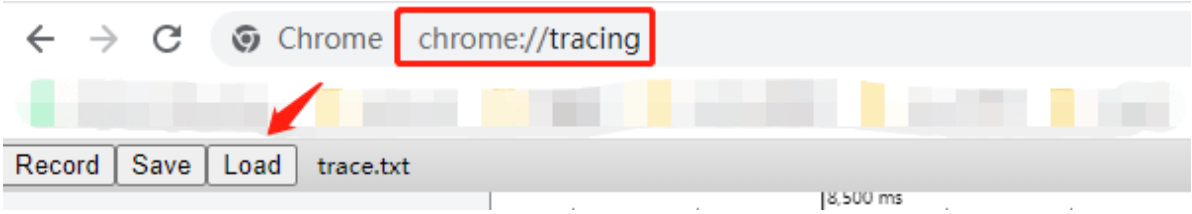
```
echo 0 > sched/enable                          # 调度事件
echo 1 > sched/sched_switch/enable             # 调度切换事件
echo 1 > sched/sched_wakeup/enable             # 调度唤醒事件
echo 1 > block/enable                          # 块设备事件
echo 1 > scsi/enable                           # SCSI 事件
echo 1 > irq/enable                            # 中断事件
echo 1 > workqueue/enable                      # 工作队列事件
echo 1 > power/enable                          # 电源事件

# 返回跟踪目录
cd ..

# 设置跟踪的缓冲区大小（以千字节为单位）
echo 96000 > buffer_size_kb

# 启用进程组 ID 的记录
echo 1 > options/record-tgid

# 将跟踪时钟设置为 'global'
echo global > trace_clock

# 开始跟踪
echo 1 > tracing_on

# 等待 10 秒钟以捕获数据
sleep 10

# 停止跟踪
echo 0 > tracing_on

# 将捕获的跟踪保存到指定的文件中
cat trace > $1
```

命令执行示例:

```
# 将trace抓取存放至/data/trace.txt
./captrue.sh /data/trace.txt
```

4.1.3 Trace文件查看方法

打开 Chrome 浏览器并输入网址 "chrome://tracing/"，然后按 Enter 键。这将打开 Chrome 浏览器的跟踪页面。



点击Load导入输出的`trace.txt`。

4.1.4 Trace可视化操作说明

在 Chrome 浏览器的跟踪页面，W、A、S、D 键通常用于在时间轴上进行导航和缩放。以下是相应的快捷键说明：

- **W**：时间轴放大。
- **A**：向左滚动时间轴。
- **S**：时间轴缩小。
- **D**：向右滚动时间轴。
- **<**：向左移动任务（移动过程被选中的任务会高亮显示）。
- **>**：向右移动任务（移动过程被选中的任务会高亮显示）。

4.2 实时线程扫描脚本

4.2.1 扫描脚本

可通过`ps -A`命令获取当前系统主要的进程ID，通过以下脚本`proc_thread_name.sh`扫描遍历指定进程的任务列表，并输出每个任务的名称、文件名、调度策略、优先级和 CPU 亲和性。

```
#!/bin/sh

# 遍历指定进程的任务列表
for file in $(ls /proc/$1/task/)
do
    # 获取任务的名称
    pid_name=$(cat /proc/$1/task/$file/status | grep Name)
    # 获取任务的 CPU 亲和性
    cpu_set=$(cat /proc/$1/task/$file/status | grep Cpus_allowed_list)
    # 获取任务的调度策略
    policy=$(cat /proc/$1/task/$file/sched | grep policy)
    # 获取任务的优先级
    prio=$(cat /proc/$1/task/$file/sched | grep prio)
    # 输出任务的信息
    echo "$pid_name $file $policy $prio $cpu_set"
done
```

4.2.2 执行命令

```
./proc_thread_name.sh $PID
```

4.2.3 执行结果示例

policy为1，即为实时线程；policy为0，即为普通线程。

prio为线程任务的优先级。

```
root@RK3588:/# ./proc_thread_name.sh 1308
Name:   sem_bev_node.g 1308 policy                                      :
             0 prio                                        :
 101 Cpus_allowed_list:  0-5
Name:   sem_bev_node.g 1380 policy                                      :
             0 prio                                        :
 120 Cpus_allowed_list:  0-5
Name:   sem_bev_node.g 1381 policy                                      :
             0 prio                                        :
 120 Cpus_allowed_list:  0-5
Name:   sem_bev_node.g 1382 policy                                      :
             0 prio                                        :
 120 Cpus_allowed_list:  0-5
Name:   gc 1383 policy                                      :
 0 prio                                    :                 120
Cpus_allowed_list:      0-5
Name:   dq.builtins 1384 policy                                         :
         0 prio                                    :                 120
Cpus_allowed_list:     0-5
Name:   dq.user 1385 policy                                        :
     0 prio                                    :                 120
Cpus_allowed_list: 0-5
Name:   tev 1386 policy                                      :
 0 prio                                    :                 120
Cpus_allowed_list:     0-5
Name:   recv 1387 policy                                       :
   0 prio                                    :                 120
Cpus_allowed_list:    0-5
Name:   recvUC 1388 policy                                        :
     0 prio                                    :                 120
Cpus_allowed_list:  0-5
Name:   sem_bev_node.g 1389 policy                                      :
             0 prio                                        :
 120 Cpus_allowed_list:  0-5
Name:   sem_bev_node.g 1390 policy                                      :
             0 prio                                        :
 120 Cpus_allowed_list:  4
Name:   sem_bev_node.g 1391 policy                                      :
             0 prio                                        :
 120 Cpus_allowed_list:  0-5
Name:   sem_bev_node.g 1392 policy                                      :
             0 prio                                        :
 120 Cpus_allowed_list:  4
```

```
Name:   sem_bev_node.g 1393 policy                                    :
            0 prio                                          :
 120 Cpus_allowed_list:  0-5
Name:   sem_bev_node.g 1394 policy                                    :
            0 prio                                          :
 120 Cpus_allowed_list:  0-5
Name:   sem_bev_node.g 1395 policy                                    :
            0 prio                                          :
 120 Cpus_allowed_list:  5
Name:   sem_bev_node.g 1396 policy                                    :
            0 prio                                          :
 120 Cpus_allowed_list:  5
Name:   sem_bev_node.g 1397 policy                                    :
            0 prio                                          :
 120 Cpus_allowed_list:  5
Name:   sem_bev_node.g 1398 policy                                    :
            0 prio                                          :
 120 Cpus_allowed_list:  0-5
Name:   sem_bev_node.g 1399 policy                                    :
            0 prio                                          :
 120 Cpus_allowed_list:  0-5
Name:   sem_bev_node.g 1400 policy                                    :
            0 prio                                          :
 120 Cpus_allowed_list:  0-5
Name:   sem_bev_node.g 1401 policy                                    :
            0 prio                                          :
 120 Cpus_allowed_list:  0-5
Name:   sem_bev_node.g 1402 policy                                    :
            0 prio                                          :
 120 Cpus_allowed_list:  0-5
Name:   sem_bev_node.g 1403 policy                                    :
            0 prio                                          :
 120 Cpus_allowed_list:  0-5
Name:   sem_bev_node.g 1404 policy                                    :
            0 prio                                          :
 120 Cpus_allowed_list:  0-5
```

4.3 cycletest

cycletest"是一个用于测试 Linux 实时性能的工具。它主要用于评估系统的实时性能和响应时间，并帮助检测系统中的潜在问题。

使用 cycletest"工具，可以执行以下操作：

1. 测试周期：可以指定测试的周期，即测试中运行的任务的执行间隔时间。这可以帮助评估系统在不同负载下的实时性能。
2. 任务类型：工具提供了不同类型的任务，例如 CPU 密集型任务和 I/O 密集型任务。可以选择适合您需求的任务类型进行测试。
3. 任务数量：可以设置测试中运行的任务数量。通过同时运行多个任务，可以模拟系统中的多线程/多进程环境，并评估系统在高并发负载下的实时性能。
4. 实时性能度量：工具会测量每个任务的执行时间、响应时间和中断延迟等指标。这些度量值可以帮助您评估系统的实时性能，并找出潜在的性能瓶颈或问题。
5. 结果输出：测试完成后，工具会生成测试报告。报告中包含了每个任务的执行时间、平均响应时间、最大中断延迟等指标，以及潜在的问题和建议的改进措施。

4.3.1 测试命令

```
cyclictest -c 0 -m -n -t 3 -a -p 99
```