S.Ha.R.K. User Manual

Volume V

*** DRAFT ***

The S.Ha.R.K. New Tracer

Written by

Tullio Facchinetti (tullio.facchinetti@unipv.it)

Scuola Superiore di Studi e Perfezionamento S. Anna

RETIS Lab

Via Carducci, 40 - 56100 Pisa

Contents

Chapter 1

The S.Ha.R.K. tracer

The S.Ha.R.K. new Tracer is a powerful tool to understand what happens while a S.Ha.R.K. application is executed. The Tracer logs a set of events corresponding with the most important activities within the kernel, such as preemptions, interrupt activations, mutexes blocks and releases, and much more.

During the execution, the kernel logs the sequence of events in memory; such events are then written to disk or sent through the network, typically at the end of the application execution. Each event is made by: a high resolution timestamp (TSC, Time Stamp Counter) corresponding to the instant at which the event has been logged; the event type; 2 optional parameters to add additional information to the event.

The Tracer can also be used to log custom events, since it reserves a number of free event types for user events.

To use the features made available by the Tracer, the Tracer functions must be enabled into the kernel by specifying

TRACER = NEW

into the *shark.cfg* configuration file. The S.Ha.R.K. kernel must be built with this option set.

Chapter 2

Primitives

2.1 FTrace_enable

```
int FTrace_enable();
```

Enable the Tracer. When this function is called, the Tracer starts the event logging.

2.2 FTrace_disable

```
int FTrace_disable();
```

Disable the Tracer. When this function is called, the Tracer stops the event logging.

2.3 FTrace_chunk_create

```
int FTrace_chunk_create(int normal_size, int emergency_size, FTrace_flags flags);
```

Create a new chunk.

2.4 FTrace_chunk_delete

```
int FTrace_chunk_delete(int number);
```

Delete a Chunk.

2.5 FTrace_set_chunk_flags

```
int FTrace_set_chunk_flags(int number, FTrace_flags flags);
```

Set the chunk flags.

2.6 FTrace_get_chunk_flags

```
int FTrace_get_chunk_flags(int number, FTrace_flags *flags);
```

Returns chunk flags.

2.7 FTrace_actual_chunk_select

```
int FTrace_actual_chunk_select(int number);
```

Select the actual chunk.

2.8 FTrace_chunk_link

```
int FTrace_chunk_link(int chunk_A, int chunk_B);
```

Link two chunks.

2.9 FTrace_get_first_chunk

```
int FTrace_get_first_chunk(FTrace_flags flags);
```

Find the first chunk with specific flags.

2.10 FTrace_get_chunk_table

```
FTrace_Chunk_Ptr *FTrace_get_chunk_table();
```

Get one chunks status.

2.11 FTrace_compress_chunk

```
int FTrace_compress_chunk(int number, FTrace_flags new_flags);
```

Create a new memory region where the compressed data are stored.

2.12 FTrace_send_chunk

```
int FTrace_send_chunk(int number, int osd_flags, FTrace_flags new_flags);
```

Send the chunk out from the memory.

2.13 FTrace_init_disk_writer

```
int FTrace_init_disk_writer(char *fname, int flag, char *l_ip, char *t_ip);
```

Initialize the disk Tracer chunk dumper. It sets the internal chunk sender to the function that writes chunks on disk. It initializes the filename that will be used to open the file for saving chunks.

2.14 FTrace_disk_writer

```
void FTrace_disk_writer(FTrace_Chunk_Ptr c);
```

This function is called by the application when it asks to write chunk c on disk. It saves the chunk data into the chunk_to_disk array. At the runlevel after the exit, all the saved chunks will be written to disk.

2.15 FTrace_OSD_init_udp

```
int FTrace_OSD_init_udp(int flag, char *l_ip, char *t_ip);
```

Initialize the Tracer chunk network sender using the UDP protocol supported by S.Ha.R.K. If flag = 1 initializes the network driver, otherwise it considers that the network layer has already been initialized. It also sets the internal chunk sender to the function that initializes the task for sending the chunk.

2.16 FTrace_set_filter

```
void FTrace_set_filter(BYTE family, int status);
```

Set the filter for a specific family of events (see Table 3.13 for a list of all the event families) . When the filter is enabled for a given family of events, all the events belonging to that family are not logged.

While `status` set to 1 enables the filter, `status` set to 0 disables the filter.

2.17 TRACER_LOGEVENT

```
TRACER_LOGEVENT(WORD type, WORD par1, DWORD par2);
```

Stores a new event of `type` type into the current chunk, together with the 2 parameters `par1` (2 bytes) and `par2` (4 bytes).

Chapter 3

Event types description

This Chapter reports all the available event type codes currently supported by S.Ha.R.K.

Name	Code	Parameter 1	Parameter 2
FTrace_EVT_empty	0x00	.	.
FTrace_EVT_cycles_per_msec	0x10	.	[clk/msec]
FTrace_EVT_trace_start	0x20	.	.
FTrace_EVT_trace_stop	0x30	.	.
FTrace_EVT_blackout_start	0x40	.	.
FTrace_EVT_blackout_end	0x50	.	.
FTrace_EVT_id	0x60	context	pid
FTrace_EVT_numevents	0x70	.	.

Table 3.1: General trace events.

Name	Code	Parameter 1	Parameter 2
FTrace_EVT_ipoint	0x01	number	.

Table 3.2: Lightweight tracing events.

Name	Code	Parameter 1	Parameter 2
FTrace_EVT_task_create	0x02	context	pid
FTrace_EVT_task_activate	0x12	context	.
FTrace_EVT_task_dispatch	0x22	.	.
FTrace_EVT_task_epilogue	0x32	.	.
FTrace_EVT_task_end	0x42	context	pid
FTrace_EVT_task_begin_cycle	0x52	.	.
FTrace_EVT_task_end_cycle	0x62	context	level
FTrace_EVT_task_sleep	0x72	.	.
FTrace_EVT_task_schedule	0x82	exec_shadow.context	exec.context
FTrace_EVT_task_timer	0x92	context	level
FTrace_EVT_task_disable	0xA2	.	.
FTrace_EVT_task_deadline_miss	0xB2	context	.
FTrace_EVT_task_wcet_violation	0xC2	context	.

Table 3.3: Task related events.

Name	Code	Parameter 1	Parameter 2
FTrace_EVT_interrupt_start	0x03	int	.
FTrace_EVT_interrupt_end	0x13	int	.
FTrace_EVT_interrupt_hit	0x23	Instant where interrupt was hit (no end)	
FTrace_EVT_interrupt_count	0x33	Number of interrupts raised since last interrupt_count	

Table 3.4: Interrupt events, even more lightweight than ipoints.

Name	Code	Parameter 1	Parameter 2
FTrace_EVT_to_real_mode	0x04	.	.
FTrace_EVT_to_protected_mode	0x14	.	.
FTrace_EVT_CLI	0x24	.	.
FTrace_EVT_STI	0x34	.	.

Table 3.5: Other CPU specific events.

Name	Code	Parameter 1	Parameter 2
FTrace_EVT_set_priority	0x05	.	.
FTrace_EVT_context_switch	0x15	context	.
FTrace_EVT_inheritance	0x25	exec_shadow.context	exec.context

Table 3.6: Changes on task attributes and state.

Name	Code	Parameter 1	Parameter 2
FTrace_EVT_set_mutex_create	0x06	.	.
FTrace_EVT_set_mutex_lock	0x16	context	mutex
FTrace_EVT_set_mutex_inherit	0x26	.	.
FTrace_EVT_set_mutex_unlock	0x43	context	mutex
FTrace_EVT_set_mutex_wait	0x46	context	mutex
FTrace_EVT_set_mutex_post	0x56	context	mutex

Table 3.7: Mutex events.

Name	Code	Parameter 1	Parameter 2
FTrace_EVT_signal	0x07	.	.

Table 3.8: Signal events.

Name	Code	Parameter 1	Parameter 2
FTrace_EVT_server_create	0x08	.	server
FTrace_EVT_server_replenish	0x18	.	server
FTrace_EVT_server_exhaust	0x28	.	server
FTrace_EVT_server_reclaiming	0x38	.	server
FTrace_EVT_server_remove	0x48	.	server
FTrace_EVT_server_active	0x58	.	server
FTrace_EVT_server_using_rec	0x68	reclaiming	server

Table 3.9: Specific server events.

Name	Code	Parameter 1	Parameter 2
FTrace_EVT_user_event_0	0x09	free	free
FTrace_EVT_user_event_1	0x19	free	free
FTrace_EVT_user_event_2	0x29	free	free
FTrace_EVT_user_event_3	0x39	free	free
FTrace_EVT_user_event_4	0x49	free	free
FTrace_EVT_user_event_5	0x59	free	free
FTrace_EVT_user_event_6	0x69	free	free
FTrace_EVT_user_event_7	0x79	free	free
FTrace_EVT_user_event_8	0x89	free	free
FTrace_EVT_user_event_9	0x99	free	free
FTrace_EVT_user_event_10	0xA9	free	free
FTrace_EVT_user_event_11	0xB9	free	free
FTrace_EVT_user_event_12	0xC9	free	free
FTrace_EVT_user_event_13	0xD9	free	free
FTrace_EVT_user_event_14	0xE9	free	free

Table 3.10: User defined events.

Name	Code	Parameter 1	Parameter 2
FTrace_EVT_timer_post	0x0B	.	.
FTrace_EVT_timer_delete	0x1B	.	.
FTrace_EVT_timer_wakeup_start	0x2B	.	.
FTrace_EVT_timer_wakeup_end	0x3B	context	.

Table 3.11: Timer events.

Name	Code	Parameter 1	Parameter 2
FTrace_EVT_data_pointer	0x1A	holds a pointer of data from previous event	
FTrace_EVT_next_chunk	0xFF		

Table 3.12: Generic data events.

Name	Code
FTrace_filter_trace_Events	0xF0
FTrace_filter_ipoint	0xF1
FTrace_filter_task	0xF2
FTrace_filter_interrupt	0xF3
FTrace_filter_CPU	0xF4
FTrace_filter_priority	0xF5
FTrace_filter_mutex	0xF6
FTrace_filter_signal	0xF7
FTrace_filter_server	0xF8
FTrace_filter_user	0xF9
FTrace_filter_data	0xFA
FTrace_filter_timer	0xFB
FTrace_family_mask	0x0F

Table 3.13: Filter management.

Chapter 4

Example of Tracer usage

Algorithm 1 Example of Tracer usage.

```
/** Declarations */
int a,b,c;
SYS_FLAGS f;

/** Create 3 chunks for storing the tracer events.  */
a = FTrace_chunk_create(1000000, 1000000, FTRACE_CHUNK_FLAG_FREE | FTRACE_CHUNK_FLAG_CYC);
b = FTrace_chunk_create(1000000, 1000000, FTRACE_CHUNK_FLAG_FREE | FTRACE_CHUNK_FLAG_JTN);
c = FTrace_chunk_create(1000000, 1000000, FTRACE_CHUNK_FLAG_FREE | FTRACE_CHUNK_FLAG_CYC);
FTrace_chunk_link(a,b);
FTrace_chunk_link(b,c);

/** Select the first chunk for saving the events.  */
FTrace_actual_chunk_select(a);

/** Start the tracer.  */
FTrace_enable();

/** Enable filtering for timer related events.  */
FTrace_set_filter(FTrace_filter_timer, 1);
TRACER_LOGEVENT(FTrace_EVT_trace_start,proc_table[exec_shadow].context,clk_per_msec);
for (i=0;i<10;i++)
if (proc_table[i].context != 0) TRACER_LOGEVENT(FTrace_EVT_id, (unsigned short
int)proc_table[i].context,i);

// *** do something ***

/** Enable filtering for timer related events.  */
FTrace_set_filter(FTrace_filter_timer, 0);

/** Change the chunk where the events are stored.  */
TRACER_LOGEVENT(FTrace_EVT_next_chunk,0,0);

TRACER_LOGEVENT(FTrace_EVT_ipoint,6000,0);

// *** do something ***

/** Store a TFrace stop event.  */
TRACER_LOGEVENT(FTrace_EVT_trace_stop,0,0);

/** Stop the tracer.  */
FTrace_disable();

/** Initialize the network for remotely saving the trace.  */
FTrace_OSD_init_udp(1, "192.168.1.10", "192.168.1.1");

/**
* If want to save the events to disk, simply change
* the network initialization instruction with the following line:
*
* FTrace_init_disk_writer("trace.dat", 0, NULL, NULL);
*
*/

/** Save the chunk.  */
FTrace_send_chunk(a, 0, FTRACE_CHUNK_FLAG_FREE | FTRACE_CHUNK_FLAG_CYC);
FTrace_send_chunk(b, 0, FTRACE_CHUNK_FLAG_FREE | FTRACE_CHUNK_FLAG_JTN);
```

Chapter 5

Tracer output

When the trace is saved, locally or remotely, into a file, the resulting binary file appears as a sequence of bytes where each event stored within the trace output file is 16 bytes long. The format of each single event is depicted in Table 5.1.

The fields have the following meaning:

- the Code represent the event type (see Chapter 3 for the full list);
- Parameter 1 and 2 are the parameters used when `TRACER_LOGEVENT` is invoked;
- TSC is the Time Stamp Counter associated with the event;

If prt points to the first byte of an event, the correct TSC value can be obtained with the following instructions:

```
unsigned long long tsc_value;
tsc_value = (unsigned long long)(*(unsigned int *)(ptr + 4)) << 32;
tsc_value += (unsigned long long)(*(unsigned int *)(ptr + 8));
```

Code	Parameter 1	TSC (high part)	TSC (low part)	Parameter 2
2 bytes	2 bytes	4 bytes	4 bytes	4 bytes

Table 5.1: Event output file format.