Use on **RT-THREAD** **SYSTEMVIEW** analysis tool

RT-THREAD Documentation Center

WWW.RT-THREAD.ORG

Friday 28th September, 2018

Table of contents

This application note introduces the SystemView visualization analysis tool and how to use it in
It is used on RT-Thread to debug and analyze the system.

1 Purpose and structure of this paper

1.1 Purpose and Background of this Paper

As MCU performance becomes stronger and stronger, the functions of embedded products become more and more complex, which poses new challenges to system debugging and analysis. Debugging a function or problem usually requires a lot of effort. SystemView is a powerful tool to help users debug and analyze the system, which can significantly shorten the development and debugging time and improve development efficiency. The purpose of this article is to help you use the SystemView tool on RT-Thread to debug and analyze the system.

1.2 Structure of this paper

This article first introduces what the SystemView tool is, then introduces the SystemView software package on RT-Thread, as well as the specific enabling and configuration methods. After that, a specific DEMO is used to introduce the detailed usage process of the SystemView analysis tool. Finally, a parameter directory is placed for your reference when configuring.

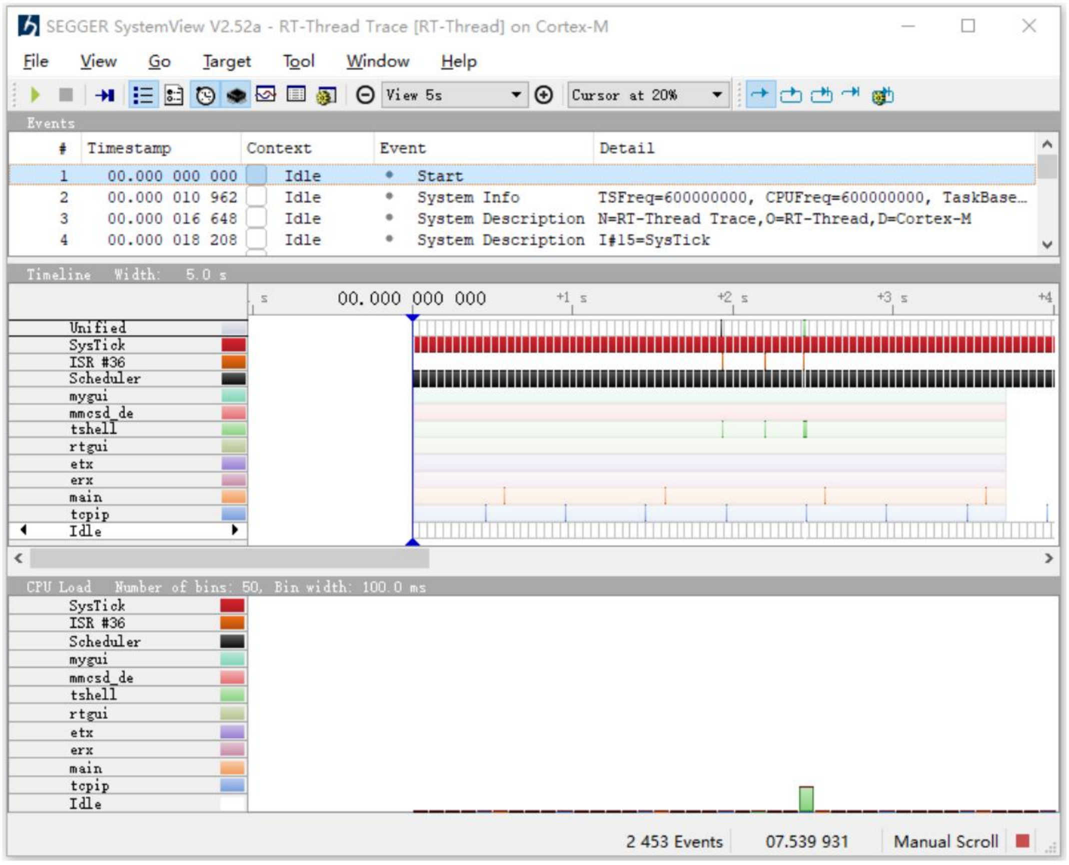
2 What is SystemView

SystemView is a tool for online debugging of embedded systems. It can analyze which interrupts and tasks have been executed, as well as the order in which these interrupts and tasks are executed. It can also view the time points when some kernel objects are held and released, such as semaphores, mutexes, events, message queues, etc. This is especially effective when developing and processing complex systems with multiple threads and events.

SystemView consists of two parts:

- PC-side program, collects and displays data from the embedded end, and can also save this data locally for later use analyze.
- Embedded-side program that collects the operating data of the embedded system and transmits them through the RTT module of J-Link For PC

Because SystemView's data transmission utilizes J-Link's RTT technology, SystemView can only be used when the development board is connected with J-Link.



#	Timestamp	Context	Event	Detail
1	00.000 000 000	Idle	Start	
2	00.000 010 962	Idle	System Info	TSFreq=600000000, CPUFreq=600000000, TaskBase…
3	00.000 016 648	Idle	System Description	N=RT-Thread Trace,O=RT-Thread,D=Cortex-M
4	00.000 018 208	Idle	System Description	I#15=SysTick

2 453 Events 07.539 931 Manual Scroll

Figure 1: *SystemView* The main interface

3 SystemView package on RT-Thread

3.1 Introduction

SystemView package on RT-Thread It is an embedded end program implementation of the SystemView tool. Its main functions include: configuring specific parameters of SYSTEMVIEW and RTT, collecting and formatting monitoring data, and sending data to the PC through J-Link. You only need to use the env tool launched by RT-Thread By enabling the SystemView software package and performing a simple configuration on it, you can complete the configuration of the SystemView embedded end program.

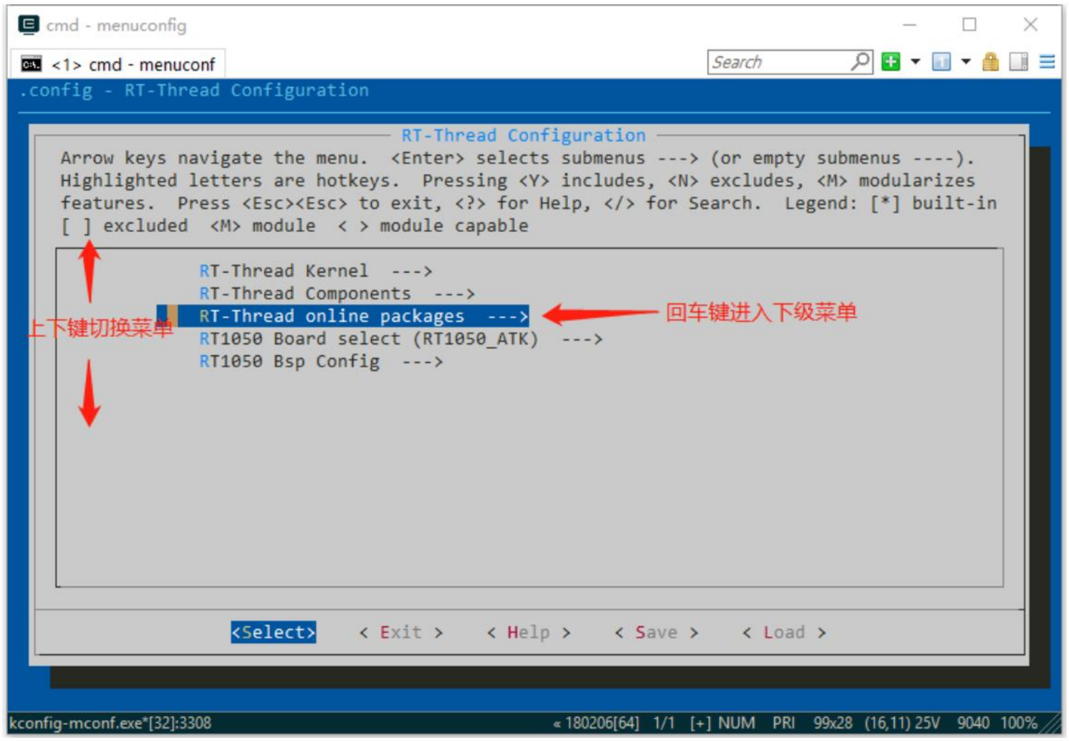
3.2 How to enable, configuration options

Take the Zhengdian Atom RT1052 development board as an example

Step 1: Enter the menuconfig graphical configuration tool in the env tool

Open the env tool and use the command cd D:\rt-thread\bsp\imxrt1052-evk to switch to the RT-Thread source bsp root directory, and then enter the menuconfig command to configure the project.

menuconfig is a graphical configuration tool that RT-Thread uses to configure and tailor the entire system.



```
.config - RT-Thread Configuration
                          RT-Thread Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
  features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in
  [ ] excluded  <M> module  < > module capable

              RT-Thread Kernel  --->
              RT-Thread Components  --->
              RT-Thread online packages  --->
              RT1050 Board select (RT1050_ATK)  --->
              RT1050 Bsp Config  --->

        <Select>    < Exit >    < Help >    < Save >    < Load >
```

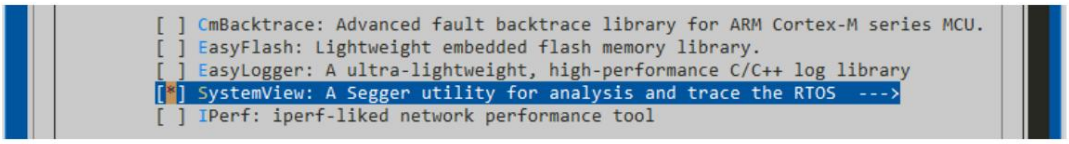
cmd - menuconfig
<1> cmd - menuconf
Search
上下键切换菜单
回车键进入下级菜单
kconfig-mconf.exe*[32]:3308
« 180206[64] 1/1 [+] NUM PRI 99x28 (16,11) 25V 9040 100%

Figure 2: *menuconfig* Tool interface

Step 2: Enable the SystemView software package.

Use the up and down keys to select RT-Thread online packages, press Enter to enter the sub-menu, and then in tools

Open SystemView in packages.



```
[ ] CmBacktrace: Advanced fault backtrace library for ARM Cortex-M series MCU.
[ ] EasyFlash: Lightweight embedded flash memory library.
[ ] EasyLogger: A ultra-lightweight, high-performance C/C++ log library
[*] SystemView: A Segger utility for analysis and trace the RTOS  --->
[ ] IPerf: iperf-liked network performance tool
```

image 3: Enable *SystemView*

Step 3: Specific configuration.

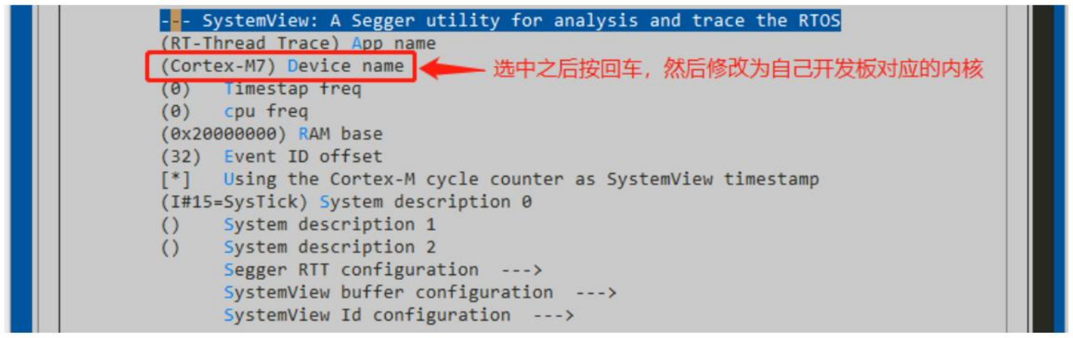
Press the Enter key to enter the sub-menu and make specific configurations (enter? to display specific information of the options).

The following are two common options. Other options can be configured as needed (refer to the last section).

Number of directories):

1. Change the option circled in red in the figure below to the kernel corresponding to your development board, such as I.MX-RT1052 corresponding to M7 kernel.

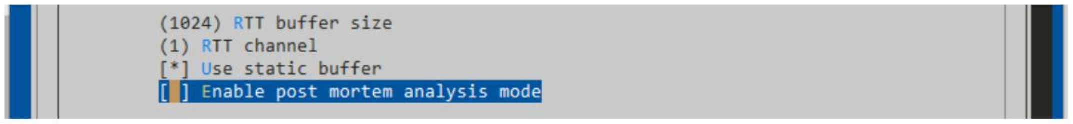
nuclear



```
--- SystemView: A Segger utility for analysis and trace the RTOS
(RT-Thread Trace) App name
(Cortex-M7) Device name
(0)    Timestamp freq
(0)    cpu freq
(0x20000000) RAM base
(32)   Event ID offset
[*]    Using the Cortex-M cycle counter as SystemView timestamp
(I#15=SysTick) System description 0
()     System description 1
()     System description 2
       Segger RTT configuration  --->
       SystemView buffer configuration  --->
       SystemView Id configuration  --->
```

Figure 4: Specific configuration

2. Enter the SystemView buffer configuration submenu and turn off post-analysis mode



```
(1024) RTT buffer size
(1) RTT channel
[*] Use static buffer
[ ] Enable post mortem analysis mode
```

Figure 5: Turn off post-mortem mode

After enabling post-analysis mode, you need to call the function SEGGER_SYSVIEW_Start() in the project to start recording. System events will be continuously recorded and old events will be overwritten when the buffer is full. Therefore, when reading the buffer, only the latest events can be read.

Post-mortem analysis can be useful when a system has been running for a long time and suddenly crashes. In this case, the most recent events can be read from the buffer in the target and SystemView can show what happened just before the system crashed.

After turning off the post-analysis mode, it will be in continuous recording mode, and you can control the start and end of recording on the PC. Continuously record the operation of the system. To analyze the operation of the following demo, you need to turn on the continuous recording mode.

After configuring the options, press ESC to return, exit and save the configuration. The enabling and related configuration of the SystemView software package is now complete.

4 Use of SystemView Tool

We will use a specific demo to explain the use of the SystemView tool

4.1 Adding demo code

Add the following code to the file main.c, and then call the demo initialization function demo_init (); in the main function to run the demo.

Machine Translated by Google

```
/*
* Program listing: systemview demo code
*
* In this example, a dynamic semaphore (initial value is 0) and two dynamic threads will be created.
        In thread
* Thread 2 will try to hold the semaphore by waiting forever, and send the running flag after holding it successfully. * Thread 1 will first send the running
flag, and then release the semaphore once. Because thread 2 has a higher priority, thread 1 will send the running flag first, and then release the semaphore once.
        Process 2 holds the semaphore and intercepts thread 1.
* Then after thread 2 sends the running flag, it cannot obtain the semaphore and is suspended, and thread 1 continues to run
*/

#define THREAD_PRIORITY                 25
#define THREAD_STACK_SIZE               512
#define THREAD_TIMESLICE /*             5
Pointer to semaphore */
rt_sem_t sem_food; /*
Thread 1 entry */
void thread1_entry(void* parameter) {

        while (1) {

                /* Thread 1 runs for the first
                time*/ rt_kprintf("thread1 is run!\n"); /* Release
                a semaphore*/
                rt_sem_release(sem_food); /*
                Thread 1 runs for the second
                time*/ rt_kprintf("thread1 run again!\n"); /* Thread
                1 delays for 1 second*/
                rt_thread_delay(RT_TICK_PER_SECOND);
        }

} /* Thread 2 entry */
void thread2_entry(void* parameter) {

        while (1) {

                /* Try to hold the semaphore and wait forever until the semaphore is held*/
                rt_sem_take(sem_food, RT_WAITING_FOREVER); /*
                Thread 2 is running*/
                rt_kprintf("thread2 is run!\n");
        }
}
/* DEMO initialization function
*/ void demo_init(void) {
```

Machine Translated by Google

```
    /* Pointer to thread control block */
    rt_thread_t thread1_id, thread2_id; /* Create a
    semaphore, the initial value is 0 */
    sem_food = rt_sem_create("sem_food", 0, RT_IPC_FLAG_FIFO); if (sem_food ==
    RT_NULL) {

        rt_kprintf("sem created fail!\n"); return ;


    } /* Create thread 1 */
    thread1_id = rt_thread_create("thread1", thread1_entry,
                            RT_NULL,/* Thread entry is thread1_entry, parameter RT_NULL */

                            THREAD_STACK_SIZE, THREAD_PRIORITY, THREAD_TIMESLICE)
                                ;
    if (thread1_id != RT_NULL)
        rt_thread_startup(thread1_id); /* Create
    thread 2 */ thread2_id
    = rt_thread_create("thread2", thread2_entry, RT_NULL,/*
                            Thread entry is thread2_entry, parameter RT_NULL */

                            THREAD_STACK_SIZE, THREAD_PRIORITY - 1,
                                THREAD_TIMESLICE);
    if (thread2_id != RT_NULL)
        rt_thread_startup(thread2_id);
}
```

4.2 Configuration and use of SystemView PC program

Step 1: Download SystemView analysis tool download link

Step 2: Add a system description file for RT-Thread

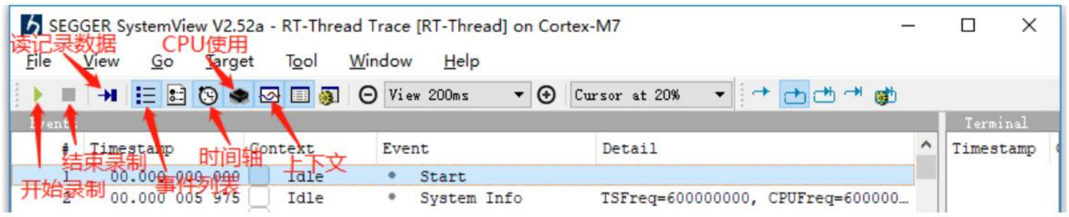
First, find the packages directory under the development board directory, and then find the segger_debug-xxx directory under the packages directory. There is a SystemView_Description folder in this directory, and the description file of the RT-Thread system is in it. The specific directory structure is as follows:

bsp\\your own development board\\packages\\segger_debug-xxx\\SystemView_Description\\
SYSVIEW_RT-Thread.txt

Copy this file to the Description directory under the SystemView tool installation directory, so that SystemView can identify the RT-Thread system.

Step 3: Configure device information and start recording

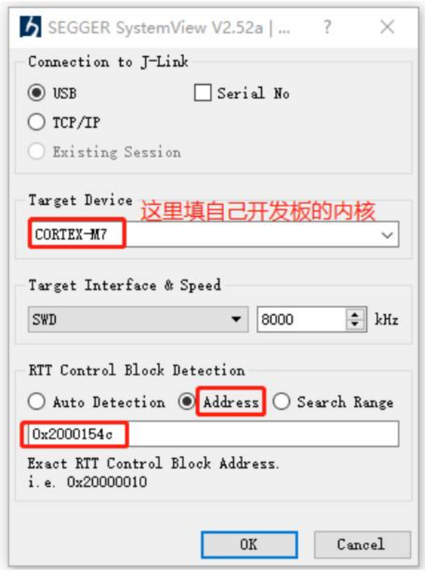
Double-click to open the SystemView PC program. Below is an introduction to the functions of some commonly used buttons on the main interface.


SEGGER SystemView V2.52a - RT-Thread Trace [RT-Thread] on Cortex-M7
读记录数据
CPU使用
File View Go Target Tool Window Help
View 200ms
Cursor at 20%
结束录制
时间轴
上下文
开始录制
事件列表
Terminal
Timestamp

```
#  Timestamp      Context  Event          Detail
1  00.000 000 000 Idle     Start
2  00.000 005 975 Idle     System Info    TSFreq=600000000, CPUFreq=600000…
```

Figure 6: *PC* Terminal program button introduction

When in continuous recording mode, click the Start Recording button and a window will pop up to configure the device information.



Connection to J-Link
[✓ USB] Serial No
TCP/IP
Existing Session

Target Device
CORTEX-M7

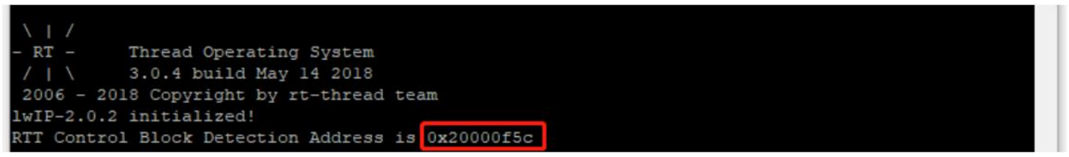
Target Interface & Speed
SWD 8000 kHz

RTT Control Block Detection
Auto Detection [✓ Address] Search Range
0x2000154c
Exact RTT Control Block Address.
i.e. 0x20000010

Figure 7: Window for configuring device information

The address of the RTT control block in the figure has been printed out through the serial port. Just open the terminal software and print the printed address.

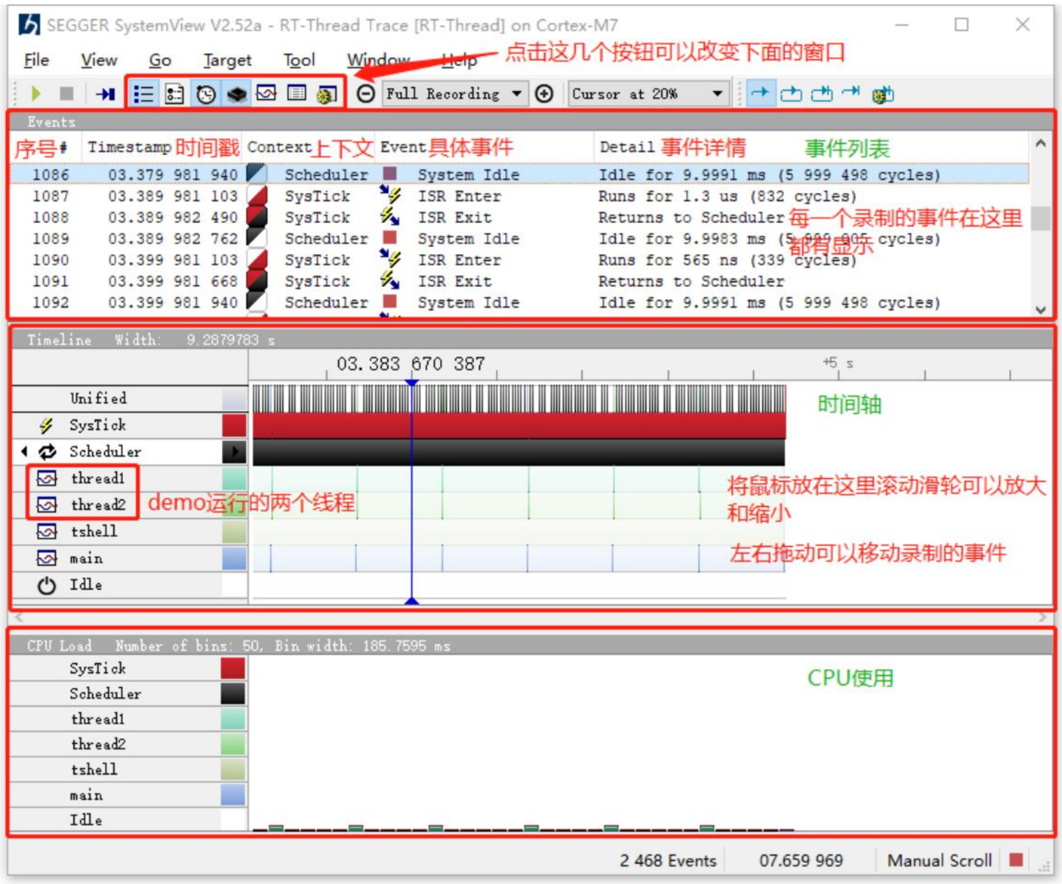
Copy the address above.



```
 \ | /
- RT -     Thread Operating System
 / | \     3.0.4 build May 14 2018
 2006 - 2018 Copyright by rt-thread team
lwIP-2.0.2 initialized!
RTT Control Block Detection Address is 0x20000f5c
```

Figure 8: Get from terminal *RTT* The address of the control block

Click OK. Now SystemView starts recording system information in real time. Below is the real-time operation status of the system.


SEGGER SystemView V2.52a - RT-Thread Trace [RT-Thread] on Cortex-M7
File View Go Target Tool Window Help
点击这几个按钮可以改变下面的窗口
Full Recording
Cursor at 20%
事件列表
每一个录制的事件在这里都有显示
Timeline Width: 9.2879783 s
03.383 670 387
+5 s
时间轴
Unified
SysTick
Scheduler
thread1
thread2
tshell
main
Idle
demo运行的两个线程
将鼠标放在这里滚动滑轮可以放大和缩小
左右拖动可以移动录制的事件
CPU Load Number of bins: 50, Bin width: 185.7595 ms
CPU使用
2 468 Events
07.659 969
Manual Scroll

Events

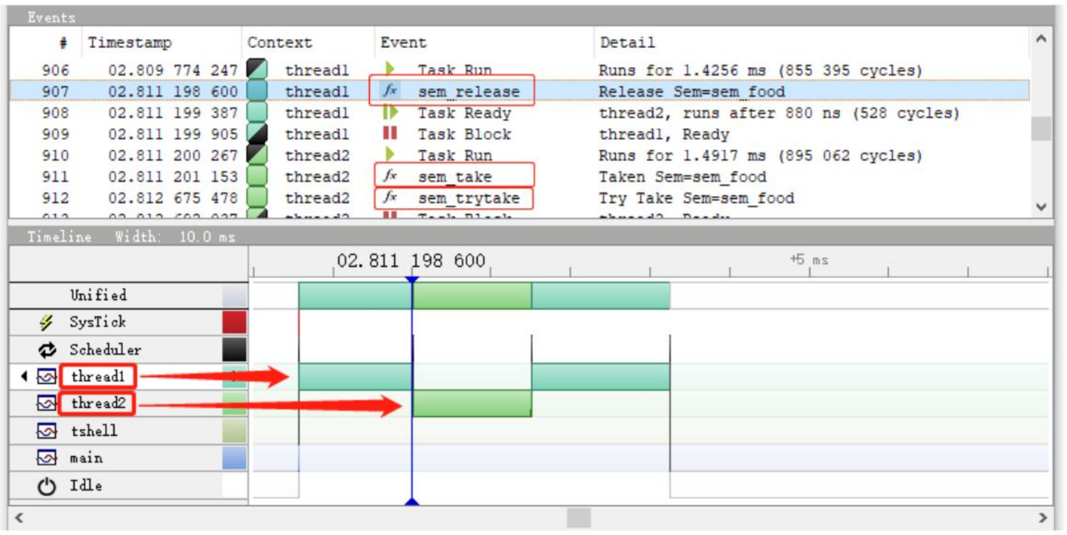
序号#	Timestamp 时间戳	Context上下文	Event具体事件	Detail 事件详情
1086	03.379 981 940	Scheduler	System Idle	Idle for 9.9991 ms (5 999 498 cycles)
1087	03.389 981 103	SysTick	ISR Enter	Runs for 1.3 us (832 cycles)
1088	03.389 982 490	SysTick	ISR Exit	Returns to Scheduler
1089	03.389 982 762	Scheduler	System Idle	Idle for 9.9983 ms (5 999 005 cycles)
1090	03.399 981 103	SysTick	ISR Enter	Runs for 565 ns (339 cycles)
1091	03.399 981 668	SysTick	ISR Exit	Returns to Scheduler
1092	03.399 981 940	Scheduler	System Idle	Idle for 9.9991 ms (5 999 498 cycles)

Figure 9: Real-time system operation status

Step 4: End recording and analyze the system

Click the End Recording button to end the recording. Place the mouse in the timeline window and use the scroll wheel to zoom in on the event to a suitable size.

Size of combined analysis



Events

#	Timestamp	Context	Event	Detail
906	02.809 774 247	thread1	Task Run	Runs for 1.4256 ms (855 395 cycles)
907	02.811 198 600	thread1	sem_release	Release Sem=sem_food
908	02.811 199 387	thread1	Task Ready	thread2, runs after 880 ns (528 cycles)
909	02.811 199 905	thread1	Task Block	thread1, Ready
910	02.811 200 267	thread2	Task Run	Runs for 1.4917 ms (895 062 cycles)
911	02.811 201 153	thread2	sem_take	Taken Sem=sem_food
912	02.812 675 478	thread2	sem_trytake	Try Take Sem=sem_food

Figure 10: Analysis Thread 1, 2 run

Using the SystemView tool, we can see that the system is running as we expected. Thread 1 is running first.
It starts running, and then is interrupted by thread 2 after releasing the semaphore. Then thread 2 runs, but then fails to obtain the semaphore.
From the above event list, we can also see the specific time when each thread acquires or releases the semaphore.
time.

If the post-analysis mode is turned on, you cannot click the Start Recording button, but click Read Recording Data
The other operations are the same as those in the real-time analysis mode.

After reading this document, I believe you have learned how to use the SystemView analysis tool on RT-Thread.

5 **menuconfig** parameter details

parameter	describe
App name	Application Name
Device name	The kernel used by the device
Timestap frequency	Timestamp frequency (0 means using the system default frequency)
cpu freq	CPU frequency (0 means using the system default frequency)
RAM base	RAM base address default value: 0x2000 0000
Event ID offset	Event ID offset default value: 32
Using the Cortex-M cycle …	Use system frequency as timestamp
System description 0-2	System descriptor “I#num=name, …” num is the interrupt number, name is Interrupt Name

parameter	describe
Max number of up buffer	Maximum number of RTT output buffers Default value: 2
Max num of dowm buffer	Maximum number of RTT input buffers Default value: 2
buffer size up	Number of bytes to use for RTT endpoint output channels Default value: 1024 bytes
buffer size down	Number of bytes used for the RTT terminal input channel Default value: 16 bytes Festival
Segger RTT printf buffer size	The default buffer size when sending character blocks via RTT printf is Default value: 64
Mode for pre-initialized terminal channel	Pre-initial RTT terminal channel mode default value: NO_BLOCK_SKIP

parameter	describe
Max Interrupt priority	Maximum interrupt priority level
Use RTT ASM	Using the assembly version of RTT
memcpy uses byte-loop	Use a simple byte-loop instead of memcpy

parameter	describe
RTT buffer size	The number of bytes used by SystemView to log the buffer
RTT channel	The RTT channel is used for SystemView event logging and communication. Automatic selection
Use static buffer	Using static buffers can save space
Enable post mortem analysis mode	Enable post-mortem mode

parameter	describe
ID Base	The value subtracted from the ID recorded in the SystemView package Default value: 0x1000 0000
ID Shift	The number of digits of the ID shift documented in the SystemView package. Default value: 2