System Profiling

Introduction

You will learn:

- how detailed data can be gathered from the kernel, about many things that are going on, all the way down to the interrupt and kernel call level
- how to add your own data
- how to control this data gathering from:
  - the IDE
  - the command line
  - your own code
- how to analyze this data

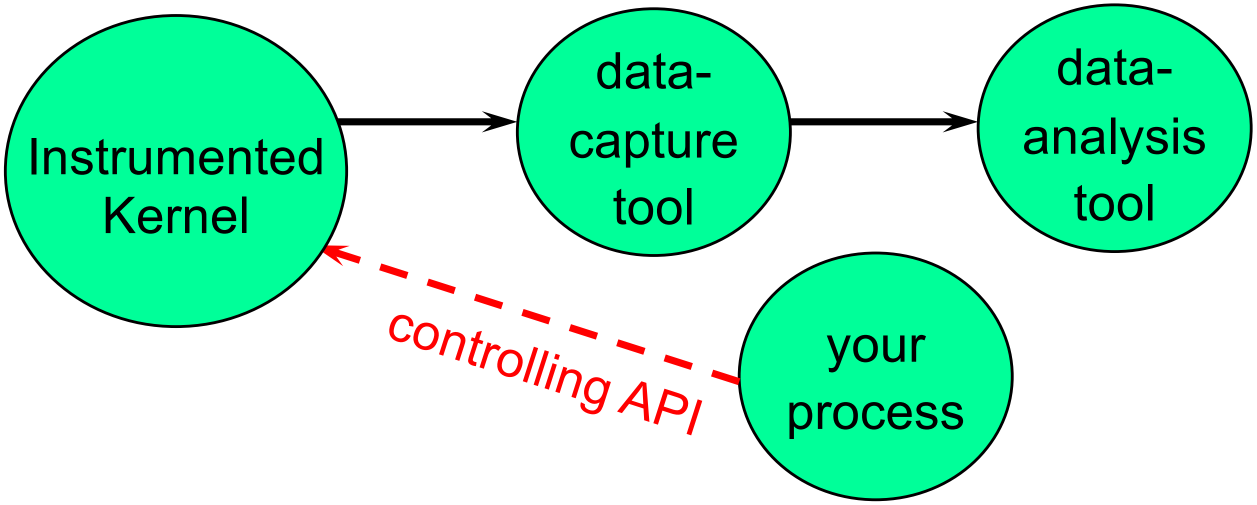
Topics:

- → **Overview**
- **Creating a Log**
- **Log Summary**
- **A Quick Tour**
- **Filtering Events and Event Owners**
- **Navigating through a log with the Timeline Pane**
- **Statistics**
- **CPU Activity Pane and CPU Usage Pane**
- **Tying The Trace to Your Code**
- **Multi-core Related Features**
- **Adaptive Partitioning: Partition Summary**
- **Conclusion**

Overview

System profiling consists of:

- an *Instrumented Kernel* that logs many different types of *events*, as they happen
- tools for capturing and analyzing that log
- an optional API for controlling logging



Examples of event types:

- kernel calls
- process manager activity (e.g. process creation)
- interrupts
- rescheduling (thread state changes)
- context switches
- user-defined trace events

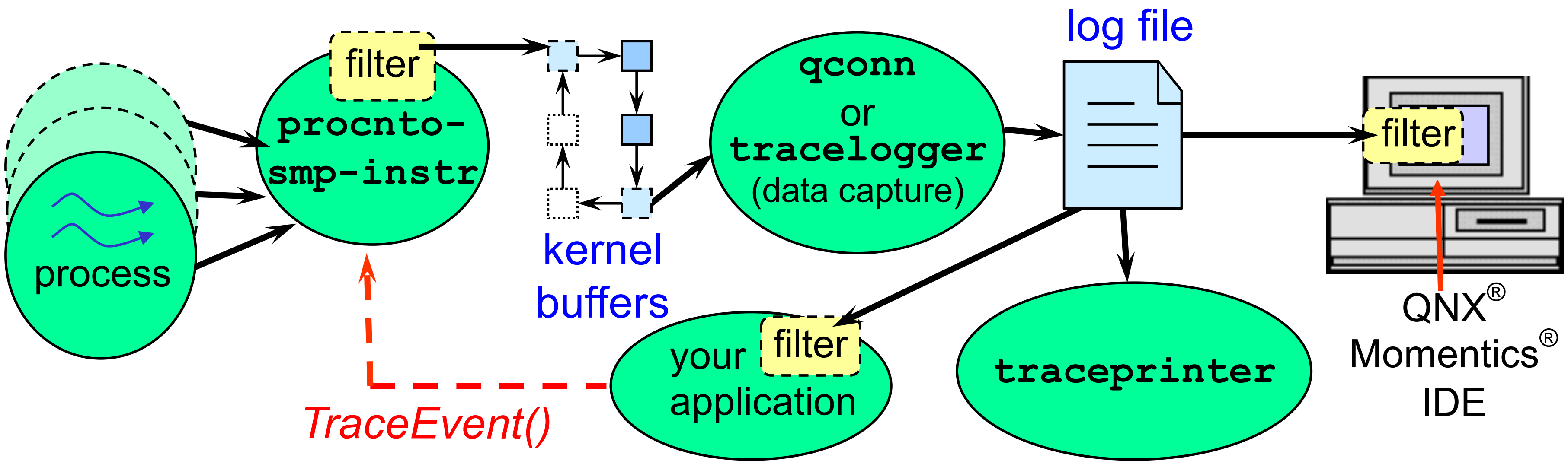
Overview

What system profiling does for you:

- gives you a way to analyze how the different processes and/or threads in your system interact
  - this goes beyond traditional debugging, which gives you a process level view, not a system level one
- gathers data for post-mortem analysis
- you can choose:
  - what types of events are logged
  - where the data is stored

Overview

Information logging involves several aspects:



- ways to control logging:
  - **`qconn`**
  - **`tracelogger`**
  - custom application (using **`TraceEvent()`**)
- choices for analyzing log:
  - Momentics® IDE System Profiler perspective
  - **`traceprinter`**
  - custom application (using traceparser API)

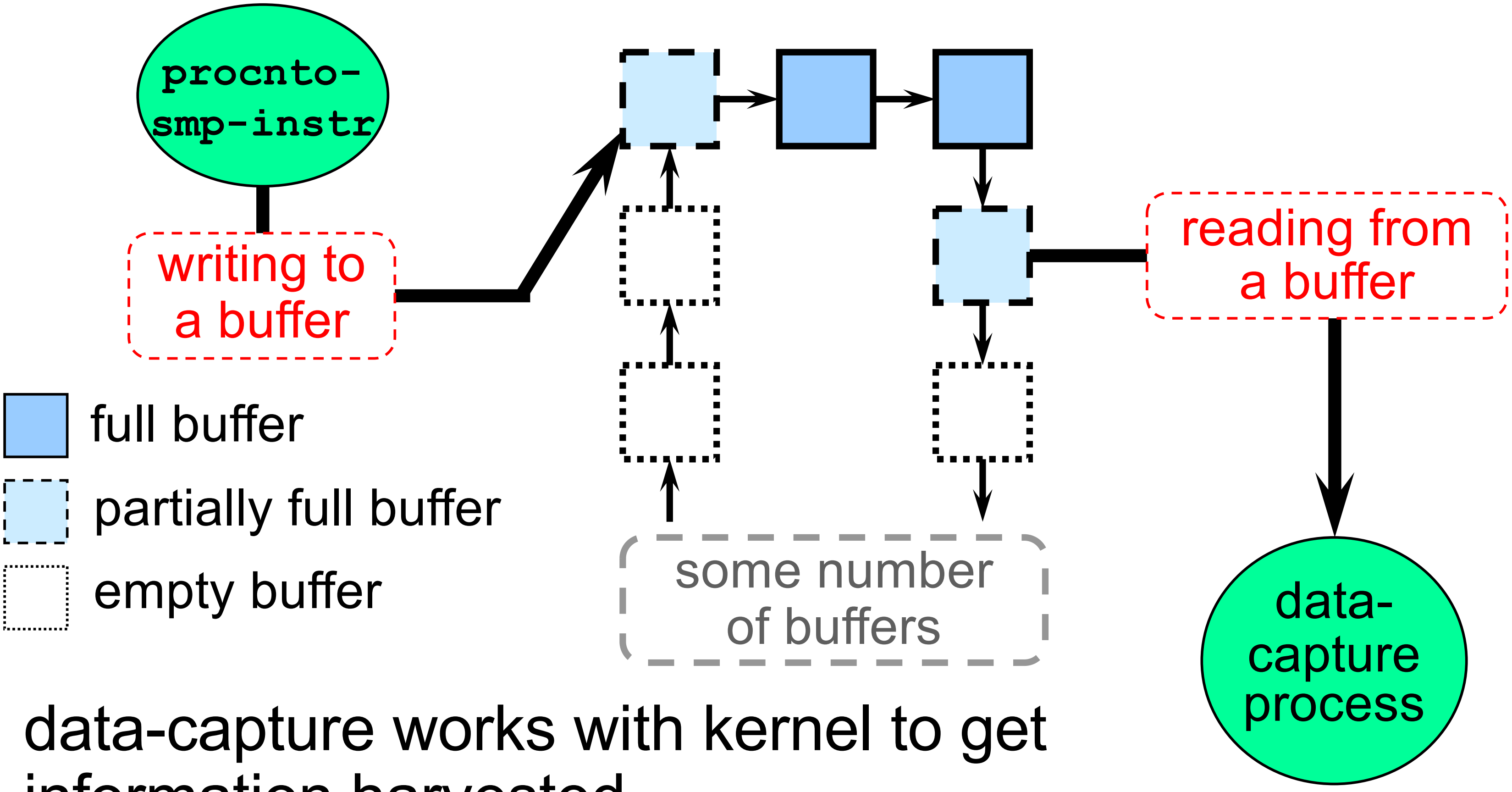
The events:

- an event will include the following:
  - what category the event belongs to (event class):
    - kernel call, interrupt servicing, process and thread management, user-generated
  - when it happened
  - event-specific data
- events can be fast or wide

	Amount of information	**Work for kernel**	**Size of log files**
Fast events	less	less	smaller
Wide events	more	more	larger

Kernel buffer management:

– events are stored in a circular list of buffers (a ring buffer)



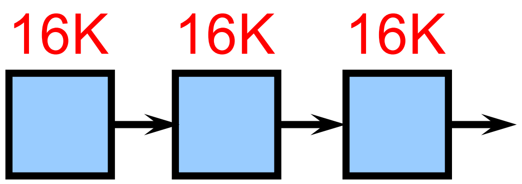
– data-capture works with kernel to get information harvested

continued... ↓

Overview - The instrumented kernel

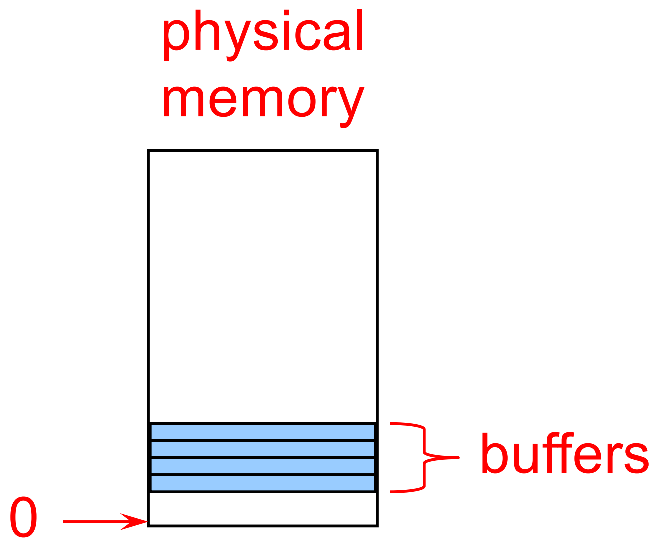
Kernel buffer management (continued):

- buffer size is fixed



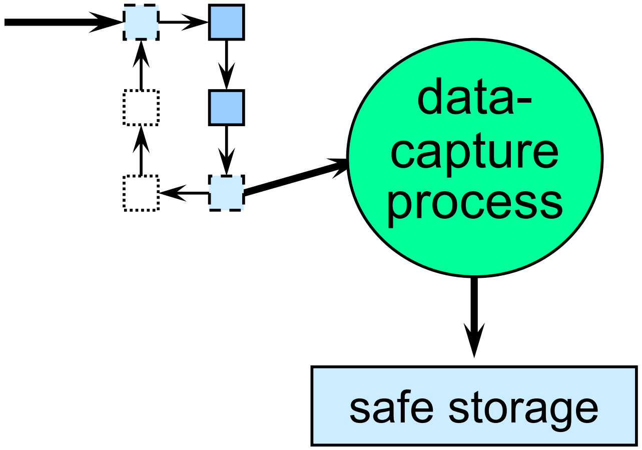
- the number of buffers that can be successfully created may vary over time

  ☞ the kernel allocates buffers from physically contiguous memory, if there isn't enough for the requested number of buffers then it will simply use less



The data-capture process:

- retrieves events logged in the kernel buffers
  - **`qconn`** is used to give data to the IDE
  - **`tracelogger`** is run for automated logging, or logging under program control
  - you can create your own data-capture process
- can do these three things:
  1. tell the kernel what sorts of events to log
  2. interface with the kernel
  3. move events from the kernel's event buffers into some other storage location

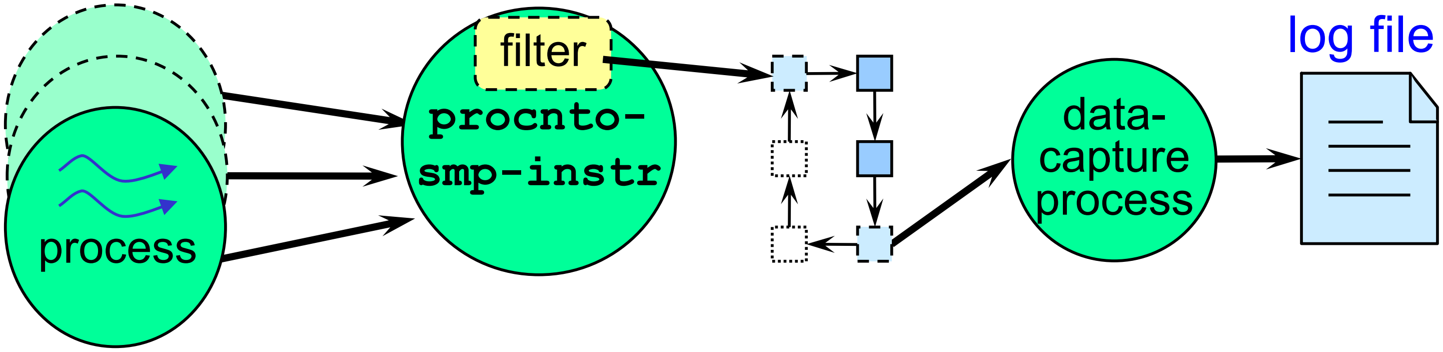


Overview - Data capture

Choosing which events to log:

- as you work on your analysis, you may find that only some data is helpful
  - in many cases, only some types of events are relevant
  - you might need only a subset of the event information
- you can make `procnto-smp-instr` filter out data
  - allows you to capture logs of longer time duration
  - improves system performance during capture
  - easier to digest the info (both for you and the IDE).

☞ no record of this data will be available



System Profiling

Topics:

Overview

→ **Creating a Log:**

- From the IDE
- Exercise
- Using tracelogger
- Under program control

Log Summary

A Quick Tour

Filtering Events and Event Owners

Navigating through a log with the Timeline Pane

Statistics

CPU Activity Pane and CPU Usage Pane

Tying The Trace to Your Code

…

Creating a log

Ways to create a log:

- from the IDE’s Target Navigator view
- from the command-line using **`tracelogger`**
- using **`tracelogger`**, but under program control

- if you want to use the System Profiler to analyze the log, it has to go in an IDE Project
  - doesn’t matter how the log was created, e.g. System Profiler can analyze log from **`tracelogger`**
  - a project is just a directory, so log can go into any one
  - some suggestions:
    - your QNX Target project
    - a project relevant to your problem

Topics:

Overview

Creating a Log:

→ – From the IDE

– Exercise

– Using tracelogger

– Under program control

Log Summary

A Quick Tour

Filtering Events and Event Owners

Navigating through a log with the Timeline Pane

Statistics

CPU Activity Pane and CPU Usage Pane

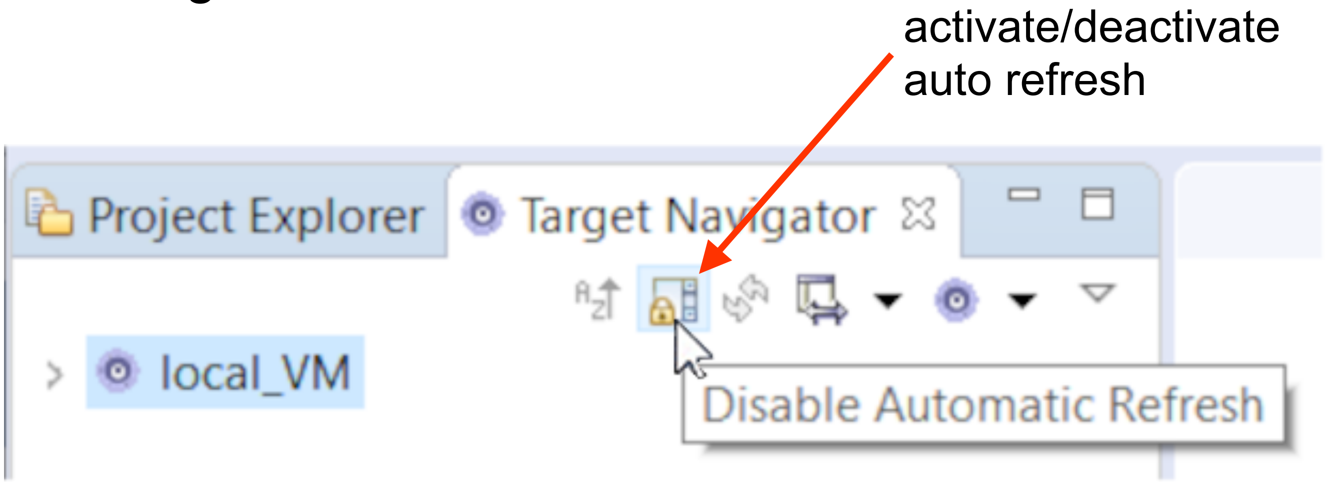
Tying The Trace to Your Code

…

Creating a Log from the IDE

Consider eliminating things that could interfere:

- the IDE periodically gathers system information
  - this generates irrelevant events



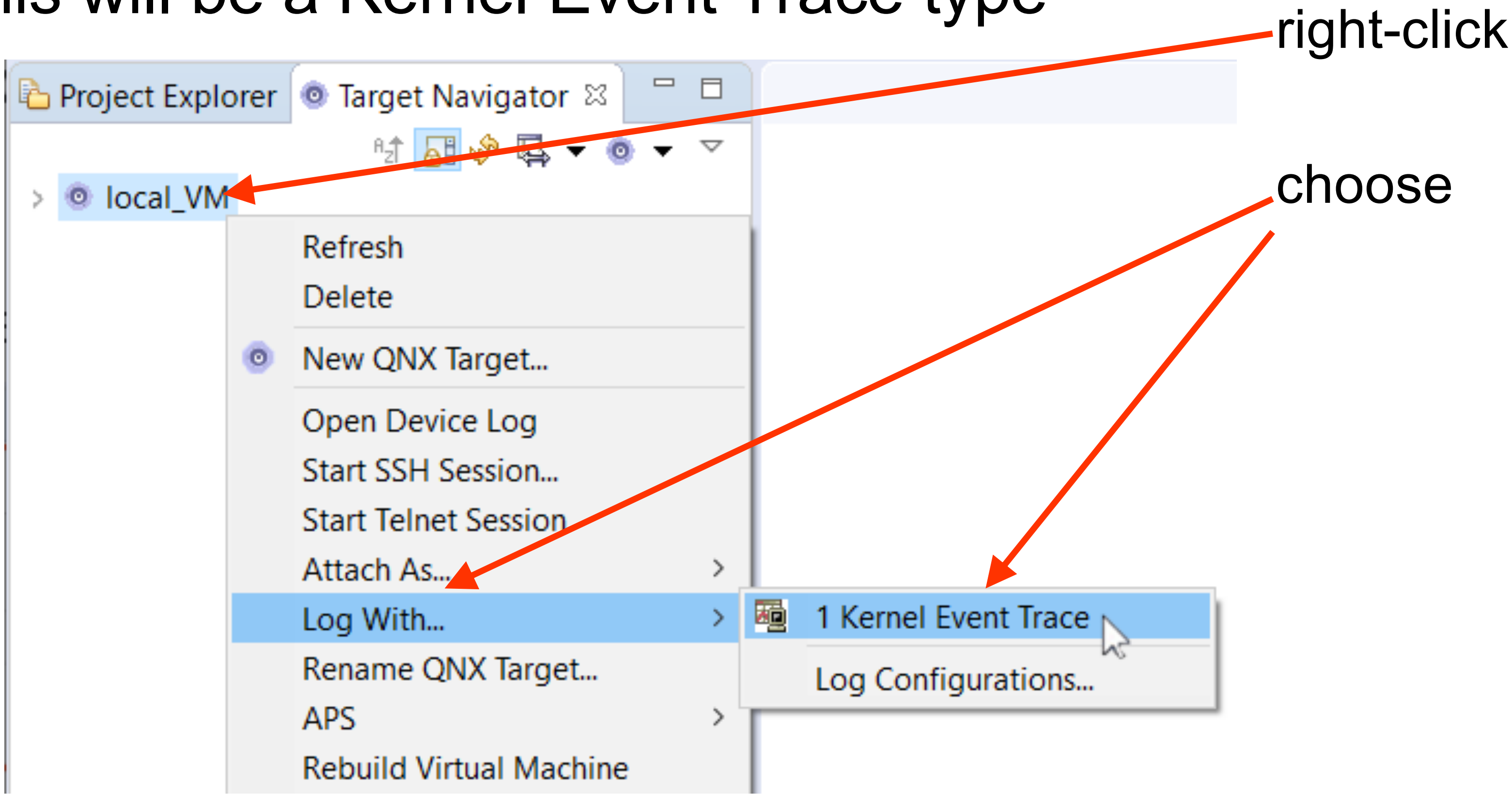
activate/deactivate auto refresh

- consider deactivating Automatic Refresh (at least during capture):

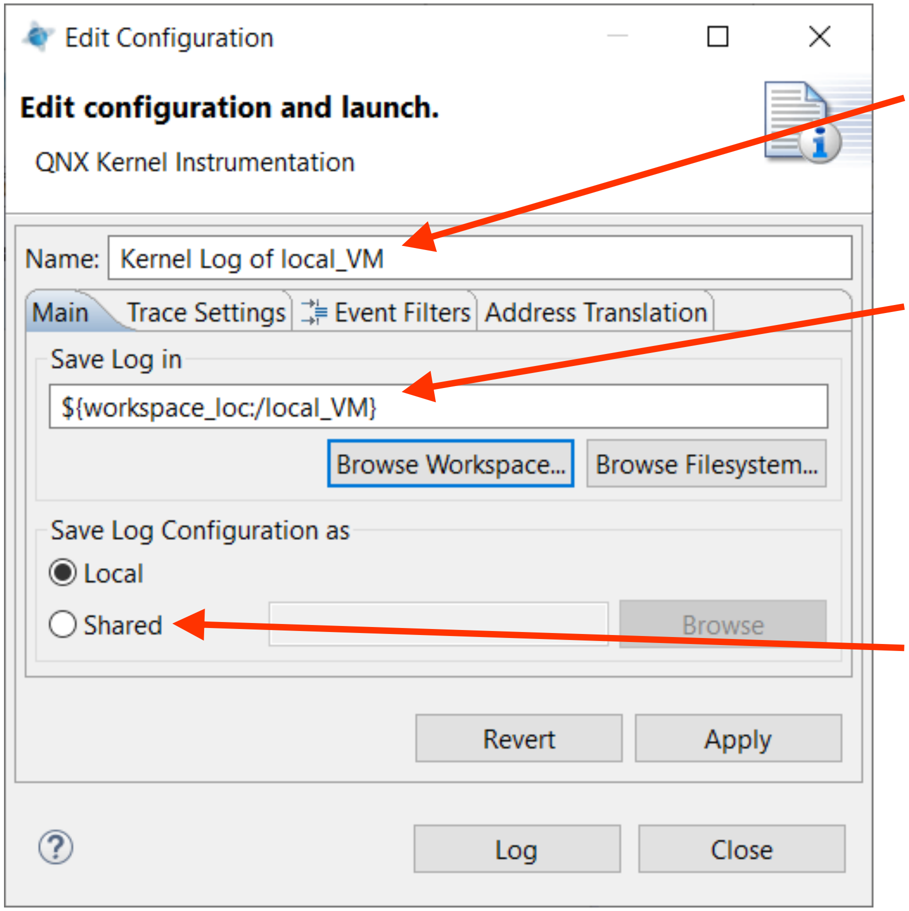
Creating a Log from the IDE

Next, create a Launch Configuration:

- the IDE uses launch configurations to remember logging settings as well as running programs
- this will be a Kernel Event Trace type



Log Configuration settings (Main tab):



Edit Configuration

Edit configuration and launch.

QNX Kernel Instrumentation

Name: Kernel Log of local_VM

Save Log in: ${workspace_loc:/local_VM}

Save Log Configuration as

● Local

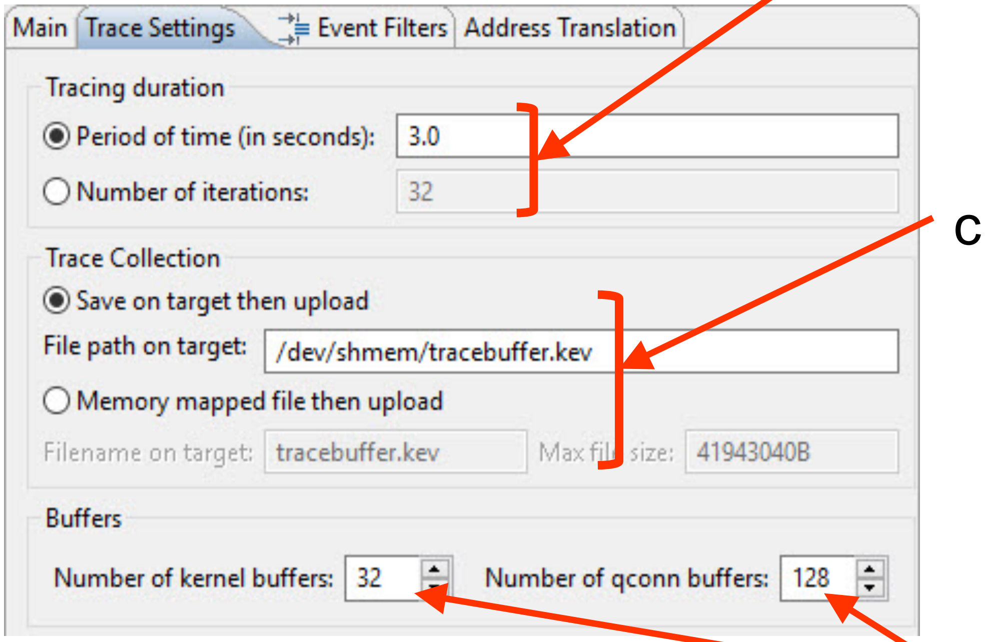
○ Shared

give the configuration a name

where to save log file

you can make it easier to give the log configuration to someone else by making it a shared configuration

Log Configuration settings (Trace Settings tab):



Main | Trace Settings | Event Filters | Address Translation

Tracing duration

● Period of time (in seconds): 3.0

○ Number of iterations: 32

Trace Collection

● Save on target then upload

File path on target: /dev/shmem/tracebuffer.kev

○ Memory mapped file then upload

Filename on target: tracebuffer.kev Max file size: 41943040B

Buffers

Number of kernel buffers: 32 Number of qconn buffers: 128

choose whether log capture completes after:

- specified time is reached, or
- specified amount of data is captured

choose where the log is stored:

- file on target, or
- Memory mapped file -- useful if no storage on target

the default buffer settings are usually fine

Creating a Log from the IDE

Log Configuration settings (Event Filters tab):

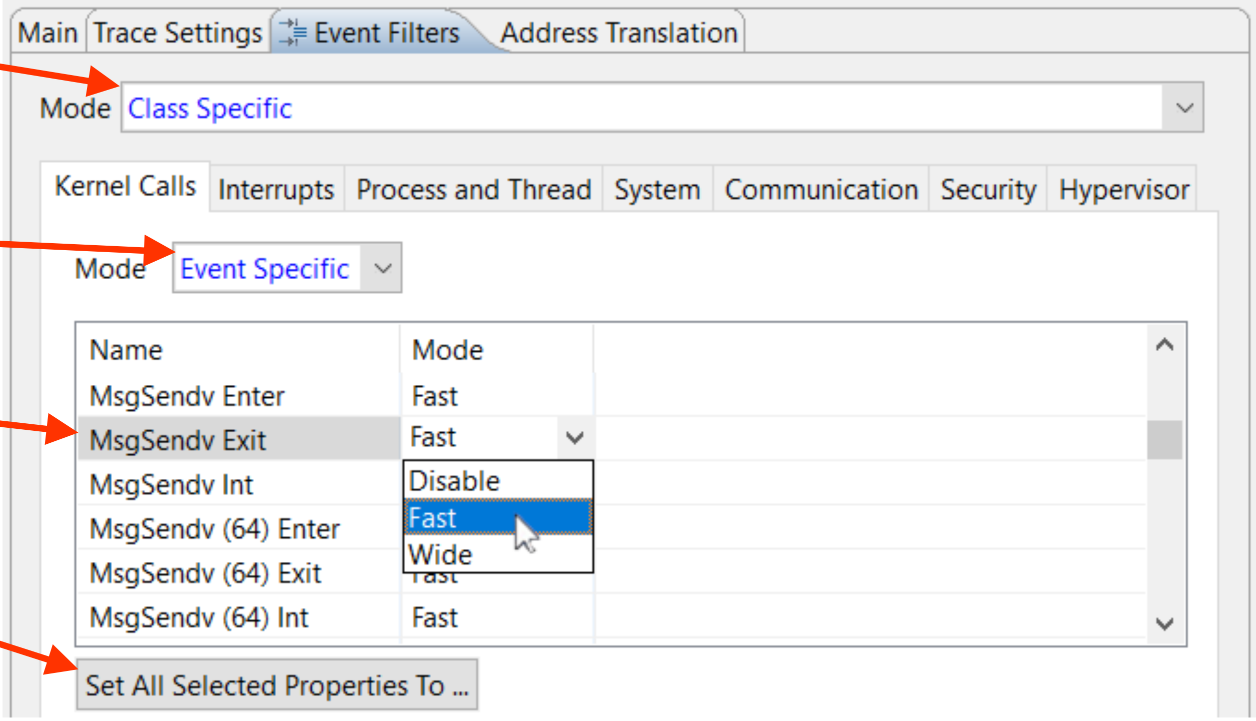
- works as a hierarchy of choices

Class Specific: enables setting modes within the tabs

Event Specific: enables setting modes for events within this class

you can configure specific events here, but only if Event Specific is set above

Or, after multi-select, you can set all selected at once



Main | Trace Settings | Event Filters | Address Translation

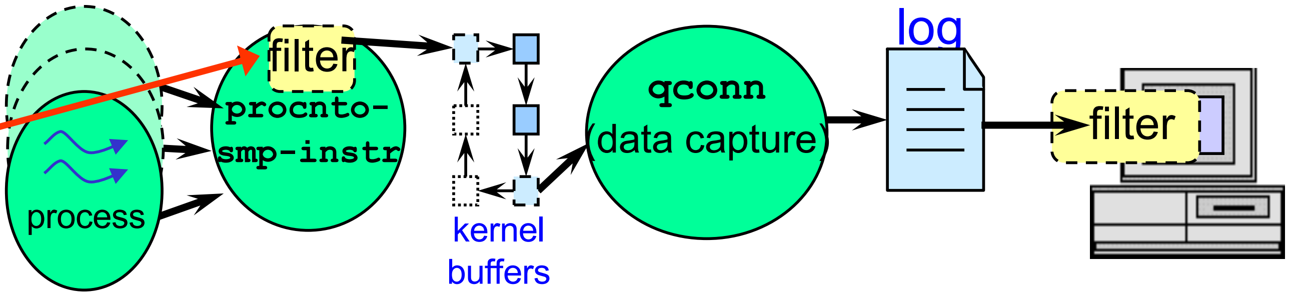
Mode: Class Specific

Kernel Calls | Interrupts | Process and Thread | System | Communication | Security | Hypervisor

Mode: Event Specific

Name	Mode
MsgSendv Enter	Fast
MsgSendv Exit	Fast
MsgSendv Int	
MsgSendv (64) Enter	
MsgSendv (64) Exit	Fast
MsgSendv (64) Int	Fast

this filtering affects this stage of the logging



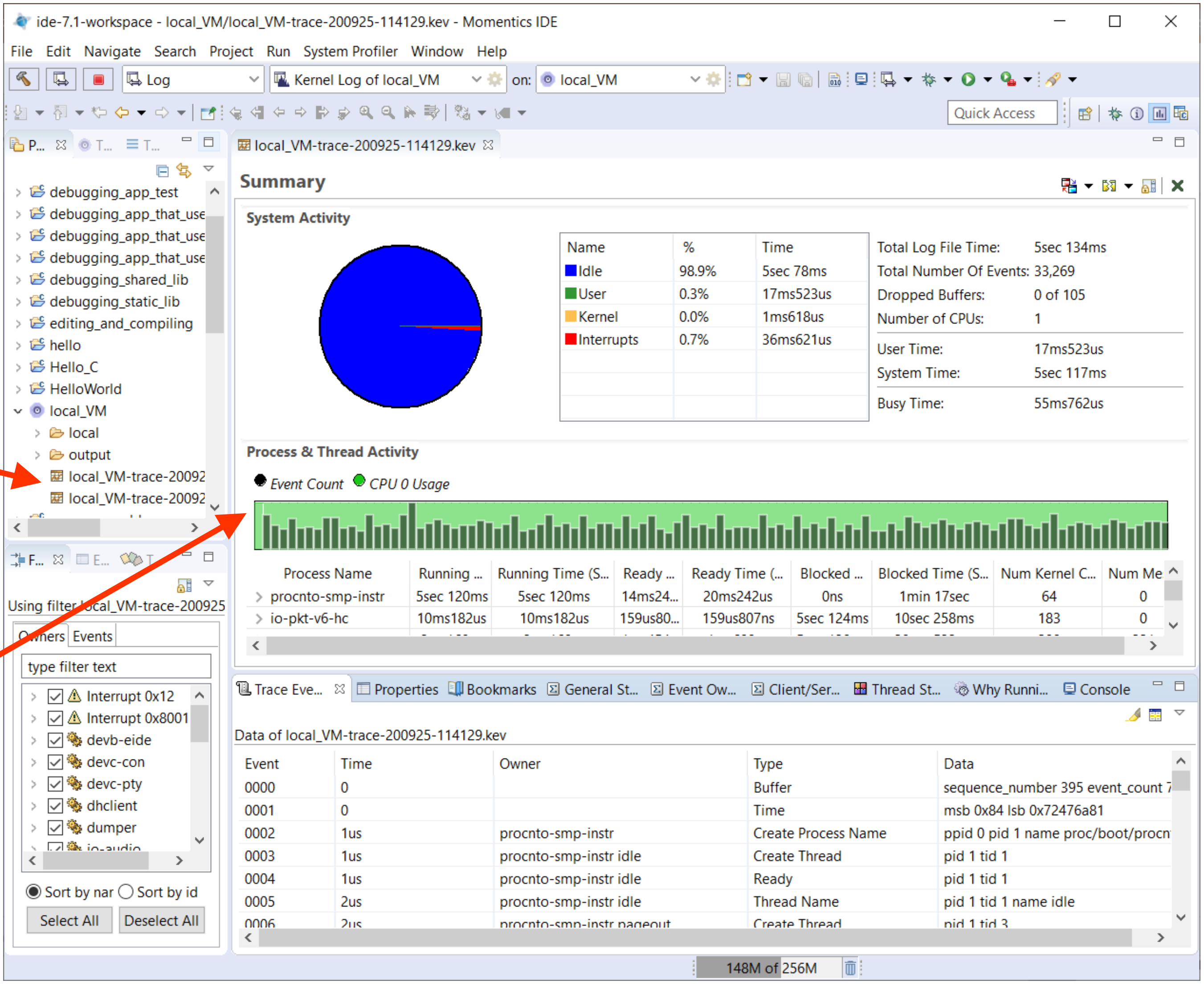
Creating a Log from the IDE

The resulting log file displayed in the IDE:

resulting log file is in the Target System project

double-click on it to open…

…Summary is displayed to give you an overview of the log



Name	%	Time
🟦 Idle	98.9%	5sec 78ms
🟩 User	0.3%	17ms523us
🟧 Kernel	0.0%	1ms618us
🟥 Interrupts	0.7%	36ms621us

Total Log File Time:	5sec 134ms
Total Number Of Events:	33,269
Dropped Buffers:	0 of 105
Number of CPUs:	1
User Time:	17ms523us
System Time:	5sec 117ms
Busy Time:	55ms762us

Process Name	Running ...	Running Time (S...	Ready ...	Ready Time (...	Blocked ...	Blocked Time (S...	Num Kernel C...	Num Me
procnto-smp-instr	5sec 120ms	5sec 120ms	14ms24...	20ms242us	0ns	1min 17sec	64	0
io-pkt-v6-hc	10ms182us	10ms182us	159us80...	159us807ns	5sec 124ms	10sec 258ms	183	0

Data of local_VM-trace-200925-114129.kev

Event	Time	Owner	Type	Data
0000	0		Buffer	sequence_number 395 event_count 7
0001	0		Time	msb 0x84 lsb 0x72476a81
0002	1us	procnto-smp-instr	Create Process Name	ppid 0 pid 1 name proc/boot/procn
0003	1us	procnto-smp-instr idle	Create Thread	pid 1 tid 1
0004	1us	procnto-smp-instr idle	Ready	pid 1 tid 1
0005	2us	procnto-smp-instr idle	Thread Name	pid 1 tid 1 name idle
0006	2us	procnto-smp-instr pageout	Create Thread	pid 1 tid 3

Topics:

- **Overview**
- **Creating a Log:**
  - – From the IDE
  - → – Exercise
  - – Using tracelogger
  - – Under program control
- **Log Summary**
- **A Quick Tour**
- **Filtering Events and Event Owners**
- **Navigating through a log with the Timeline Pane**
- **Statistics**
- **CPU Activity Pane and CPU Usage Pane**
- **Tying The Trace to Your Code**
- **…**

EXERCISE - Creating a Log from the IDE

Experiment with log creation:

1. capture a 1 second log
   - pay particular attention to the configuration of the log capture
2. open it in the system profiler
   - don't worry too much about what you are looking at, we'll get into that later
3. capture another 1 second log
   - this time, use filtering to tell the kernel not to log interrupt or kernel call event classes
   - compare the log sizes (without and with filtering)
     - (right-click on log file, choose properties)

Topics:

Overview

Creating a Log:

- From the IDE
- Exercise
- Using tracelogger
- Under program control

Log Summary

A Quick Tour

Filtering Events and Event Owners

Navigating through a log with the Timeline Pane

Statistics

CPU Activity Pane and CPU Usage Pane

Tying The Trace to Your Code

…

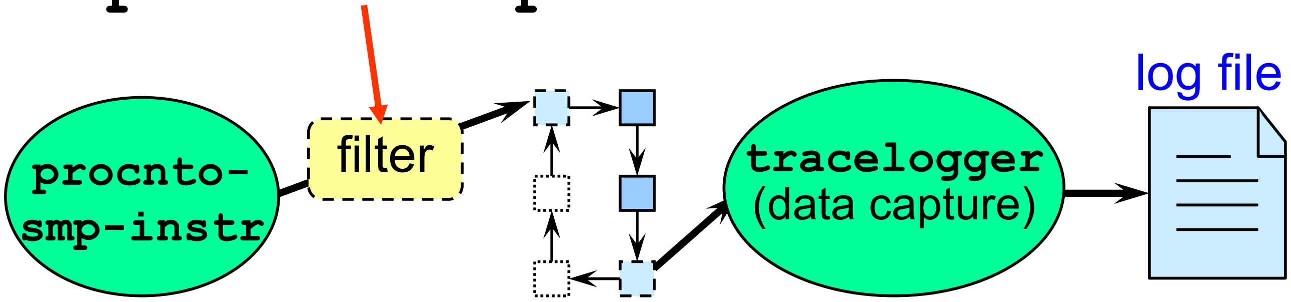
Creating a Log using tracelogger

`tracelogger` is useful if:

- you don't have the IDE handy
  - e.g. to capture what happened during system initialization
- you want to create the log under program control

Working with **`tracelogger`** you can:

- control how, where and when data is logged
- set **`procnto-smp-instr`**'s event filter



tracelogger modes

`tracelogger` can be run in 5 different modes:

- `-n iterations` : (the default), log for a specified number of kernel buffers (default 32)
- `-s seconds` : log for the specified number of seconds
- `-c` : continuous, log until tracelogger is terminated (SIGINT preferred)
- `-d1` : daemon mode, run in background waiting for program to control things
- `-r` : ring mode, generate data, but don’t dump until later

We’ll look at daemon and ring mode in more detail.

Creating a Log using tracelogger

Running **`tracelogger`**:

– example command-line:

```
tracelogger -f /dev/shmem/mylog.kev -s3
```

file to log to

how many sec. to log for

Topics:

Overview

Creating a Log:

- From the IDE
- Exercise
- Using tracelogger
- → Under program control

Log Summary

A Quick Tour

Filtering Events and Event Owners

Navigating through a log with the Timeline Pane

Statistics

CPU Activity Pane and CPU Usage Pane

Tying The Trace to Your Code

…

Program Control of Log Creation

`tracelogger` can do logging when told to by a program (daemon mode):

```
tracelogger -d1 -E -f logfile.kev
```

– can configure basic filtering, fast/wide, etc on the command line
  - the `-E` option allows this

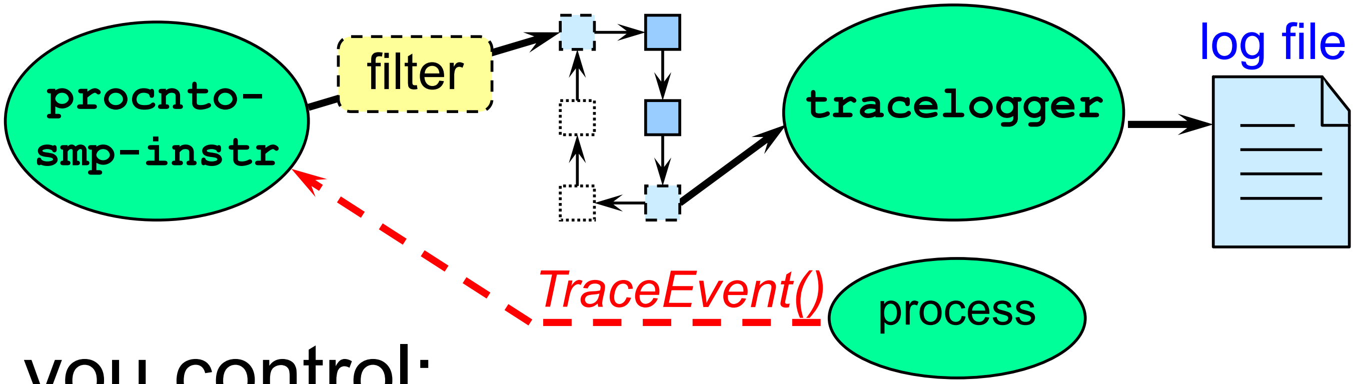
– detailed configuration and control can be done using *TraceEvent()* calls

– event logging begins only after the appropriate *TraceEvent()* call is made

Program Control of Log Creation

TraceEvent():

- manages the flow of data to **`tracelogger`**



- lets you control:
  - when to start/stop logging
  - what events to generate/filter
  - how many data buffers **`procnto-smp-instr`** is asked to use
  - fast vs. wide events
  - where data is sent (**`/dev/shmem/tracebuffer`** by default)
- can also be used to insert your own events

Program Control of Log Creation

Controlling the logging

- start logging by the kernel

```
TraceEvent (_NTO_TRACE_START);
```

- stop logging:

```
TraceEvent (_NTO_TRACE_STOP);
```

Ring mode vs. linear mode

- **`-r`** option sets ring mode; default is linear
- ring mode
  - kernel continuously goes around its ring, storing events
  - events only read by **`tracelogger`** on demand, when:
    - a program does **`TraceEvent(_NTO_TRACE_STOP);`**
    - **`tracelogger`** receives SIGINT
  - good for situations where you don't know in advance when you might need to analyze events, e.g. SIGSEGV
    - when SIGSEGV happens, have **`tracelogger`** flush the buffers, then look back in the log
  - you can spawn **`tracelogger`**, then do **`kill(tracelogger_PID, SIGINT)`** to stop logging

continued...

Program Control of Log Creation

Ring mode vs. linear mode (continued)

- linear mode
  - kernel continuously goes around its ring storing events, but when each buffer becomes full, it notifies **`tracelogger`** to read the data
  - good for situations where you know something interesting will happen at some point in the future, and you want to start **`tracelogger`** storing events

☞ the kernel always stores events in its ring of buffers, regardless of the linear vs. ring option

Program Control of Log Creation

For more information on creating a log under program control:

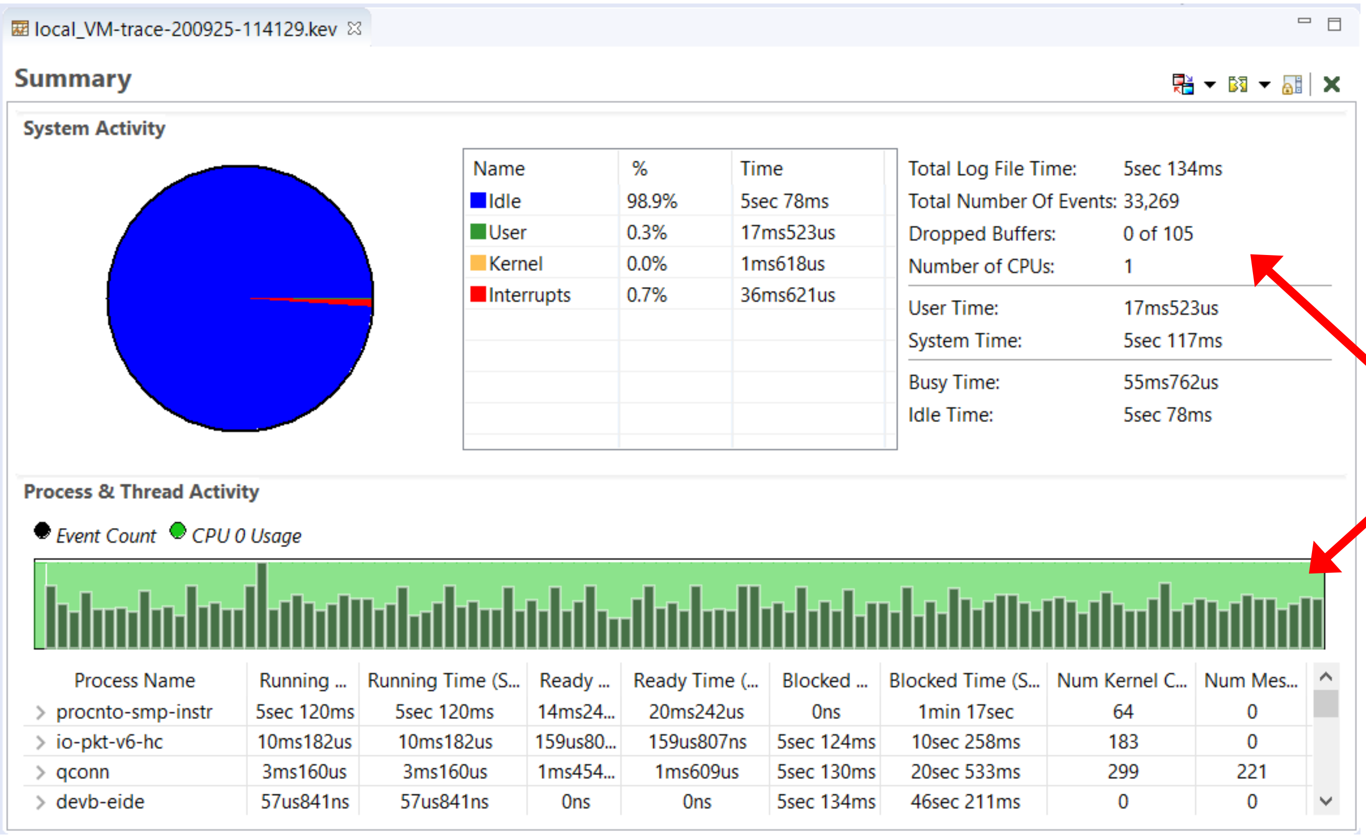
- have a look at the TraceEvent() function documentation in the Library Reference documentation
- as well as the System Analysis Toolkit User’s Guide

Topics:

- Overview
- Creating a Log
- → Log Summary
- A Quick Tour
- Filtering Events and Event Owners
- Navigating through a log with the Timeline Pane
- Statistics
- CPU Activity Pane and CPU Usage Pane
- Tying The Trace to Your Code
- Multi-core Related Features
- Adaptive Partitioning: Partition Summary
- Conclusion

Event Log Summary

After opening a log file, first thing you see is summary information:



local_VM-trace-200925-114129.kev

Summary

System Activity

Name	%	Time
🟦 Idle	98.9%	5sec 78ms
🟩 User	0.3%	17ms523us
🟧 Kernel	0.0%	1ms618us
🟥 Interrupts	0.7%	36ms621us

Total Log File Time:	5sec 134ms
Total Number Of Events:	33,269
Dropped Buffers:	0 of 105
Number of CPUs:	1
User Time:	17ms523us
System Time:	5sec 117ms
Busy Time:	55ms762us
Idle Time:	5sec 78ms

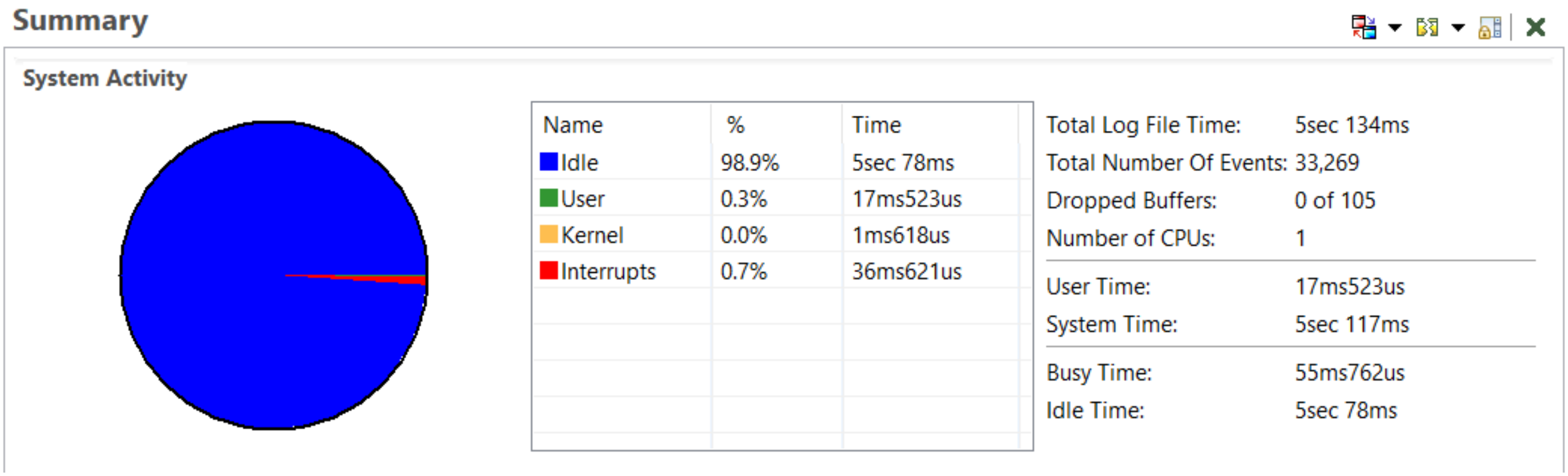
Process & Thread Activity

Process Name	Running ...	Running Time (S...	Ready ...	Ready Time (...	Blocked ...	Blocked Time (S...	Num Kernel C...	Num Mes...
procnto-smp-instr	5sec 120ms	5sec 120ms	14ms24...	20ms242us	0ns	1min 17sec	64	0
io-pkt-v6-hc	10ms182us	10ms182us	159us80...	159us807ns	5sec 124ms	10sec 258ms	183	0
qconn	3ms160us	3ms160us	1ms454...	1ms609us	5sec 130ms	20sec 533ms	299	221
devb-eide	57us841ns	57us841ns	0ns	0ns	5sec 134ms	46sec 211ms	0	0

let’s look at what these indicate…

Summary: System Activity

– indicates how busy the system was during the log



Summary

System Activity

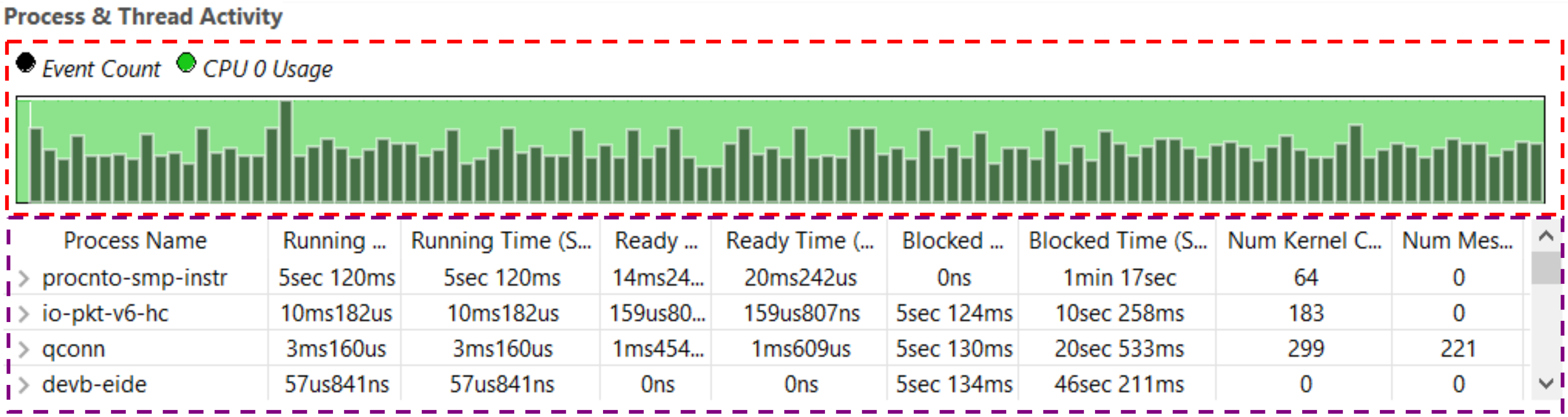
Name	%	Time
🟦 Idle	98.9%	5sec 78ms
🟩 User	0.3%	17ms523us
🟧 Kernel	0.0%	1ms618us
🟥 Interrupts	0.7%	36ms621us

Total Log File Time:	5sec 134ms
Total Number Of Events:	33,269
Dropped Buffers:	0 of 105
Number of CPUs:	1
User Time:	17ms523us
System Time:	5sec 117ms
Busy Time:	55ms762us
Idle Time:	5sec 78ms

- Idle is the time that the idle thread (or threads) was (were) executing
- User is the time spent in threads in processes
- System is time spent in the kernel
- Interrupts is time spent handling interrupts
  - this should generally be fairly low; a high load could indicate:
    - faulty hardware
    - bad driver
    - application profiling for too many applications

Summary: Process & Thread Activity

- shows change in CPU usage and event rate



Process Name	Running ...	Running Time (S...	Ready ...	Ready Time (...	Blocked ...	Blocked Time (S...	Num Kernel C...	Num Mes...
> procnto-smp-instr	5sec 120ms	5sec 120ms	14ms24...	20ms242us	0ns	1min 17sec	64	0
> io-pkt-v6-hc	10ms182us	10ms182us	159us80...	159us807ns	5sec 124ms	10sec 258ms	183	0
> qconn	3ms160us	3ms160us	1ms454...	1ms609us	5sec 130ms	20sec 533ms	299	221
> devb-eide	57us841ns	57us841ns	0ns	0ns	5sec 134ms	46sec 211ms	0	0

- table provides metrics on how busy each individual process/thread was

Topics:

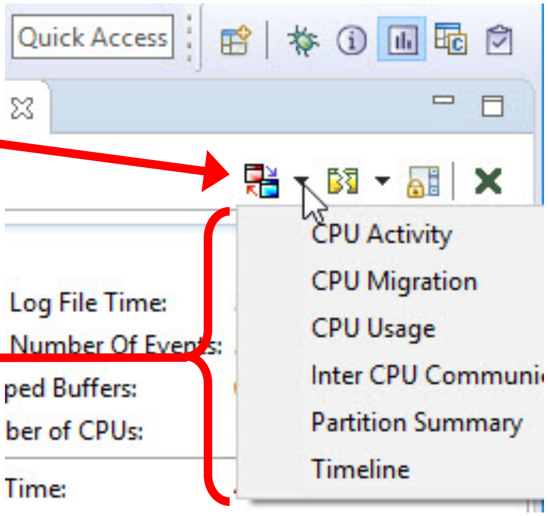
- Overview
- Creating a Log
- Log Summary
- → A Quick Tour
- Filtering Events and Event Owners
- Navigating through a log with the Timeline Pane
- Statistics
- CPU Activity Pane and CPU Usage Pane
- Tying The Trace to Your Code
- Multi-core Related Features
- Adaptive Partitioning: Partition Summary
- Conclusion

Switching Panes:

- you can switch to different ‘Panes’ in the System Profiler that give you different methods for visualizing trace data

click this icon to switch the display

let’s go through the different displays…



Switching Panes

The different display types:

- **Summary:** gives an overall picture of the log
- **CPU Activity:** tracks total CPU usage over time
- **CPU Migration:** provides information about how the kernel migrated threads between CPUs in a multi-core or multi-CPU system
- **CPU Usage:** shows the average % CPU usage, per process (or thread), for the time period in the display
- **Inter-CPU Communication:** indicates how many times the kernel migrated threads due to message passing
- **Partition Summary:** shows how much CPU usage each scheduling partition consumed
- **Timeline**: graphically shows timing of events

We'll also be looking at various views

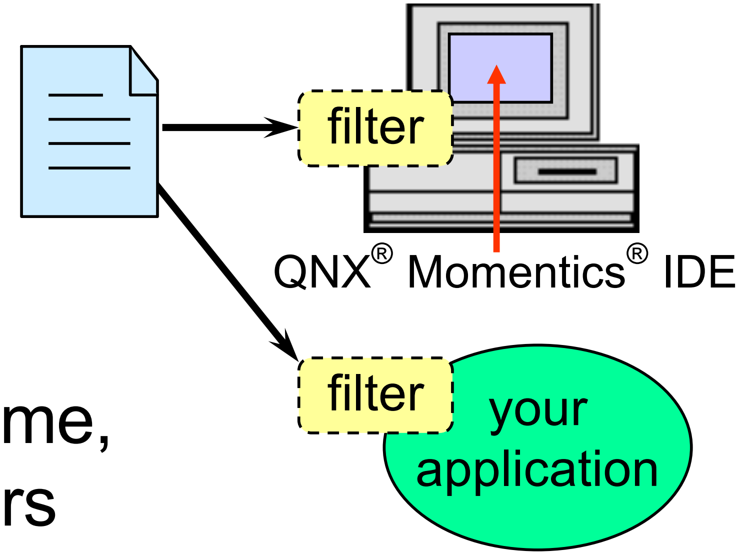
Topics:

- Overview
- Creating a Log
- Log Summary
- A Quick Tour
- → Filtering Events and Event Owners
  - Exercise
- Navigating through a log with the Timeline Pane
- Statistics
- CPU Activity Pane and CPU Usage Pane
- Tying The Trace to Your Code
- Multi-core Related Features
- Adaptive Partitioning: Partition Summary
- Conclusion

Filtering Events

You can filter events during data-analysis:

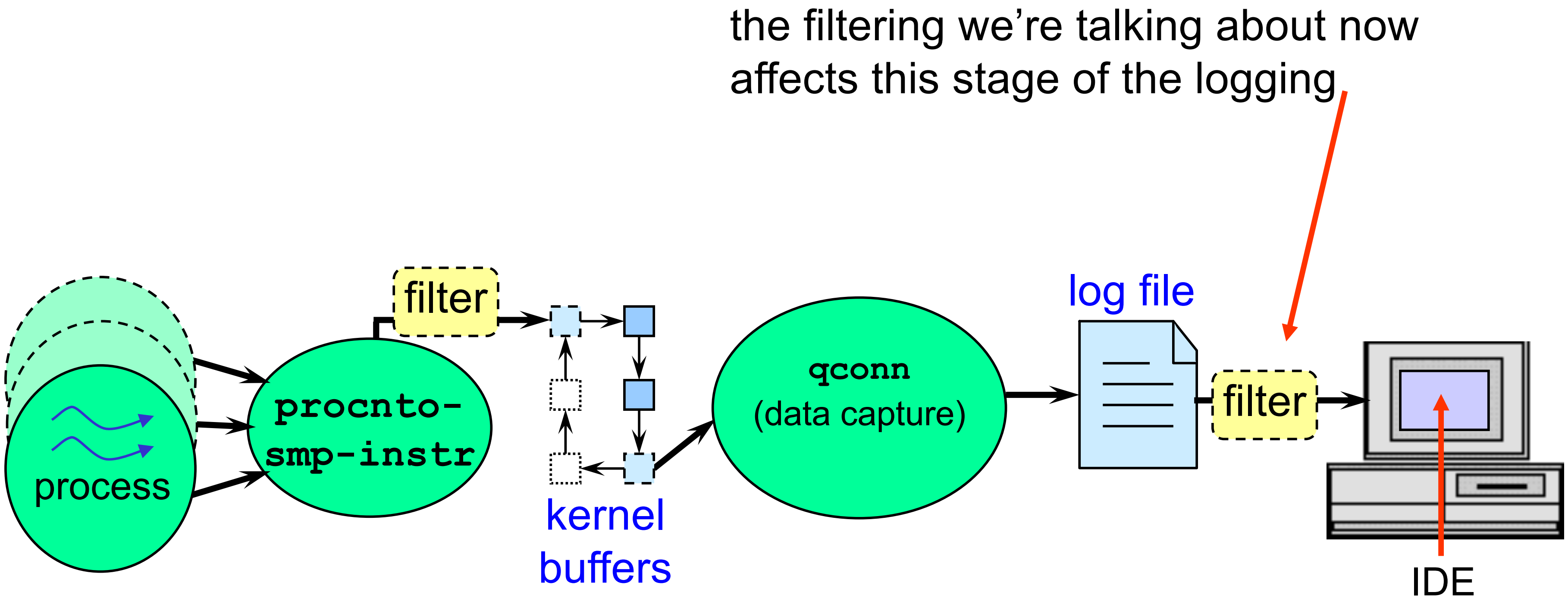
– this is useful when you want to look at a subset of your captured data



- the IDE allows you to filter out events based on type, time, and various other parameters
- the *traceparser*() library allows similar control for applications you write yourself

☞ events filtered out at this stage are not lost, unless you save the filtered log file

Kernel filter vs. System Profiler filter:

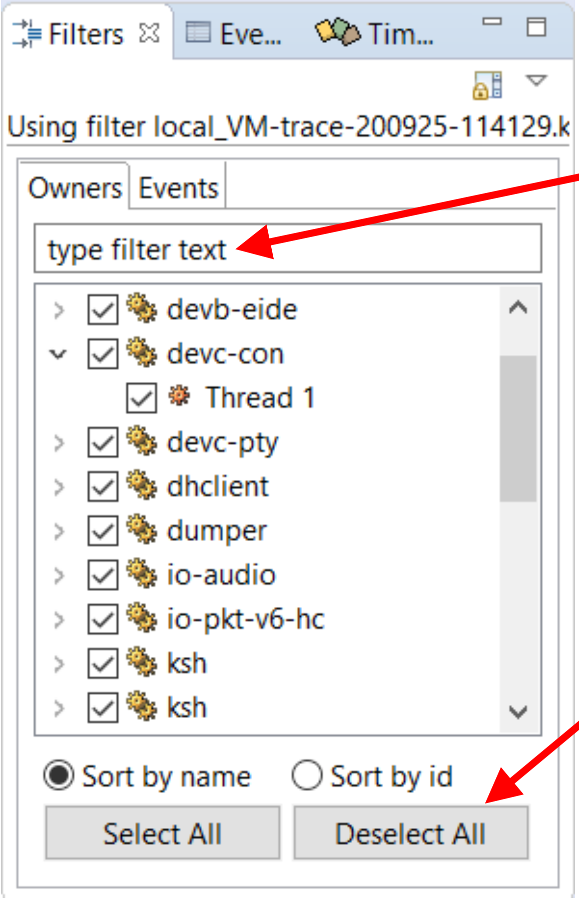


Event Owner Filter vs. Event Filter:

- Event Owner Filter
  - filters in or out events based on the owner of the event
  - event owners are:
    - processes
    - threads
    - interrupts
  - e.g. show only events for `procnto` and `HelloWorld`, or remove events for `procnto`, `qconn` and `io-pkt`
- Event Filter
  - filters in or out certain types of events
  - e.g. show only message-passing events

Owner Filters:

- to remove event owners you don’t care about uncheck in ‘Owners’ tab

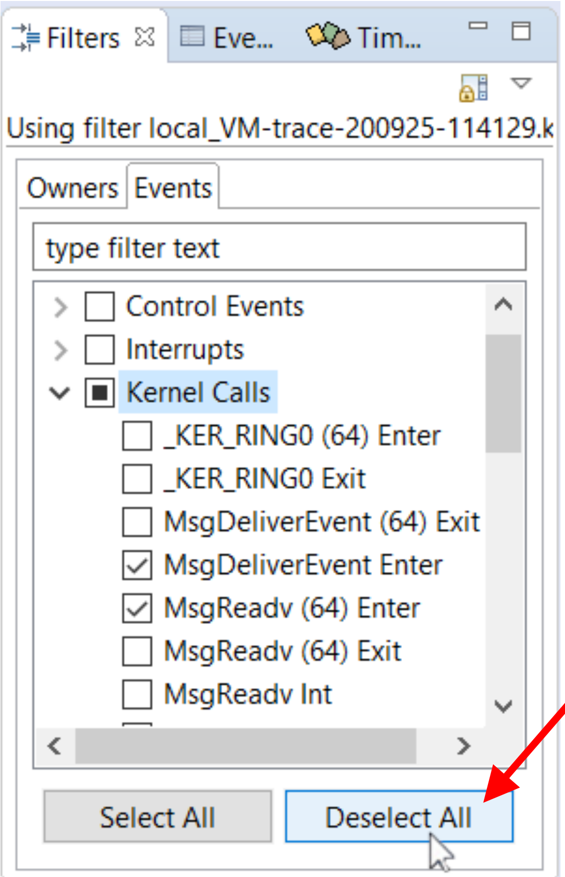


in a large list of processes, you can type some text to find the processes you want

if you just want to see a few owners, remove everything then add back in

Event Filters:

– select/select event classes or specific event types you want or don't want



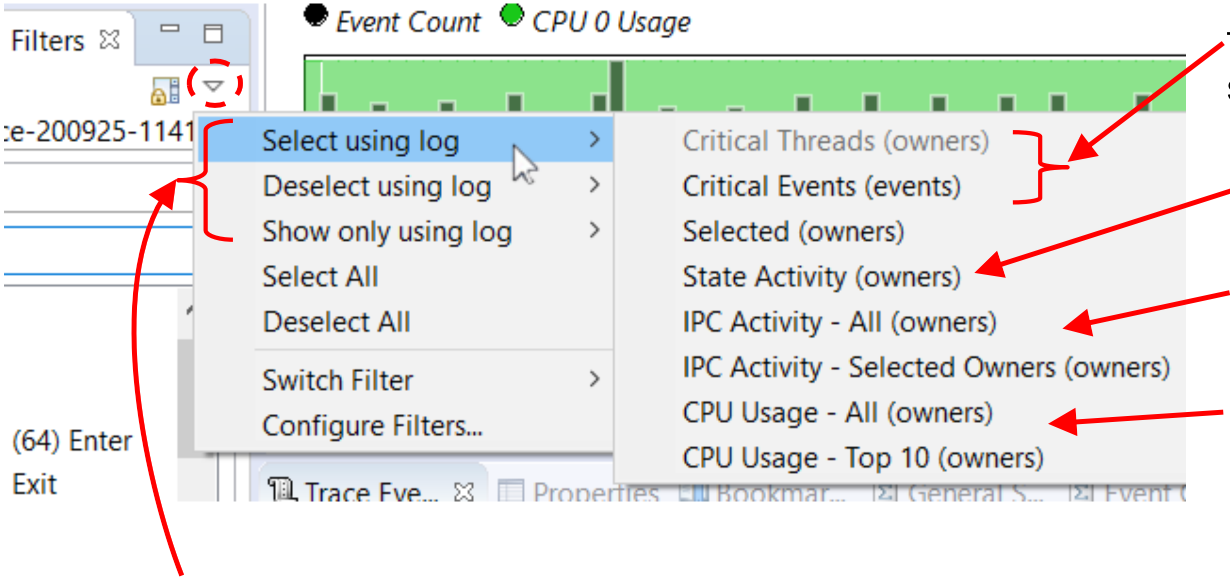
Using filter local_VM-trace-200925-114129.k

- □ Control Events
- □ Interrupts
- ■ Kernel Calls
  - □ _KER_RING0 (64) Enter
  - □ _KER_RING0 Exit
  - □ MsgDeliverEvent (64) Exit
  - ☑ MsgDeliverEvent Enter
  - ☑ MsgReadv (64) Enter
  - □ MsgReadv (64) Exit
  - □ MsgReadv Int

it's useful to be able to deselect all events and start from nothing, adding in events of interest

Built-In Filters:

– a collection of useful pre-existing filters



for finding areas of concern in a system with Adaptive Partitioning

threads that changed state e.g. SEND→REPLY

threads that were involved in message passing

threads that consumed CPU time

– how to apply the filter:

- Select using – add data that matches filter to data set
- Deselect using – remove data that matches the filter from the data set
- Show only – only include data that matches the filter in the data set

Topics:

- Overview
- Creating a Log
- Log Summary
- A Quick Tour
- Filtering Events and Event Owners
  - → – Exercise
- Navigating through a log with the Timeline Pane
- Statistics
- CPU Activity Pane and CPU Usage Pane
- Tying The Trace to Your Code
- Multi-core Related Features
- Adaptive Partitioning: Partition Summary
- Conclusion

EXERCISE – Filtering

Try out the various filters.

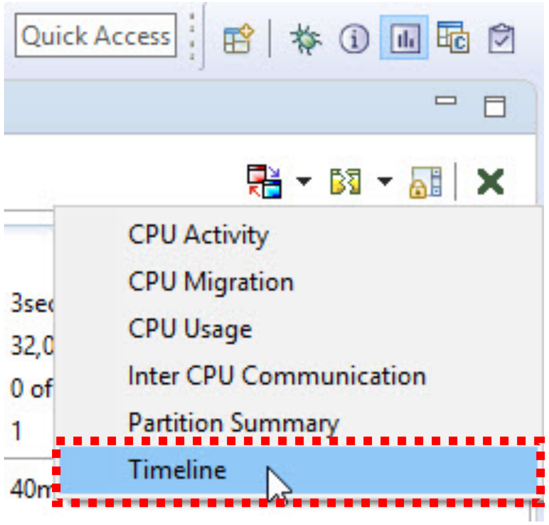
Topics:

- Overview
- Creating a Log
- Log Summary
- A Quick Tour
- Filtering Events and Event Owners
- → Navigating through a log with the Timeline Pane
  - – Exercise
- Statistics
- CPU Activity Pane and CPU Usage Pane
- Tying The Trace to Your Code
- Multi-core Related Features
- Adaptive Partitioning: Partition Summary
- Conclusion

The Timeline

The Timeline pane is probably the best visualization tool:

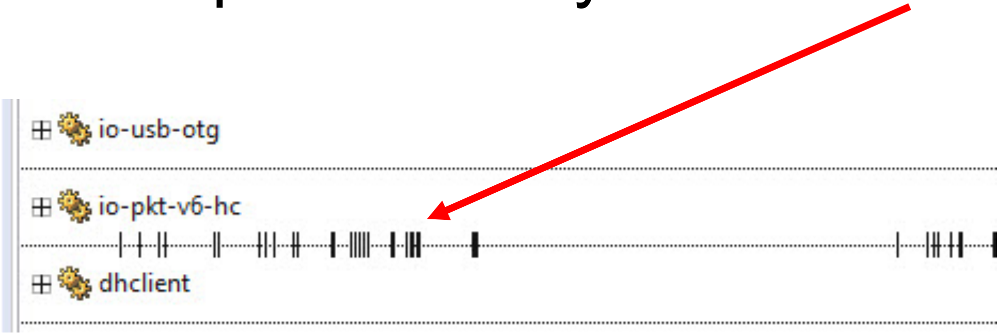
- Let’s look at it…



The Timeline

The timeline graphically shows the timing of:

- QNX native message passing
- thread states
- events that occurred, .e.g.
  - interrupts
  - entry into all kernel calls made
  - many others
- events are represented by vertical ‘ticks’



Timeline - Zooming

Zooming in/out:

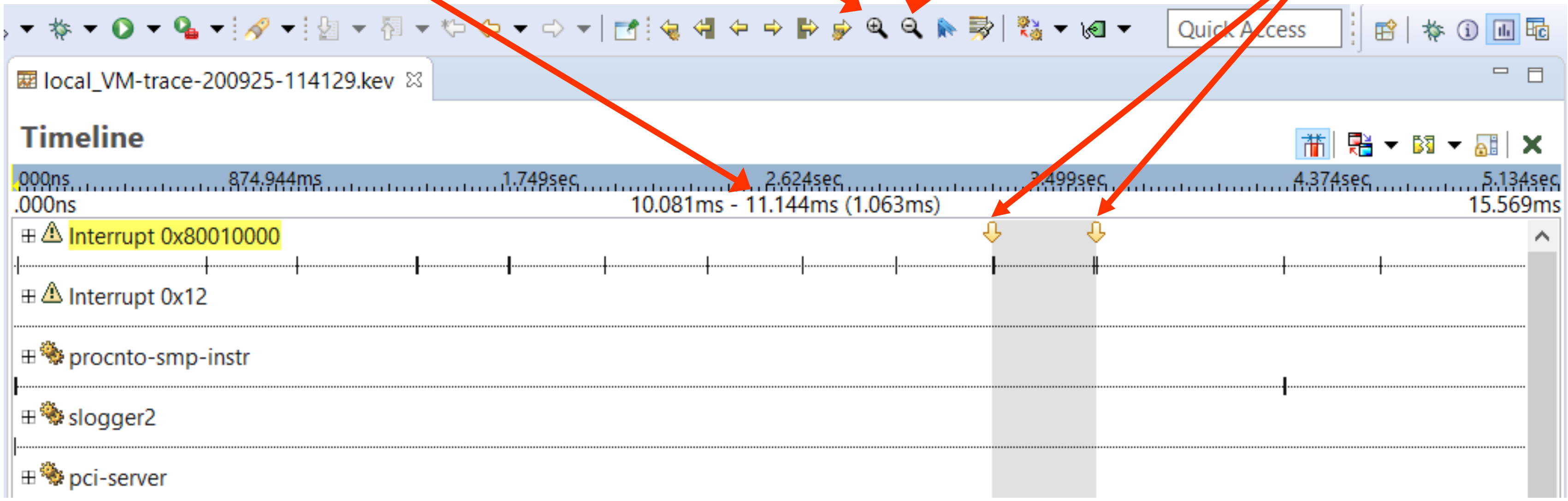
to zoom in,
select the range
you want to magnify

time range of
your selection
is shown here

clicking on
the + magnifying
glass zooms in

the -
zooms out

it automatically
snaps
to the nearest event

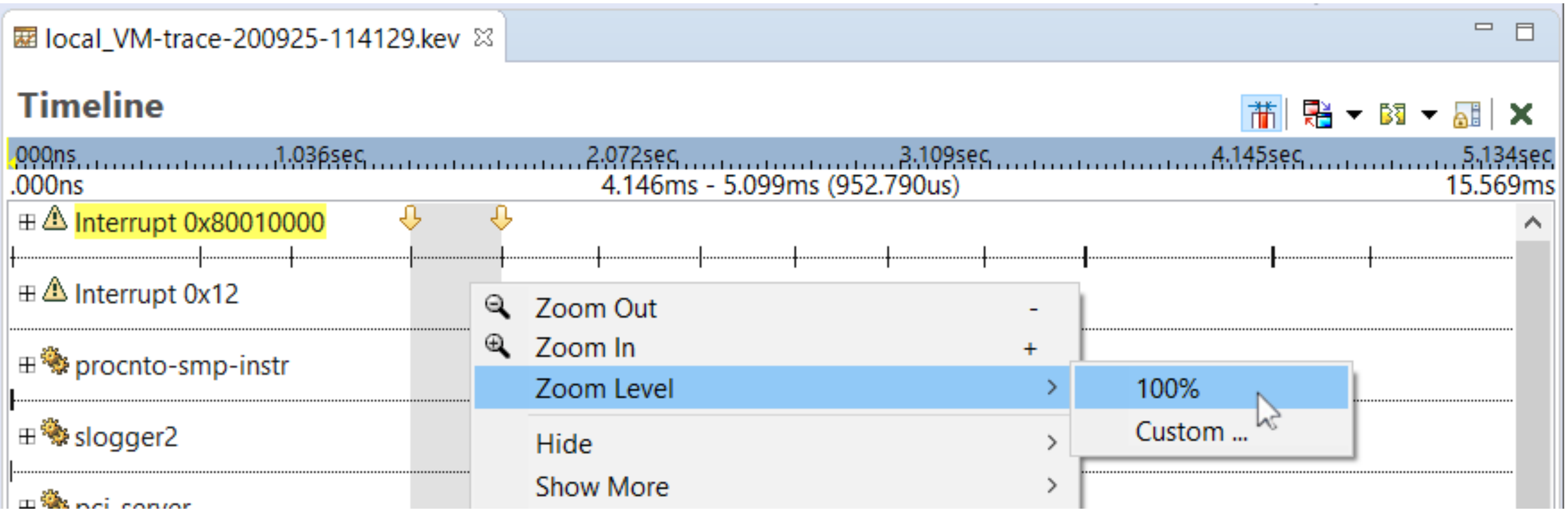


continued...

Timeline – Fixed Zoom

Zooming in/out (continued):

- for other zooming choices, right-click, choose Zoom Level:

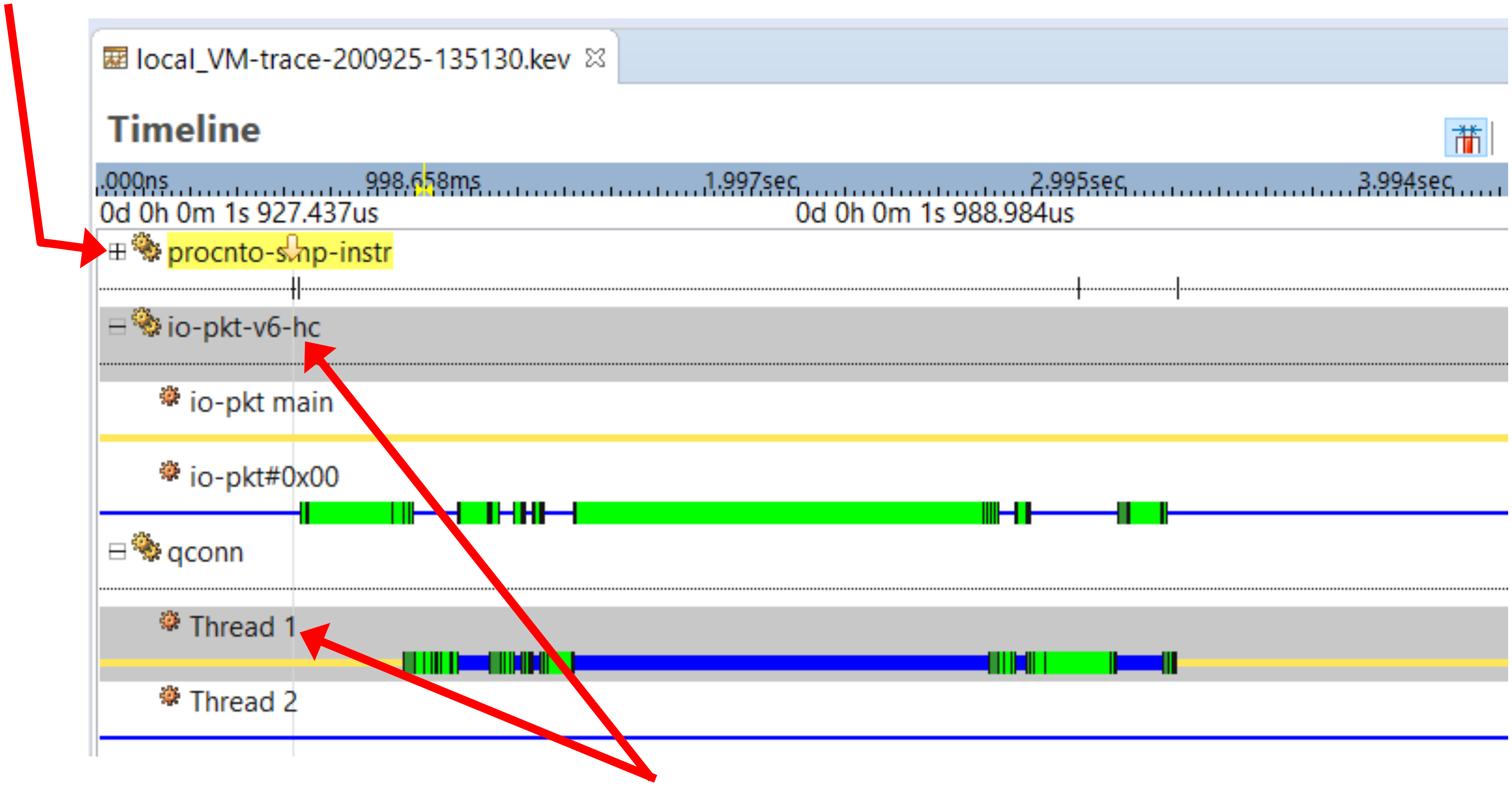


- you don’t need to select a range to use these

Timeline – Viewing Threads

Viewing thread states:

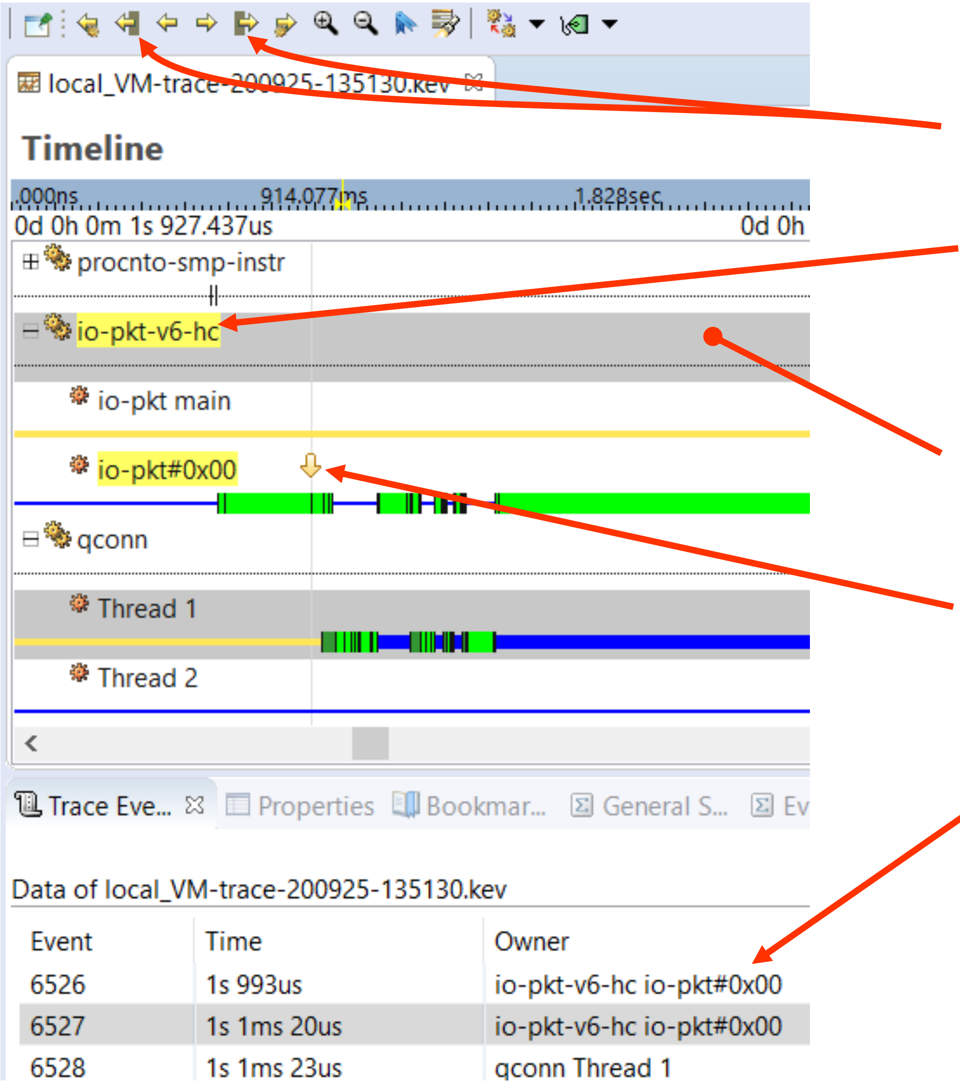
- click the ‘+’ to show threads:



- click a process or thread to select it (grey highlight) for performing a specific operation on it (`Shift` and/or `Ctrl` for multiple selections), deselect by `Ctrl`-clicking on one

Timeline – Getting Around

Getting around:



Data of local_VM-trace-200925-135130.kev

Event	Time	Owner
6526	1s 993us	io-pkt-v6-hc io-pkt#0x00
6527	1s 1ms 20us	io-pkt-v6-hc io-pkt#0x00
6528	1s 1ms 23us	qconn Thread 1

click on these buttons to move between events in the selected owner(s)

click in here and use ctrl-left and ctrl-right arrow keys to move between events

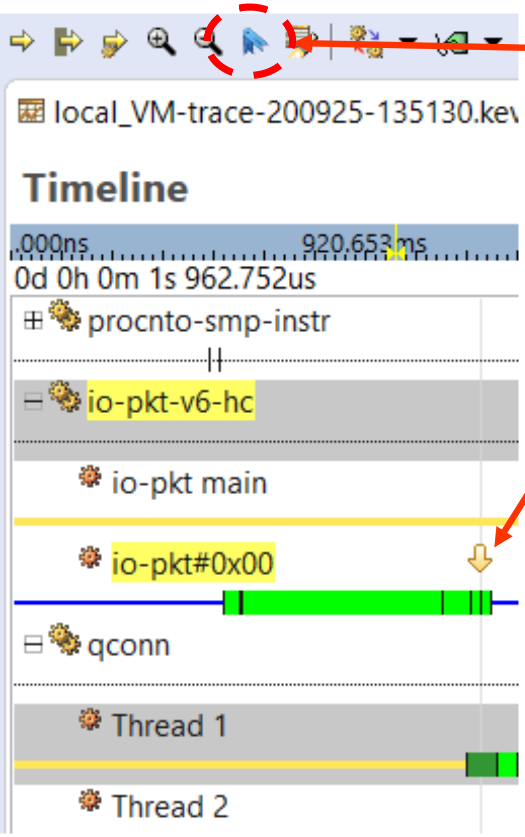
arrow points to currently selected event

walk through the events in the Trace Event Log view using the up & down arrow keys

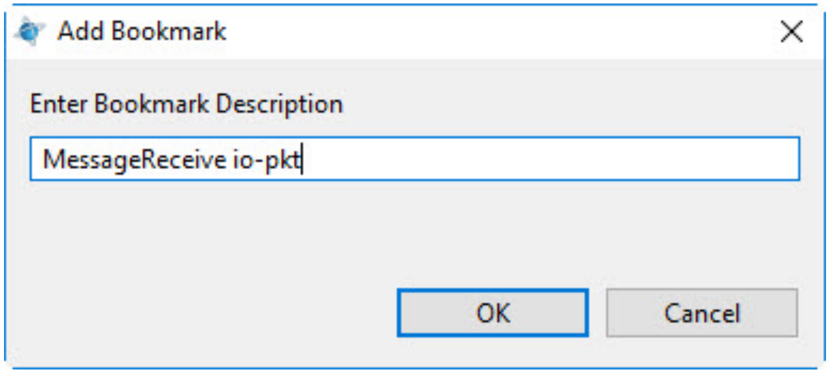
Bookmarks

It is easy to lose your place in the log:

– bookmarks can help you find your way back



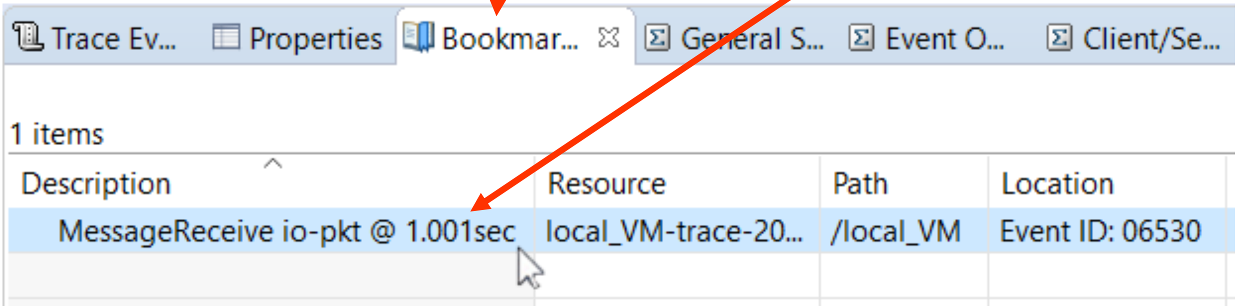
Bookmark a selected event:



Enter Bookmark Description

MessageReceive io-pkt

use the Bookmarks view to get back to it

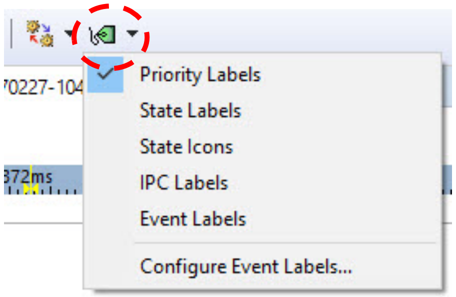


1 items

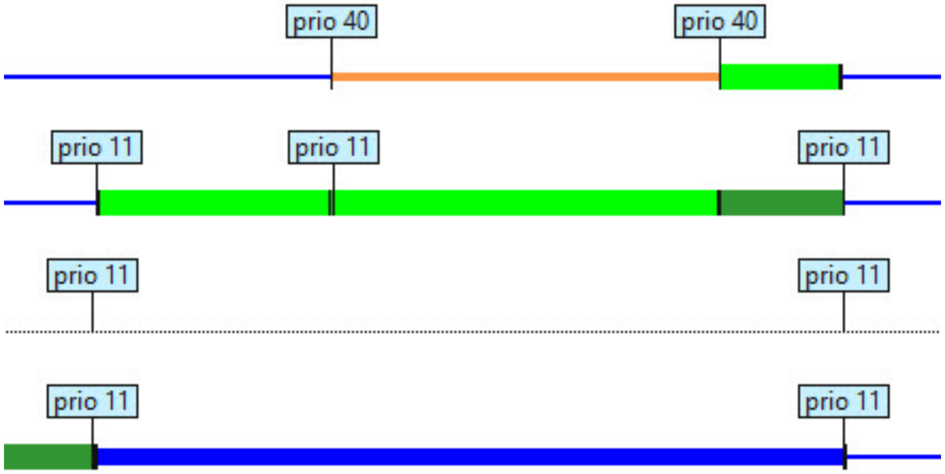
Description	Resource	Path	Location
MessageReceive io-pkt @ 1.001sec	local_VM-trace-20...	/local_VM	Event ID: 06530

Timeline - Labels

The Timeline provides a few ways of labeling things to make life easier:



- priority labels mark running threads with priority values

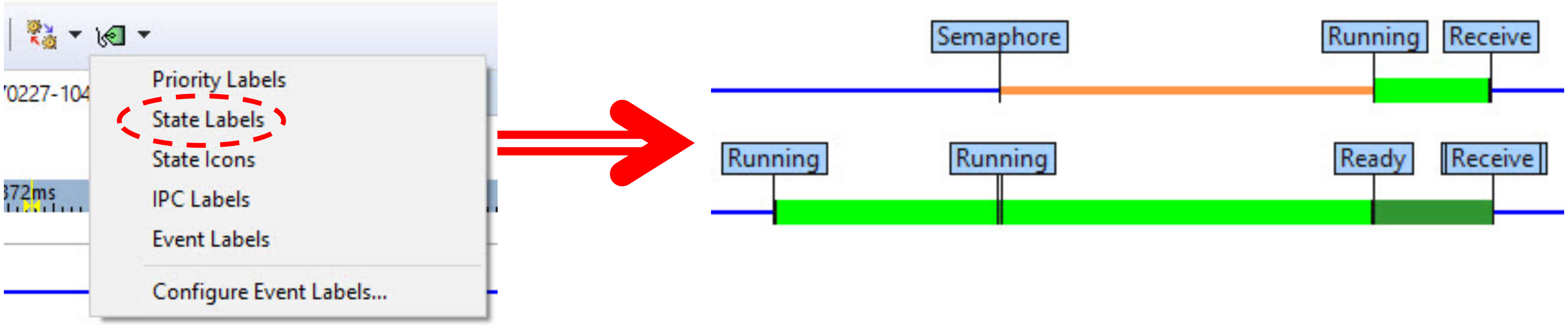


  - thread running events must be wide to include this data

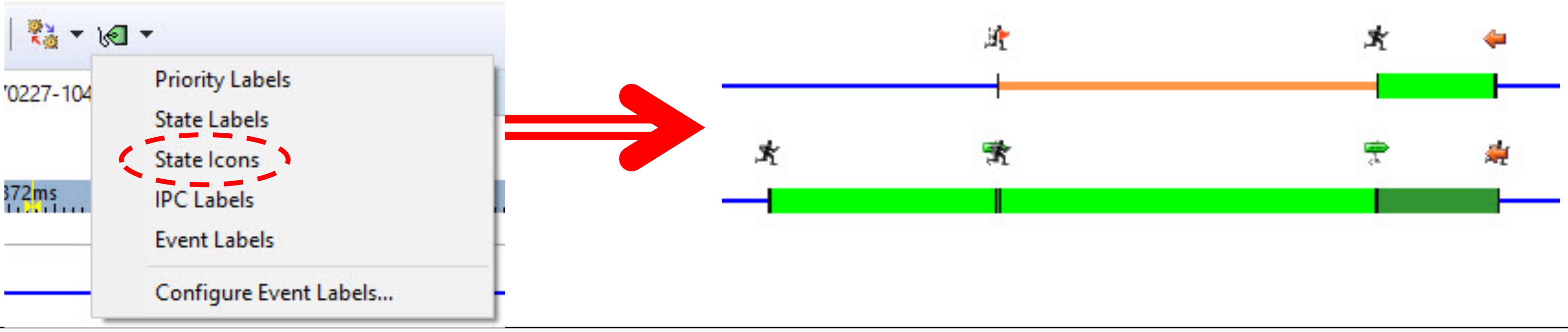
Timeline – State Labels

Colours indicate thread states:

- enabling ‘state labels’ means you don’t have to remember the colours

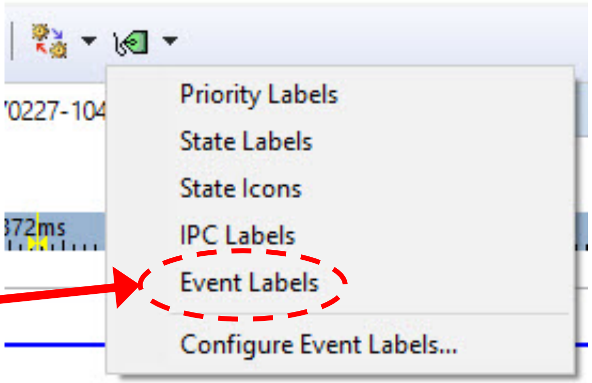


- state icons use less space, but you have to recognise the icons



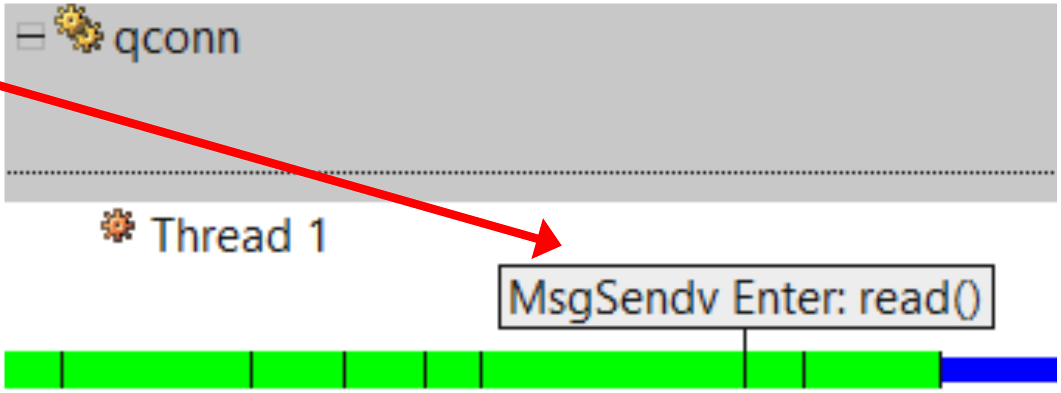
Timeline – Event Labels

Event labels:



– enabling causes some common message pass events to be labeled, e.g.

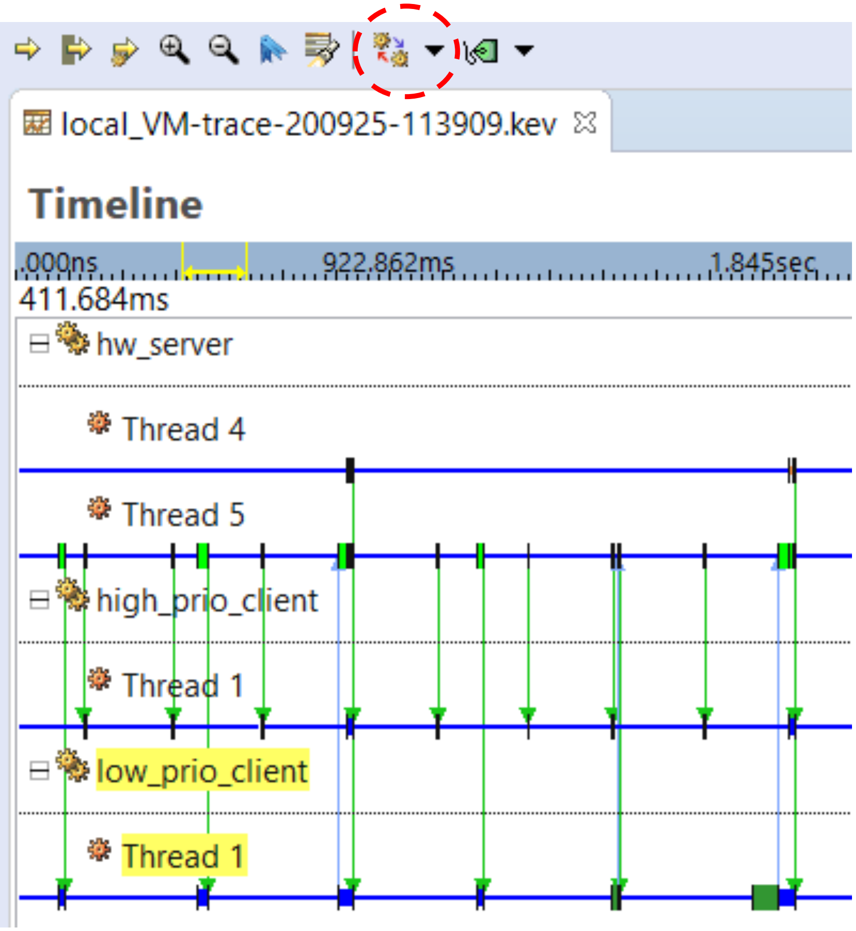
- *read()*
- *write()*
- *open()*
- *close()*



Timeline – IPC Lines

To see what QNX native messaging is occurring:

- with ‘IPC lines’ turned on, vertical lines are drawn between owners to represent IPC



local_VM-trace-200925-113909.kev

Timeline

- orange shows pulses
- blue shows sends
- green shows replies
- red shows error replies

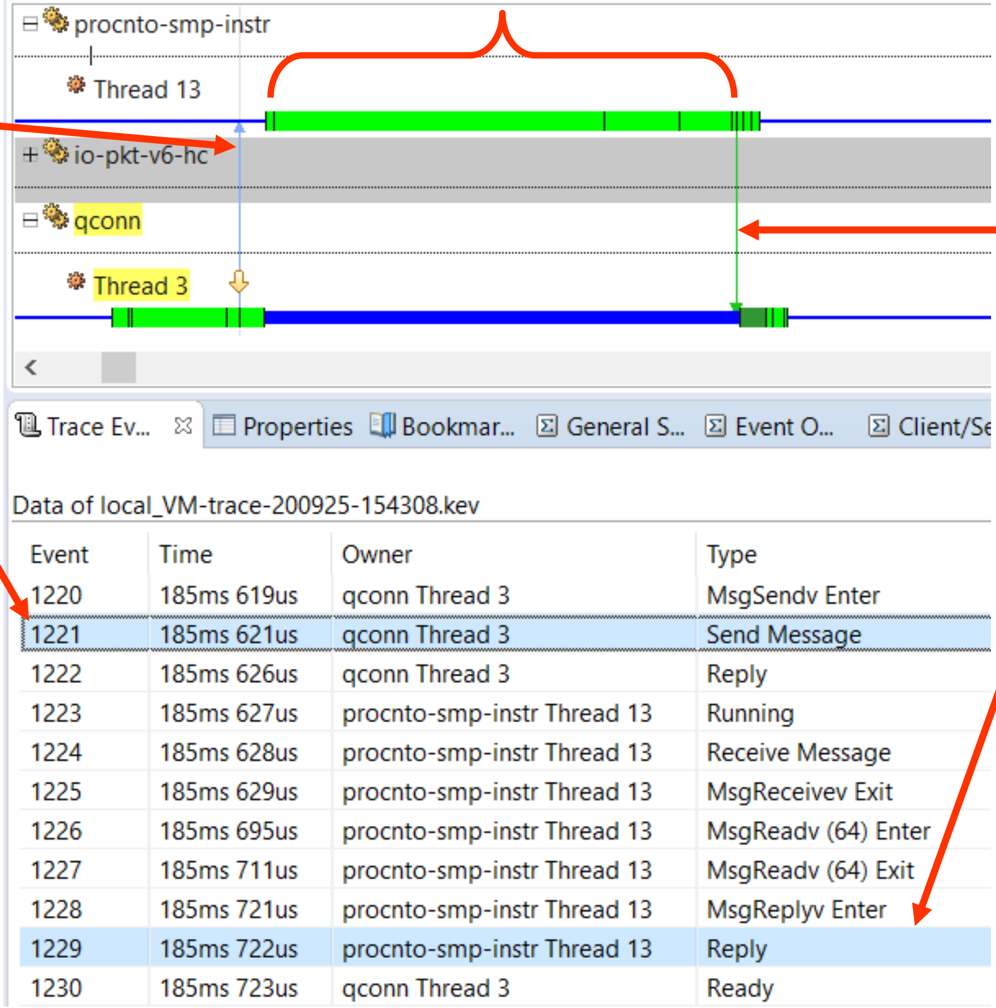
Timeline – IPC Tracing

IPC tracing example:

1 qconn sends a message to procnto-smp-instr

2 procnto-smp-instr processes msg

3 reply



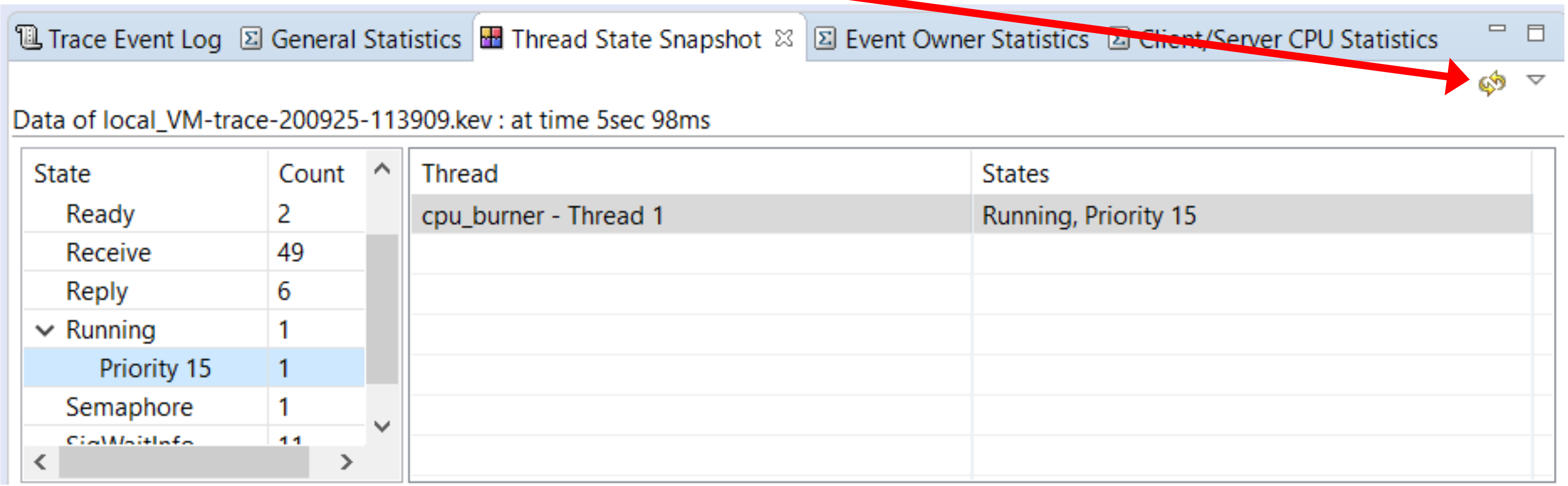
Data of local_VM-trace-200925-154308.kev

Event	Time	Owner	Type
1220	185ms 619us	qconn Thread 3	MsgSendv Enter
1221	185ms 621us	qconn Thread 3	Send Message
1222	185ms 626us	qconn Thread 3	Reply
1223	185ms 627us	procnto-smp-instr Thread 13	Running
1224	185ms 628us	procnto-smp-instr Thread 13	Receive Message
1225	185ms 629us	procnto-smp-instr Thread 13	MsgReceivev Exit
1226	185ms 695us	procnto-smp-instr Thread 13	MsgReadv (64) Enter
1227	185ms 711us	procnto-smp-instr Thread 13	MsgReadv (64) Exit
1228	185ms 721us	procnto-smp-instr Thread 13	MsgReplyv Enter
1229	185ms 722us	procnto-smp-instr Thread 13	Reply
1230	185ms 723us	qconn Thread 3	Ready

Timeline – Thread State Snapshot

What are the other threads doing?

- the Thread State Snapshot answers this question
- click somewhere in the timeline, then click the refresh button to get the list:



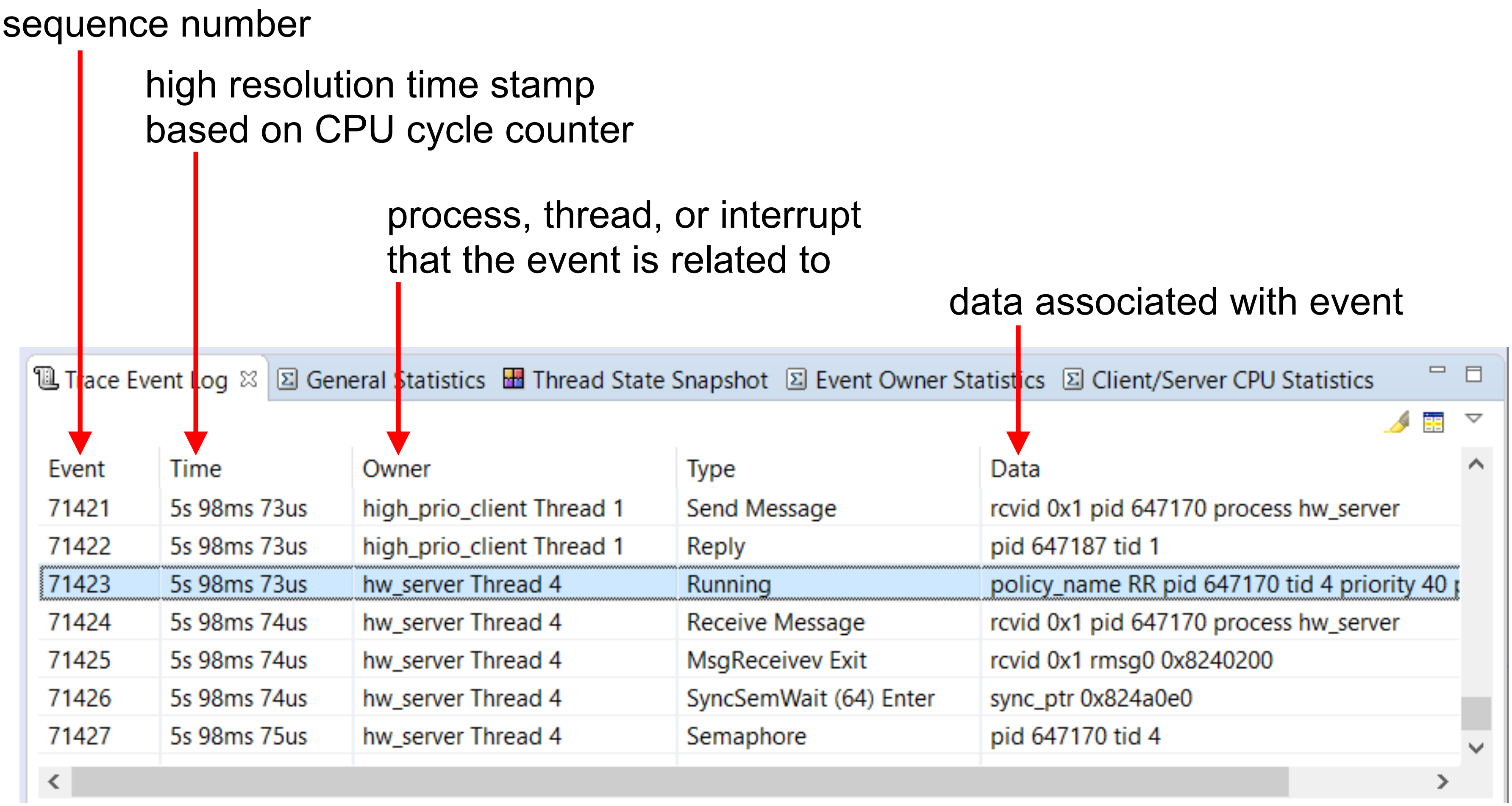
Data of local_VM-trace-200925-113909.kev : at time 5sec 98ms

State	Count
Ready	2
Receive	49
Reply	6
Running	1
Priority 15	1
Semaphore	1
SigWaitInfo	11

Thread	States
cpu_burner - Thread 1	Running, Priority 15

Timeline - Trace Event Log

If you select an event in the Timeline, details can be found in the Trace Event Log:



sequence number

high resolution time stamp based on CPU cycle counter

process, thread, or interrupt that the event is related to

data associated with event

Event	Time	Owner	Type	Data
71421	5s 98ms 73us	high_prio_client Thread 1	Send Message	rcvid 0x1 pid 647170 process hw_server
71422	5s 98ms 73us	high_prio_client Thread 1	Reply	pid 647187 tid 1
71423	5s 98ms 73us	hw_server Thread 4	Running	policy_name RR pid 647170 tid 4 priority 40 p
71424	5s 98ms 74us	hw_server Thread 4	Receive Message	rcvid 0x1 pid 647170 process hw_server
71425	5s 98ms 74us	hw_server Thread 4	MsgReceivev Exit	rcvid 0x1 rmsg0 0x8240200
71426	5s 98ms 74us	hw_server Thread 4	SyncSemWait (64) Enter	sync_ptr 0x824a0e0
71427	5s 98ms 75us	hw_server Thread 4	Semaphore	pid 647170 tid 4

Timeline - Finding vs. Searching

Two ways to locate specific events in a log:

– Find

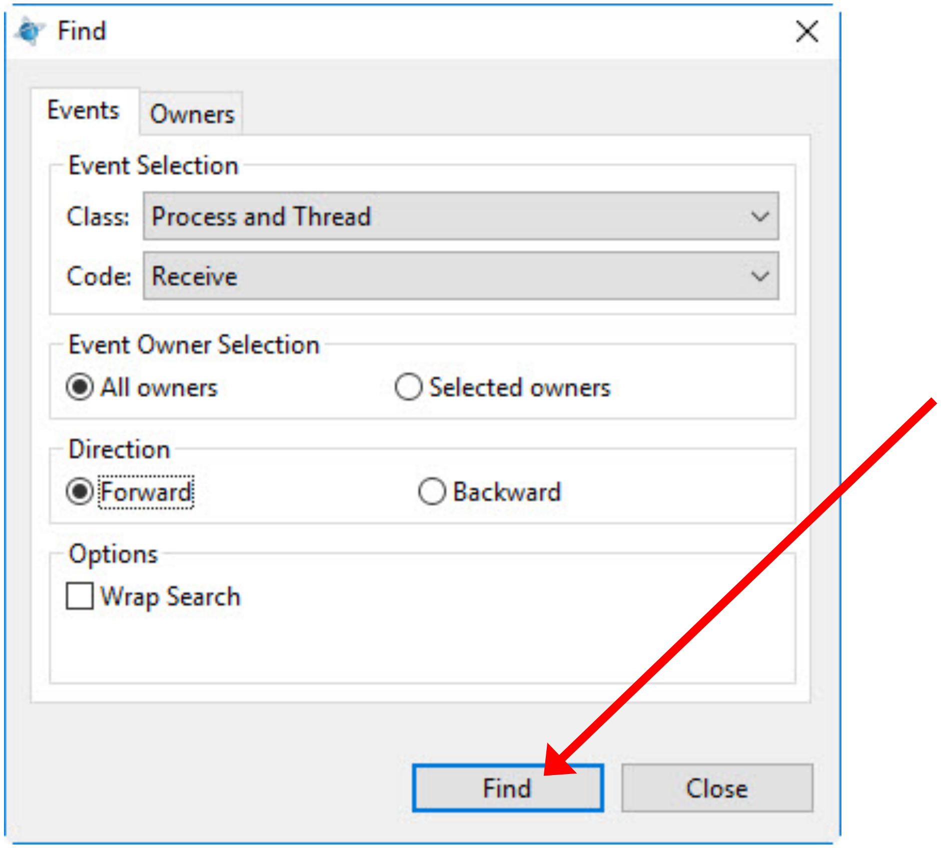
- quicker to use, simpler
- good for most times you need to locate an event
- CTRL-F, or, Edit Menu→Find
- moves selection to first event that is found
- works same way as typical ‘Find’ in office apps

– Search

- CTRL-H, or, Search Menu
- more features
- requires that you set up ‘search conditions’ that it will look for
- Search view provides you with list of all matches

Using ‘Find’:

– setup information about event you want to find



Event Selection

Class: Process and Thread

Code: Receive

Event Owner Selection

● All owners ○ Selected owners

Direction

● Forward ○ Backward

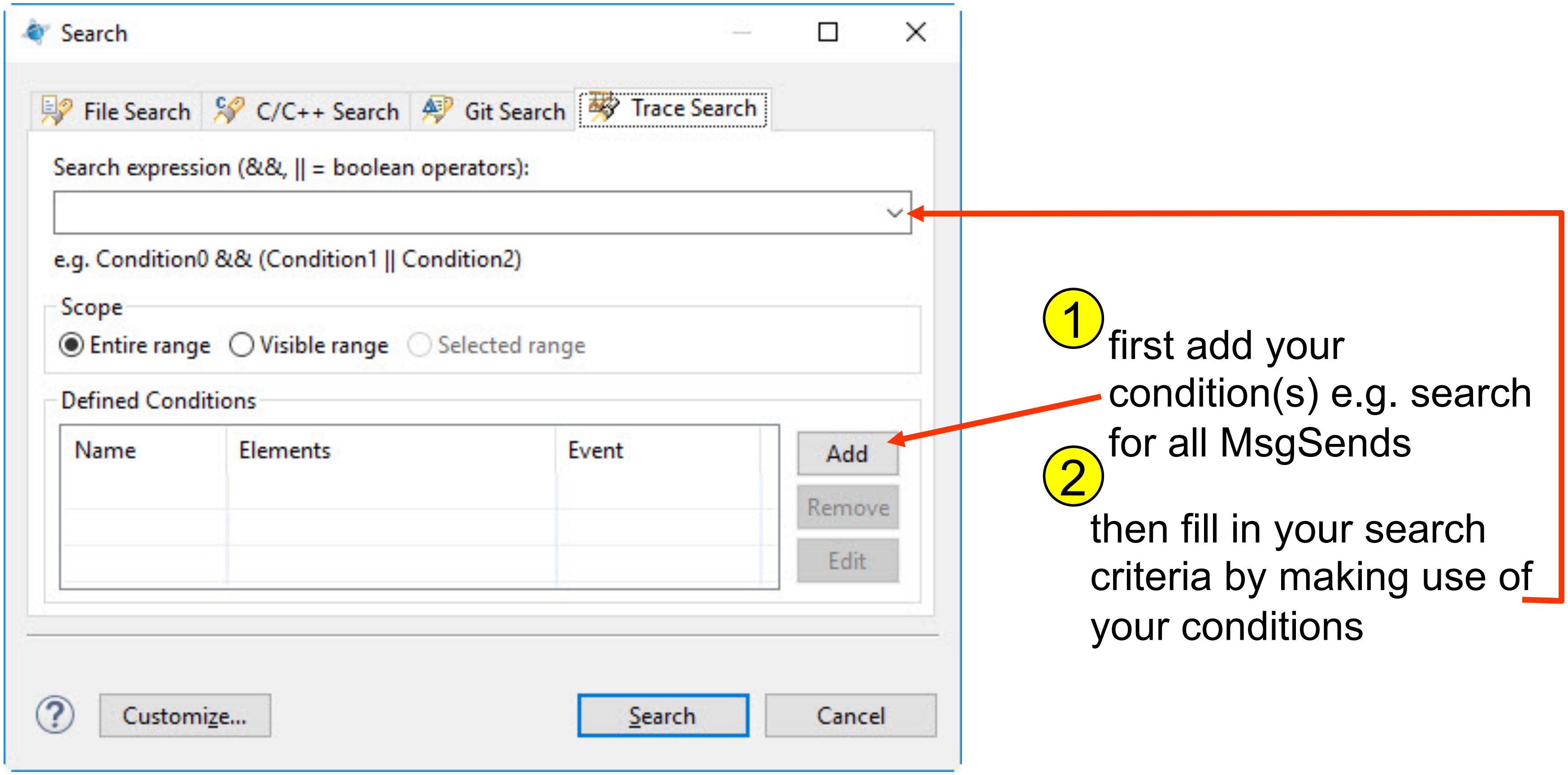
Options

□ Wrap Search

clicking find takes you to next event that is found

Using ‘Search’:

– use the Search menu or press Ctrl-H



Search expression (&&, || = boolean operators):

e.g. Condition0 && (Condition1 || Condition2)

Scope

● Entire range ○ Visible range ○ Selected range

Defined Conditions

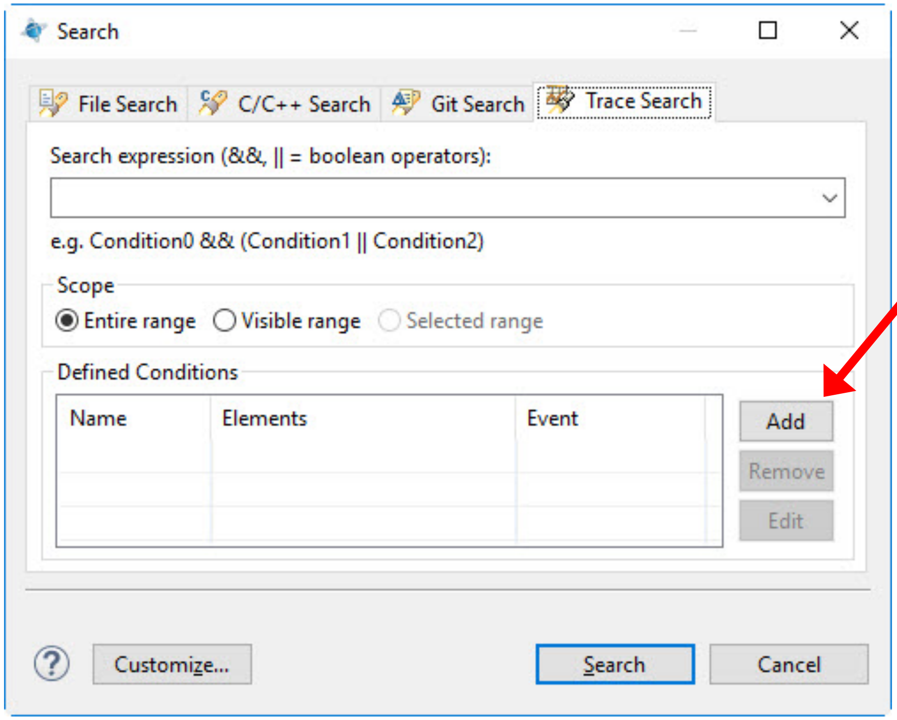
Name	Elements	Event

1 first add your condition(s) e.g. search for all MsgSends

2 then fill in your search criteria by making use of your conditions

Searching example:

- say we have a process called “**`high_prio_client`**”, which isn’t getting as much CPU time as we expect
- let’s find circumstances where **`high_prio_client`** wanted to run (i.e. was ready), and then we can look at what else was running instead



Search expression (&&, || = boolean operators):

e.g. Condition0 && (Condition1 || Condition2)

Scope

● Entire range ○ Visible range ○ Selected range

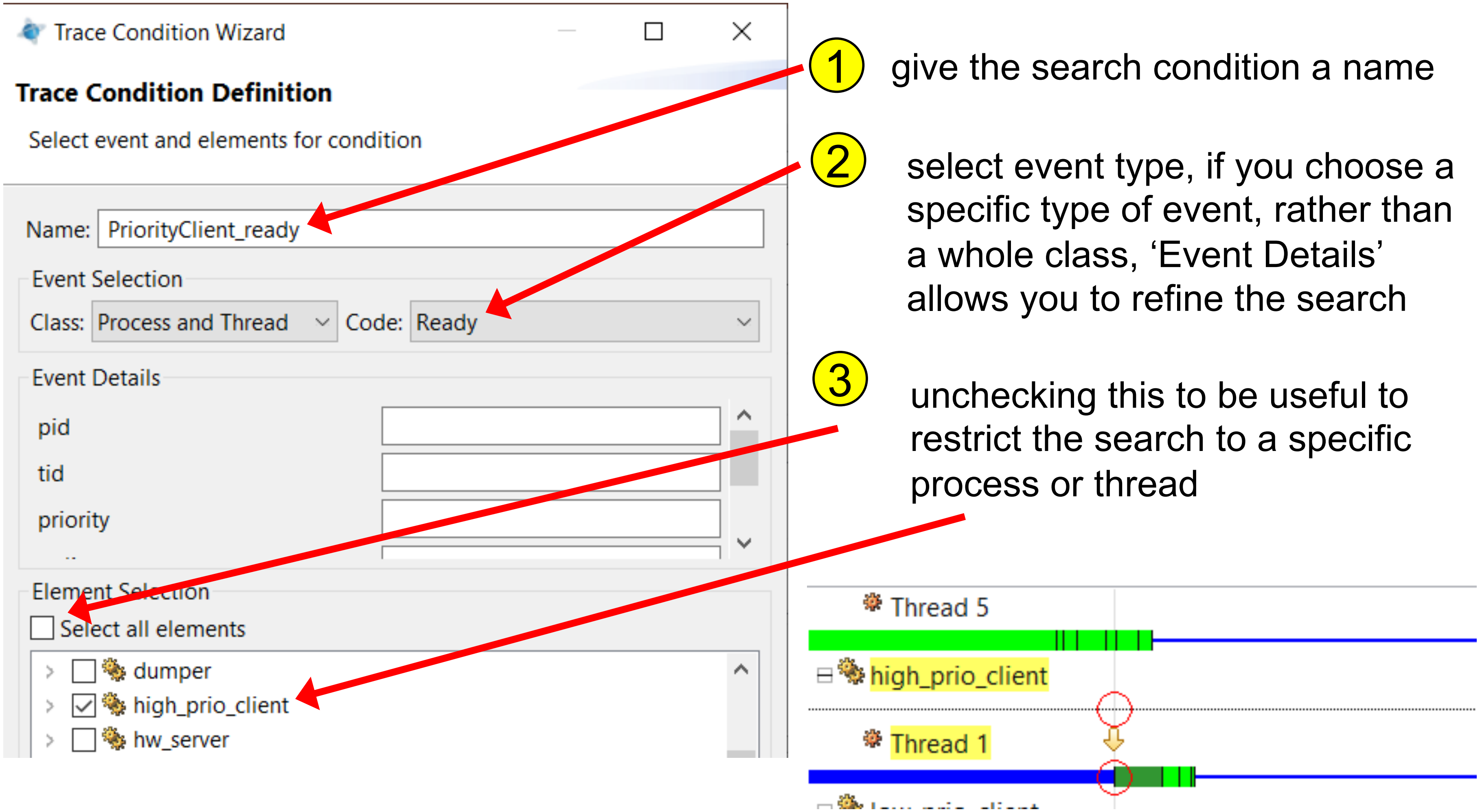
Defined Conditions

Name	Elements	Event

1. click ‘Add’ to create a search condition
2. let’s look at condition definition on next page…

continued...

Searching example (continued):



Trace Condition Wizard

Trace Condition Definition

Select event and elements for condition

Name: PriorityClient_ready

Event Selection

Class: Process and Thread Code: Ready

Event Details

pid

tid

priority

Element Selection

□ Select all elements

□ dumper

☑ high_prio_client

□ hw_server

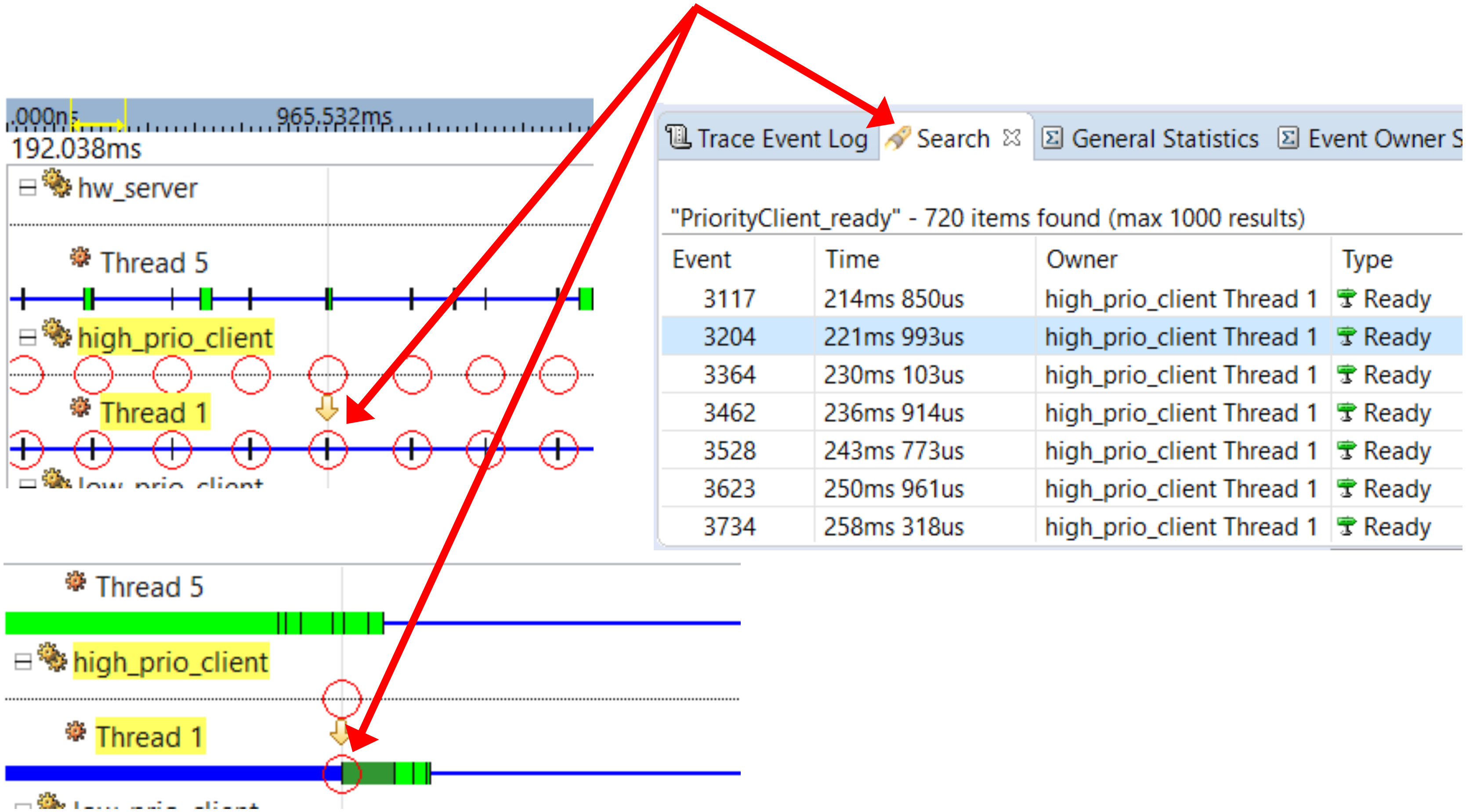
1 give the search condition a name

2 select event type, if you choose a specific type of event, rather than a whole class, ‘Event Details’ allows you to refine the search

3 unchecking this to be useful to restrict the search to a specific process or thread

Results from a search:

- events are shown as:



"PriorityClient_ready" - 720 items found (max 1000 results)

Event	Time	Owner	Type
3117	214ms 850us	high_prio_client Thread 1	Ready
3204	221ms 993us	high_prio_client Thread 1	Ready
3364	230ms 103us	high_prio_client Thread 1	Ready
3462	236ms 914us	high_prio_client Thread 1	Ready
3528	243ms 773us	high_prio_client Thread 1	Ready
3623	250ms 961us	high_prio_client Thread 1	Ready
3734	258ms 318us	high_prio_client Thread 1	Ready

Topics:

Overview

Creating a Log

Log Summary

A Quick Tour

Filtering Events and Event Owners

Navigating through a log with the Timeline Pane

→ – Exercise

Statistics

CPU Activity Pane and CPU Usage Pane

Tying The Trace to Your Code

Multi-core Related Features

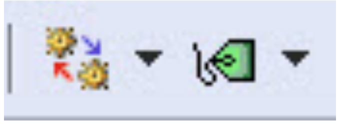
Adaptive Partitioning: Partition Summary

Conclusion

EXERCISE – Navigating a Log

Try out the log navigation techniques:

1. capture a new log, or use an existing one
2. switch panes e.g. CPU Activity, CPU Usage
3. zoom in/out, scroll
4. open some processes to see threads and thread states
5. match events in Timeline with Event Log, try and become familiar with format of events in log
6. bookmark an event
7. use Find and Search to locate an event where qconn goes into the running state.
8. turn on labels and look at the resulting information



System Profiling

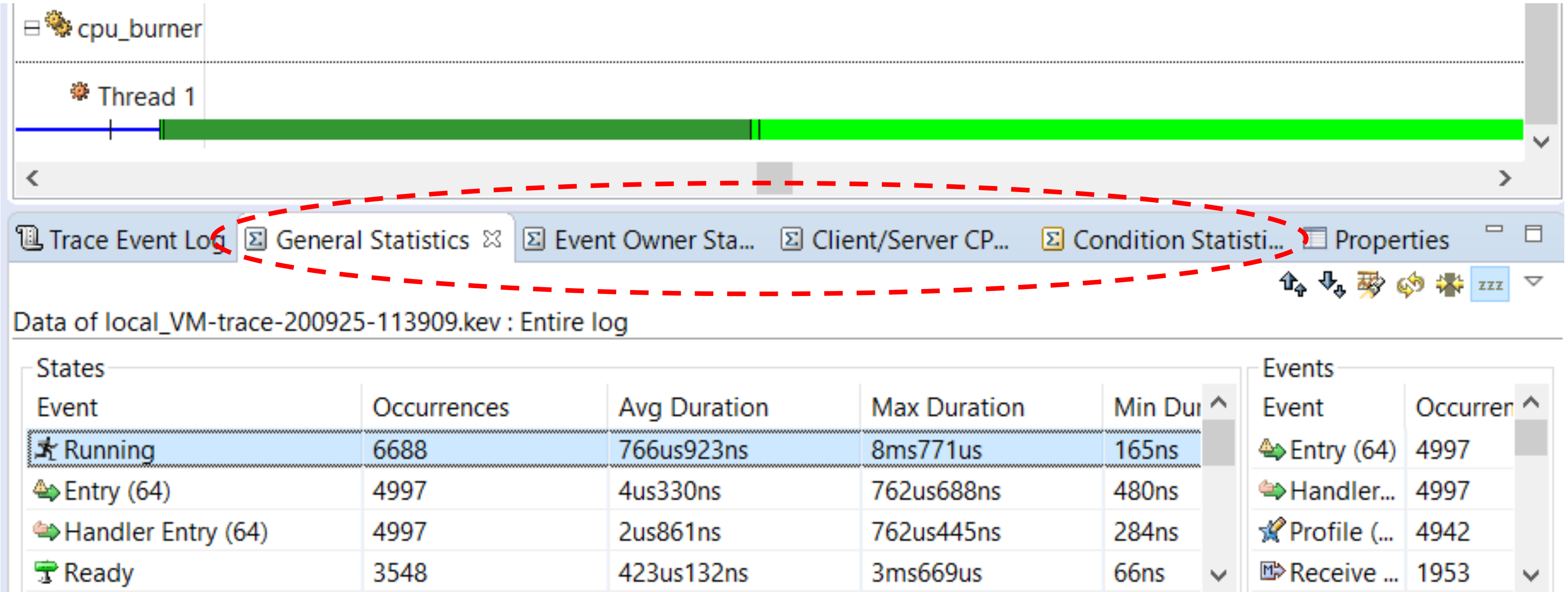
Topics:

- Overview
- Creating a Log
- Log Summary
- A Quick Tour
- Filtering Events and Event Owners
- Navigating through a log with the Timeline Pane
- → Statistics
- CPU Activity Pane and CPU Usage Pane
- Tying The Trace to Your Code
- Multi-core Related Features
- Adaptive Partitioning: Partition Summary
- Conclusion

Statistics Views

Statistics views:

- System Profiler can provide various statistical info.
- useful for:
  - getting an overall feel for what went on during a log
  - seeing some kinds of problem situations
    - e.g. a thread spend a really long time waiting for a mutex to be



Data of local_VM-trace-200925-113909.kev : Entire log

States

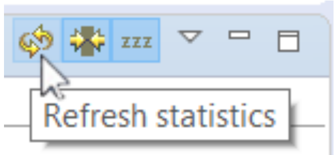
Event	Occurrences	Avg Duration	Max Duration	Min Dur
Running	6688	766us923ns	8ms771us	165ns
Entry (64)	4997	4us330ns	762us688ns	480ns
Handler Entry (64)	4997	2us861ns	762us445ns	284ns
Ready	3548	423us132ns	3ms669us	66ns

Events

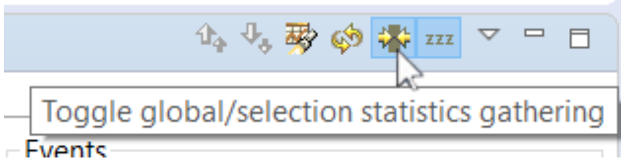
Event	Occurren
Entry (64)	4997
Handler...	4997
Profile (...	4942
Receive ...	1953

Statistics Views

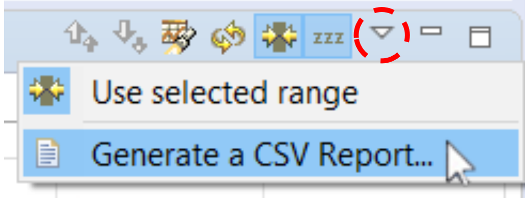
The Statistics Views all share some common behaviors:

- they don’t display any data until you click the refresh statistics button: 

- they can all operate on either the entire log, or a selected area, controlled by a toggle:



- all can generate a CSV report of their data:



Statistics Views

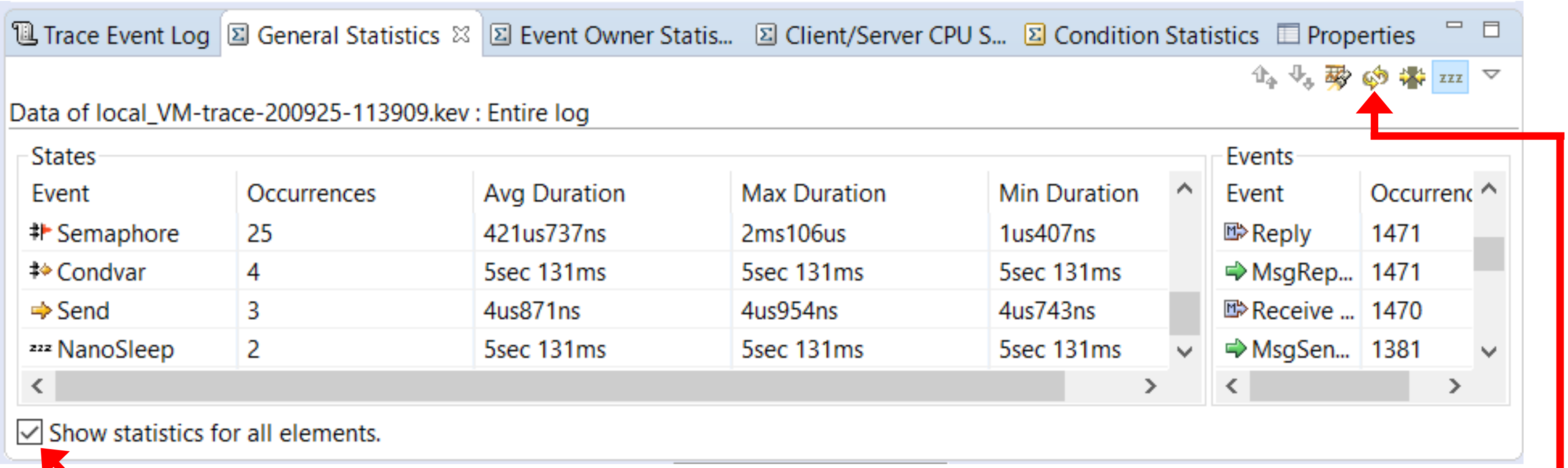
The Statistics Views are:

- General Statistics
  - give a bunch of general statistics for all owners, or selected owners
- Event Owner Statistics
  - collect statistics for a configurable list of events on a per owner basis
- Condition Statistics
  - do a time range count of configured conditions
- Client/Server CPU Statistics
  - accumulate server time back to a client when appropriate

Statistics

General Statistics: 2 ways to use it:

- show stats for all processes/threads
- show stats for only selected process/thread (or multiple)



Data of local_VM-trace-200925-113909.kev : Entire log

States

Event	Occurrences	Avg Duration	Max Duration	Min Duration
Semaphore	25	421us737ns	2ms106us	1us407ns
Condvar	4	5sec 131ms	5sec 131ms	5sec 131ms
Send	3	4us871ns	4us954ns	4us743ns
NanoSleep	2	5sec 131ms	5sec 131ms	5sec 131ms

Events

Event	Occurrenc
Reply	1471
MsgRep...	1471
Receive ...	1470
MsgSen...	1381

☑ Show statistics for all elements.

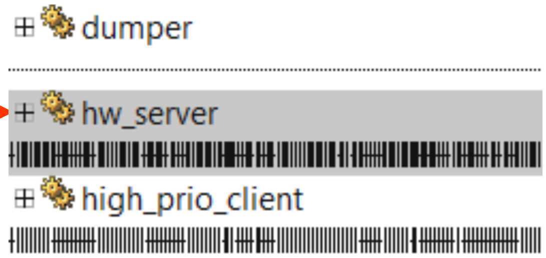
1. choose to show stats for all processes, or only a selection
   if you uncheck this, make sure to have one or more processes or threads selected in the timeline
2. click Refresh

General Statistics

Let’s use the General Statistics view:

– we’ll try to find the longest time a thread in *hw_server* is semaphore blocked for, and where this occurred

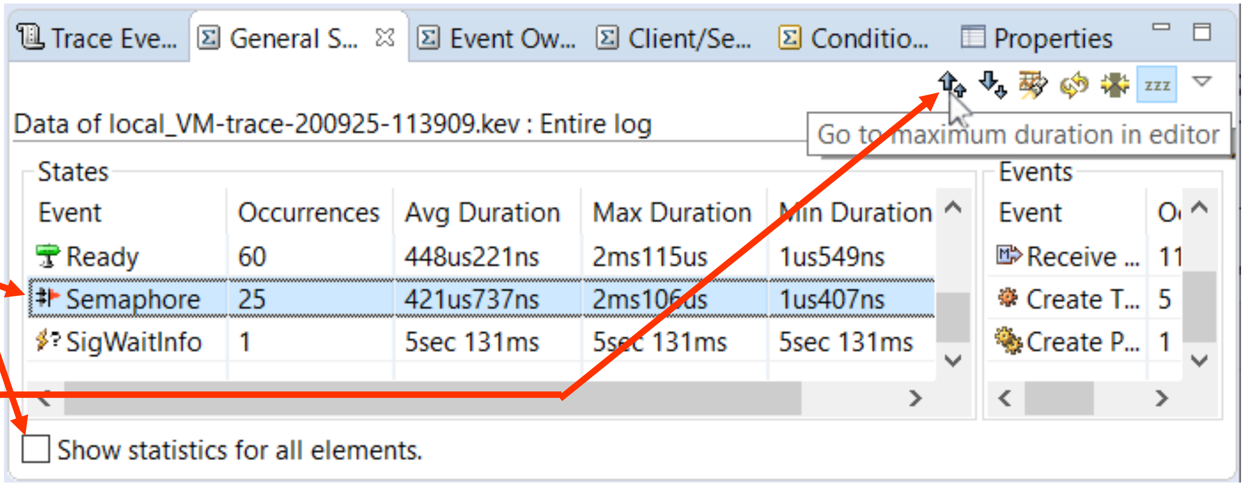
we’ll start by selecting *hw_server* in the timeline



unclick all elements and refresh if needed

select Semaphore

and click on “go to maximum duration”



Data of local_VM-trace-200925-113909.kev : Entire log

States

Event	Occurrences	Avg Duration	Max Duration	Min Duration
Ready	60	448us221ns	2ms115us	1us549ns
Semaphore	25	421us737ns	2ms106us	1us407ns
SigWaitInfo	1	5sec 131ms	5sec 131ms	5sec 131ms

Events

Event	O
Receive ...	11
Create T...	5
Create P...	1

□ Show statistics for all elements.

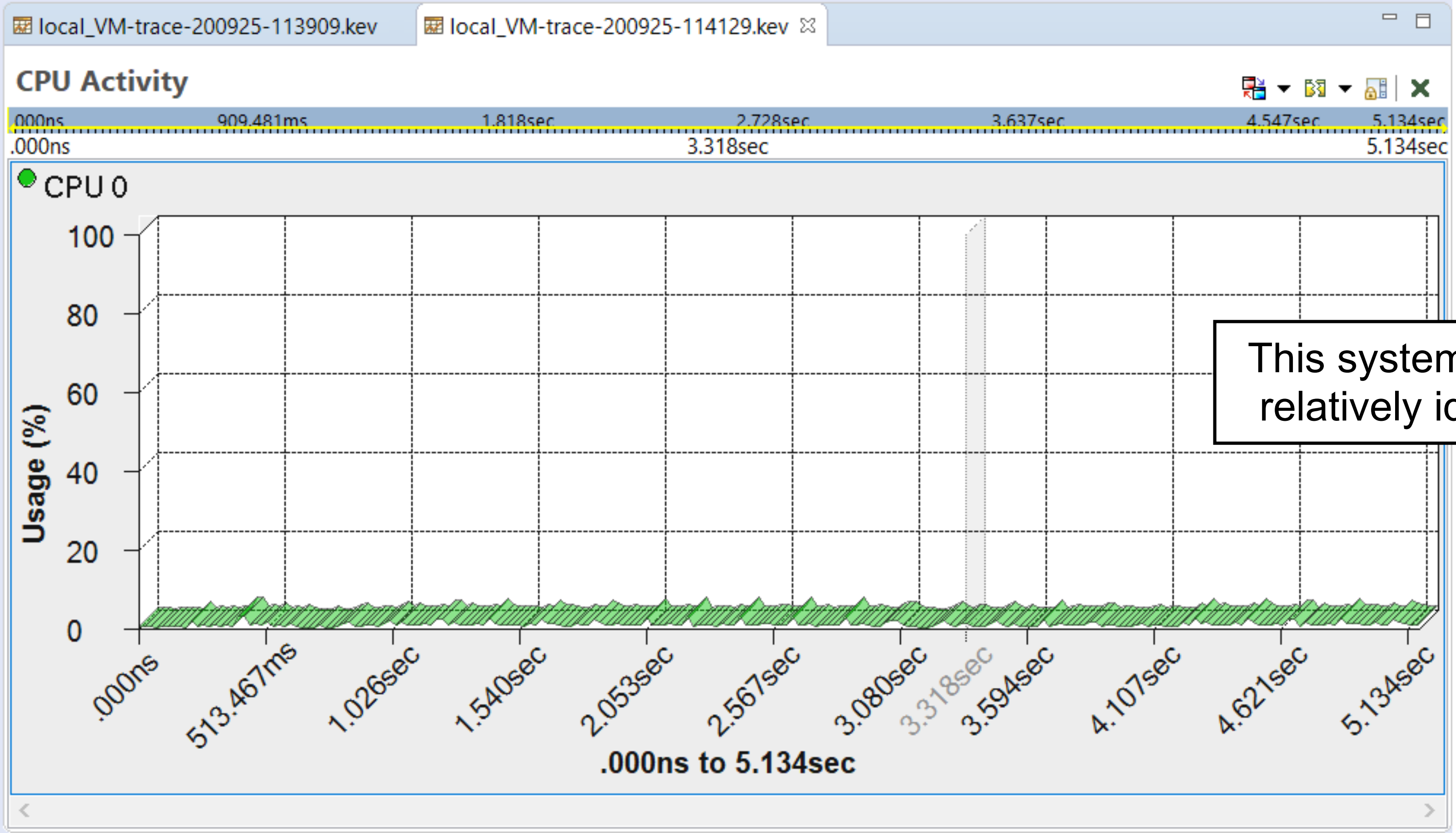
Topics:

- Overview
- Creating a Log
- Log Summary
- A Quick Tour
- Filtering Events and Event Owners
- Navigating through a log with the Timeline Pane
- Statistics
- → CPU Activity Pane and CPU Usage Pane
  - – Exercise
- Tying The Trace to Your Code
- Multi-core Related Features
- Adaptive Partitioning: Partition Summary
- Conclusion

CPU Activity

CPU Activity:

– tracks total CPU usage over time

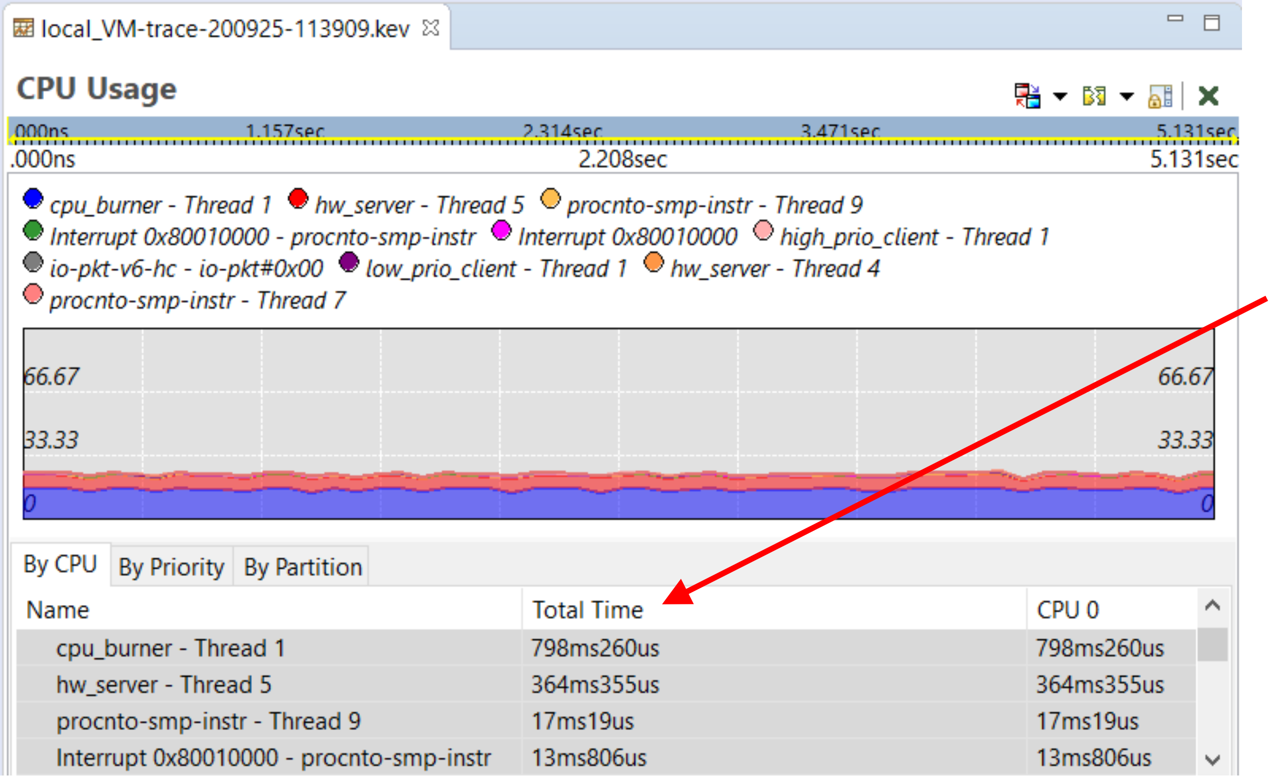


This system is relatively idle

CPU Usage

CPU Usage:

- shows the CPU consumption for the time period selected in the Timeline, showing the top ten CPU users by default
- can be shown in %, time, or both (configured in Window menu→Preferences→System Profiler→CPU Usage


local_VM-trace-200925-113909.kev
CPU Usage
.000ns
1.157sec
2.314sec
3.471sec
5.131sec
.000ns
2.208sec
5.131sec
● cpu_burner - Thread 1
● hw_server - Thread 5
● procnto-smp-instr - Thread 9
● Interrupt 0x80010000 - procnto-smp-instr
● Interrupt 0x80010000
● high_prio_client - Thread 1
● io-pkt-v6-hc - io-pkt#0x00
● low_prio_client - Thread 1
● hw_server - Thread 4
● procnto-smp-instr - Thread 7
66.67
33.33
0
66.67
33.33
0
By CPU
By Priority
By Partition

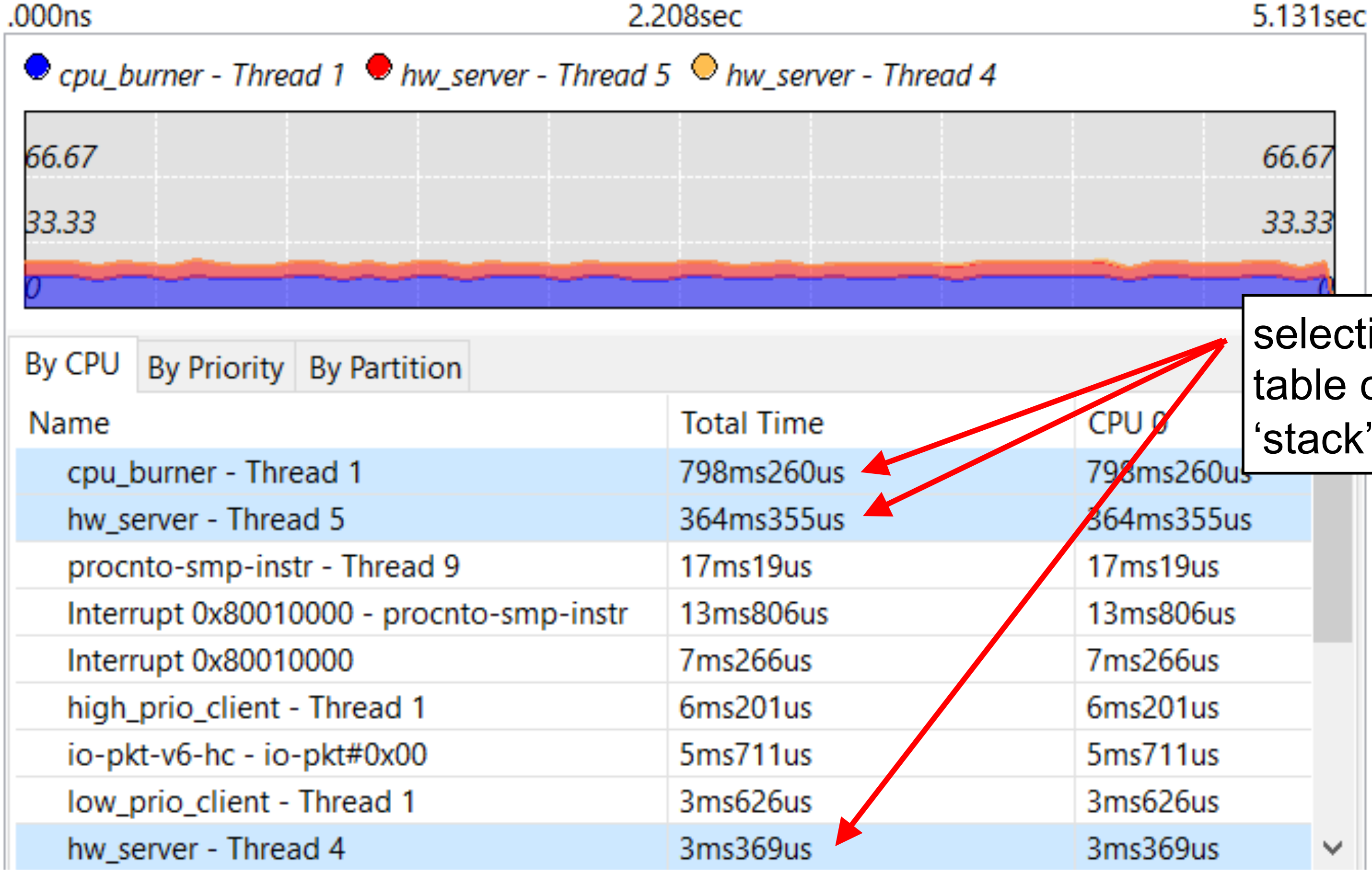
Name	Total Time	CPU 0
cpu_burner - Thread 1	798ms260us	798ms260us
hw_server - Thread 5	364ms355us	364ms355us
procnto-smp-instr - Thread 9	17ms19us	17ms19us
Interrupt 0x80010000 - procnto-smp-instr	13ms806us	13ms806us

threads are sorted in order of greatest CPU user

CPU Usage

You can select multiple threads in CPU Usage:

- hold down CTRL to select/deselect multiple

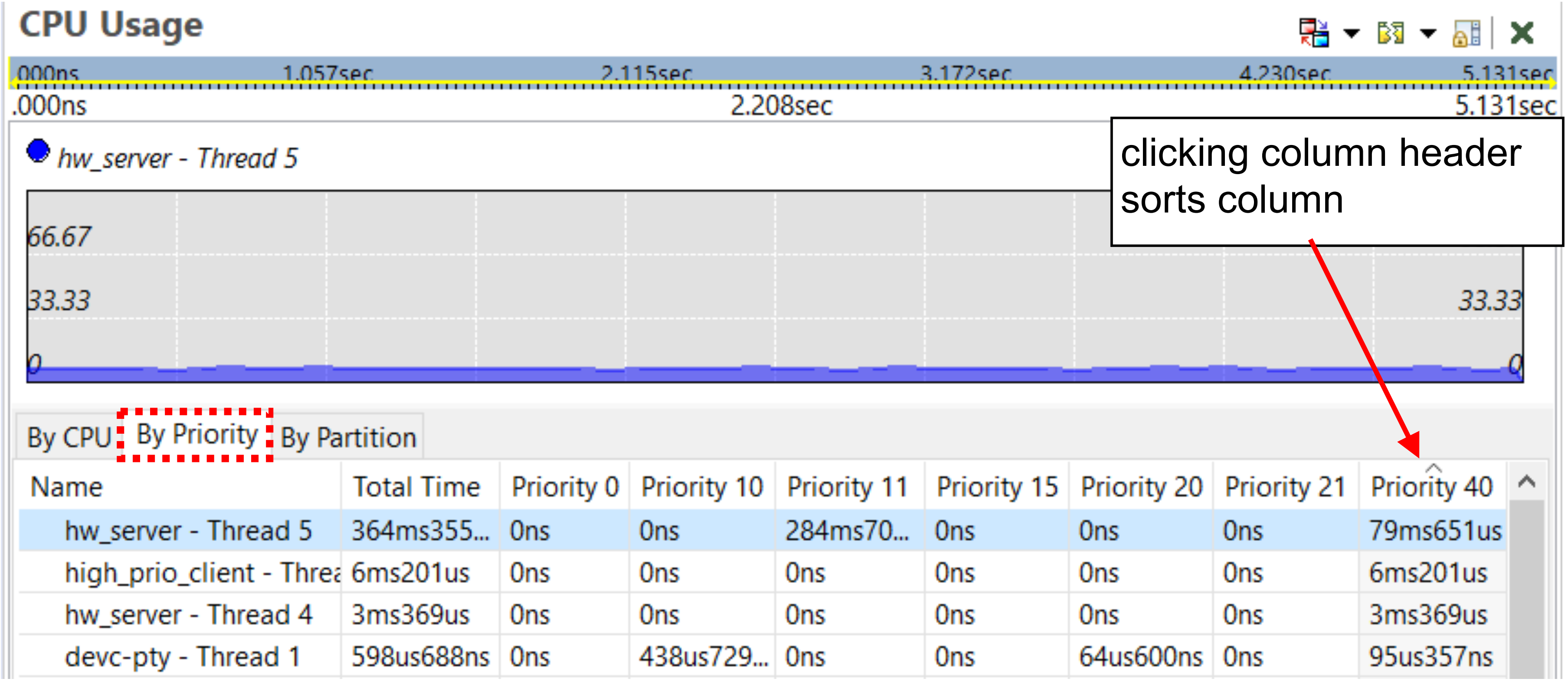


Name	Total Time	CPU 0
cpu_burner - Thread 1	798ms260us	798ms260us
hw_server - Thread 5	364ms355us	364ms355us
procnto-smp-instr - Thread 9	17ms19us	17ms19us
Interrupt 0x80010000 - procnto-smp-instr	13ms806us	13ms806us
Interrupt 0x80010000	7ms266us	7ms266us
high_prio_client - Thread 1	6ms201us	6ms201us
io-pkt-v6-hc - io-pkt#0x00	5ms711us	5ms711us
low_prio_client - Thread 1	3ms626us	3ms626us
hw_server - Thread 4	3ms369us	3ms369us

selecting threads in the table causes them to ‘stack’ visually

CPU Usage can be sorted ‘By Priority’:

– good for analyzing circumstances where threads at the same priority are competing for CPU time



clicking column header sorts column

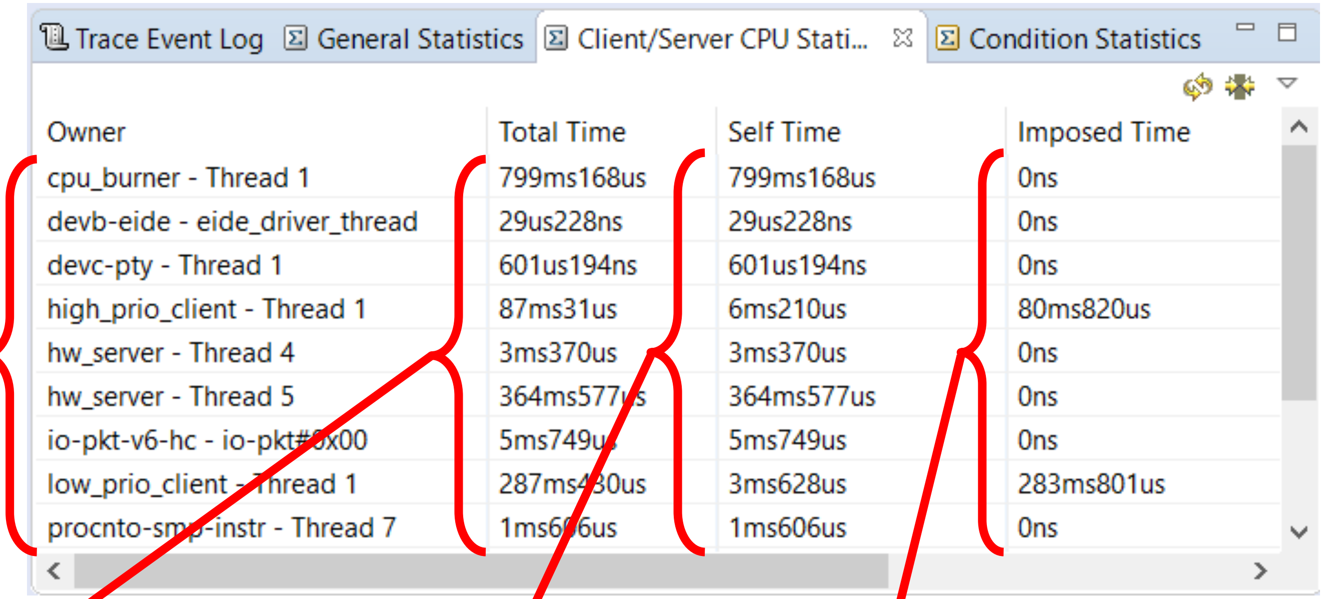
Name	Total Time	Priority 0	Priority 10	Priority 11	Priority 15	Priority 20	Priority 21	Priority 40
hw_server - Thread 5	364ms355...	0ns	0ns	284ms70...	0ns	0ns	0ns	79ms651us
high_prio_client - Threa	6ms201us	0ns	0ns	0ns	0ns	0ns	0ns	6ms201us
hw_server - Thread 4	3ms369us	0ns	0ns	0ns	0ns	0ns	0ns	3ms369us
devc-pty - Thread 1	598us688ns	0ns	438us729...	0ns	0ns	64us600ns	0ns	95us357ns

Client/Server

The CPU usage of a particular thread may not give the whole story

- Neutrino is a message passing based system
- threads will send messages to server threads to get work done on their behalf
- Client/Server CPU Statistics is a tool that can help you take server execution time into account

Client/Server CPU Statistics:



Owner	Total Time	Self Time	Imposed Time
cpu_burner - Thread 1	799ms168us	799ms168us	0ns
devb-eide - eide_driver_thread	29us228ns	29us228ns	0ns
devc-pty - Thread 1	601us194ns	601us194ns	0ns
high_prio_client - Thread 1	87ms31us	6ms210us	80ms820us
hw_server - Thread 4	3ms370us	3ms370us	0ns
hw_server - Thread 5	364ms577us	364ms577us	0ns
io-pkt-v6-hc - io-pkt#0x00	5ms749us	5ms749us	0ns
low_prio_client - Thread 1	287ms430us	3ms628us	283ms801us
procnto-smp-instr - Thread 7	1ms606us	1ms606us	0ns

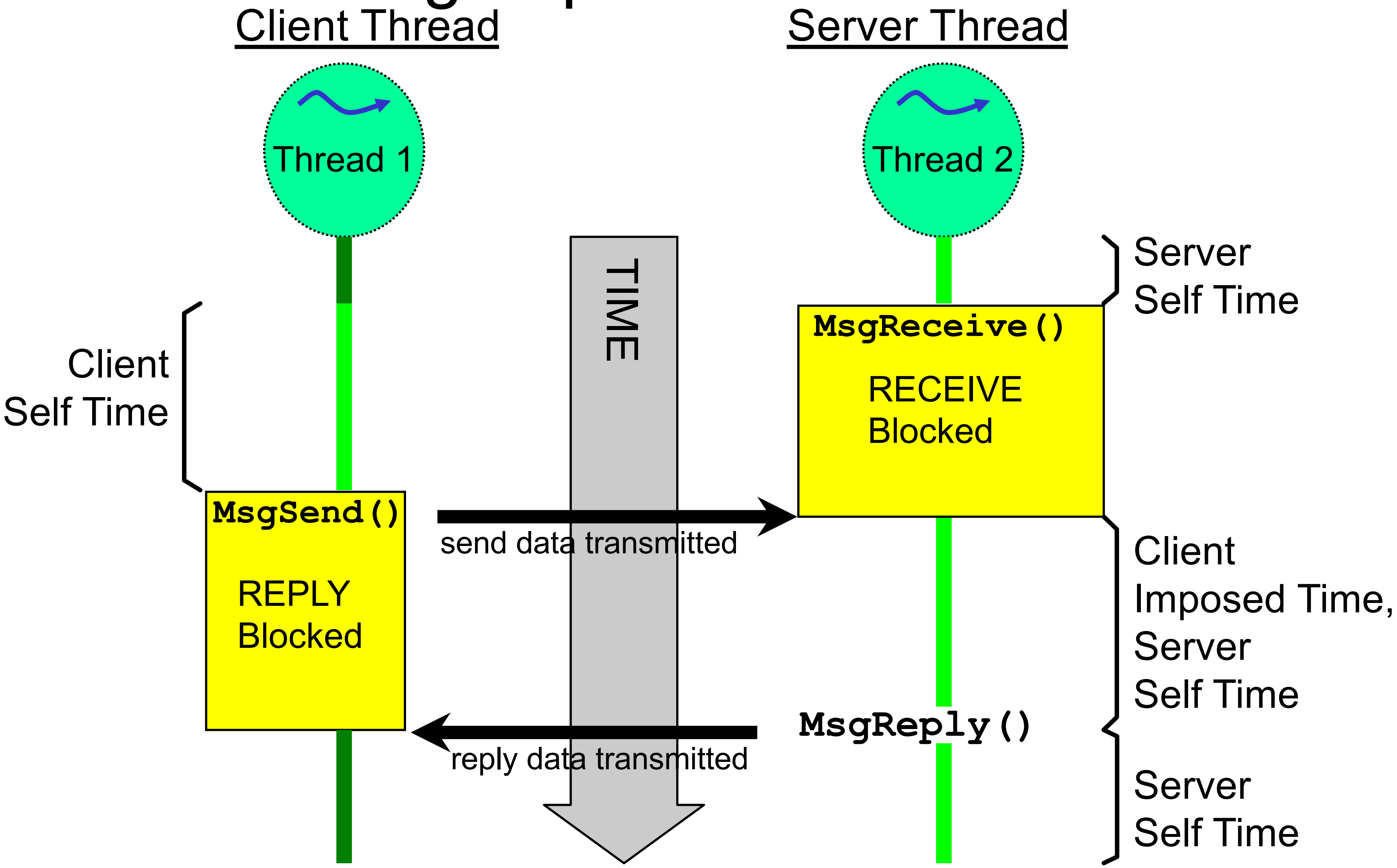
list of all threads that ran during the log

how much total time that the thread consumed (itself + server time)

time thread itself ran for

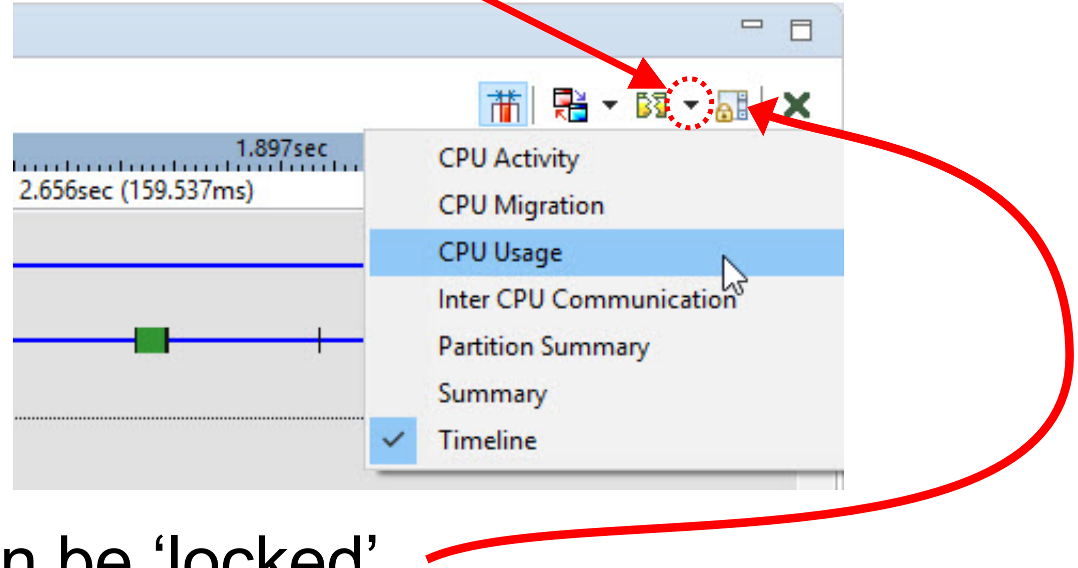
time that any servers consume while doing work on thread’s behalf

Understanding Imposed Time:



Splitting and locking panes:

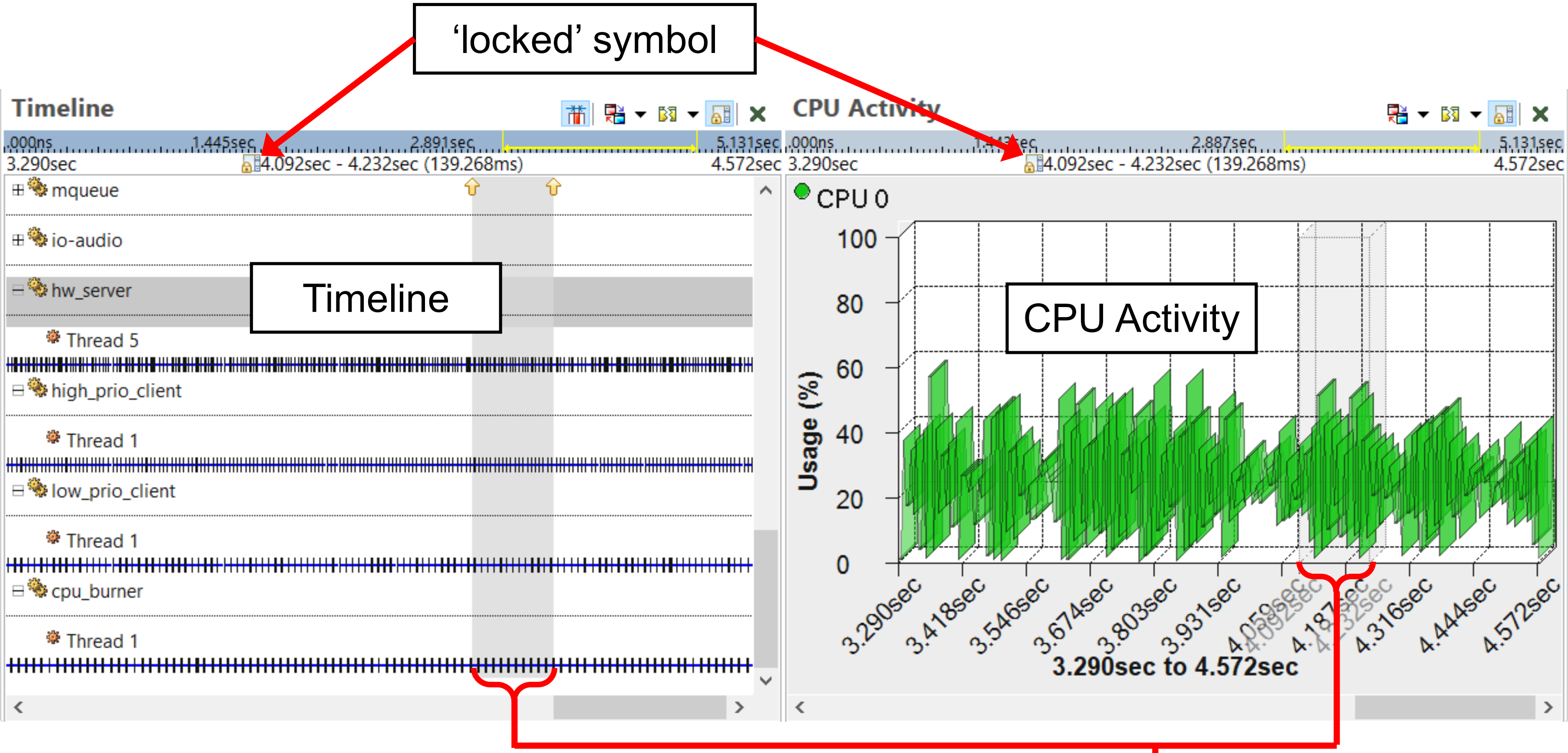
- you can split the panes
  - e.g. pane can be half Timeline and half CPU Usage
  - use this ‘pull-down’ icon



- panes can be ‘locked’
  - any scrolling, selection, or zooming that is done in one panes is reflected in the other(s)
  - all panes need to be locked

Displays – Split and Locked Example

Example of split and locked panes:



– after locking, selection and scrolling for both panes are synchronized

Topics:

- Overview
- Creating a Log
- Log Summary
- A Quick Tour
- Filtering Events and Event Owners
- Navigating through a log with the Timeline Pane
- Statistics
- CPU Activity Pane and CPU Usage Pane
  - → Exercise
- Tying The Trace to Your Code
- Multi-core Related Features
- Adaptive Partitioning: Partition Summary
- Conclusion

EXERCISE – CPU Usage and CPU Activity

Examine CPU usage:

1. use CPU Activity to gauge how busy the system was overall
2. use CPU Usage to determine which (non-idle) thread consumed the most CPU time
   - hold CTRL down and select the 2nd and 3rd busiest threads to see their CPU usage as well
3. use Client/Server Stats to look at time that clients forced servers to run
   - do you still consider the thread you found in 2. to have been the busiest?
4. split and lock CPU Usage with Timeline
   - zoom in on a peak of usage and look at the events in the Timeline to try and determine the reason for the peak

Topics:

- Overview
- Creating a Log
- Log Summary
- A Quick Tour
- Filtering Events and Event Owners
- Navigating through a log with the Timeline Pane
- Statistics
- CPU Activity Pane and CPU Usage Pane
- → Tying The Trace to Your Code
  - Exercise
- Multi-core Related Features
- Adaptive Partitioning: Partition Summary
- Conclusion

Tying Your Code to the Trace

So your client sent a message to the server…

- where in the client did it send this?
- there are two main ways to try and get this information:
  - annotate your code by inserting trace log events yourself
  - build your application(s) with Function Instrumentation Profiling and incorporate that data into your trace log and analysis

Logging your own Events

The *trace_log*()* functions can be used to log your own events:

- benefits:
  - general method of instrumenting your code
  - can be useful for aligning log events to your code
  - can be used from an interrupt handler
- different user event types:
  - simple user events - the data is 2 integers
  - string user events - the data is a string
  - complex user events – the data is a buffer

Logging your own Events

Examples of logging user events:

```
/* log a simple user event */
trace_logi(_NTO_TRACE_USERFIRST, 3, 5);

/* log a string user event */
trace_logf( _NTO_TRACE_USERFIRST+1,"Hello world");
```

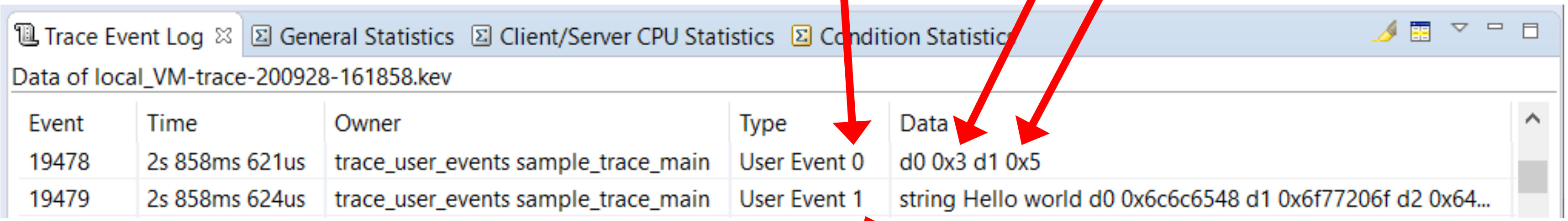
- the first argument is a code, which must be between `_NTO_TRACE_USERFIRST` and `_NTO_TRACE_USERLAST`
- for our two events we’ve used different but consecutive event IDs starting from the value of `_NTO_TRACE_USERFIRST`
- when looking at the log, you will see “User Event 0” for example, the 0 is `_NTO_TRACE_USERFIRST`
- with *trace_logf()*, it can also be a *printf()* style format string:

```
trace_logf( _NTO_TRACE_USERFIRST+2,”value: %d”, value );
```

Logging your own Events

The result of the previous page:

```
trace_logi(_NTO_TRACE_USERFIRST, 3, 5);
```



Data of local_VM-trace-200928-161858.kev

Event	Time	Owner	Type	Data
19478	2s 858ms 621us	trace_user_events sample_trace_main	User Event 0	d0 0x3 d1 0x5
19479	2s 858ms 624us	trace_user_events sample_trace_main	User Event 1	string Hello world d0 0x6c6c6548 d1 0x6f77206f d2 0x64...

```
trace_logf(_NTO_TRACE_USERFIRST+1, "Hello world");
```

Note: the IDE doesn't know what type of data you generated, it guesses based on the contents:

- integer elements with zeros in them may be displayed as a string

Logging your own Events - Parsing

Or, you can tell the IDE how to display your events:

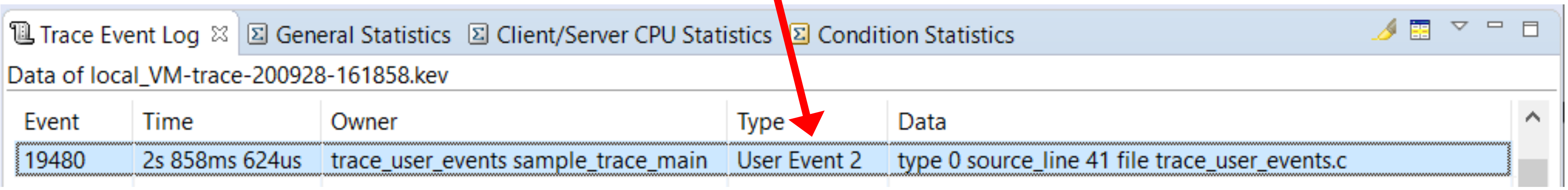
- can be global:
  - Windows-Preferences->QNX->System Profiler->User Event Data
- or per kev file:
  - Menu Click -> Properties -> User Event Data
- you will be specifying an xml file that tells the IDE how to parse your events

Parsing User events

For example, to parse:

```
struct mydata
{
int type; // 0 - trace, 1 - warn, 2 = error
int line;
char source_file[20];
} my_data_a;
...
trace_logb( _NTO_TRACE_USERFIRST+2, &my_data_a,
   sizeof(my_data_a));
```

and get:



Data of local_VM-trace-200928-161858.kev

Event	Time	Owner	Type	Data
19480	2s 858ms 624us	trace_user_events sample_trace_main	User Event 2	type 0 source_line 41 file trace_user_events.c

you would...

Parsing User Events

... use an event definition file like:

```
<?xml version="1.0" encoding="UTF-8" ?>

<eventdefinitions>
 <eventclass id="6" name="User Events">

   <event id="2" sformat="%4u1d type %4u1d source_line %1s0 file" />

 </eventclass>
</eventdefinitions>
```

– the **`sformat`** specifies a series of data and name specifications, each delimited by a space

Function Tracing

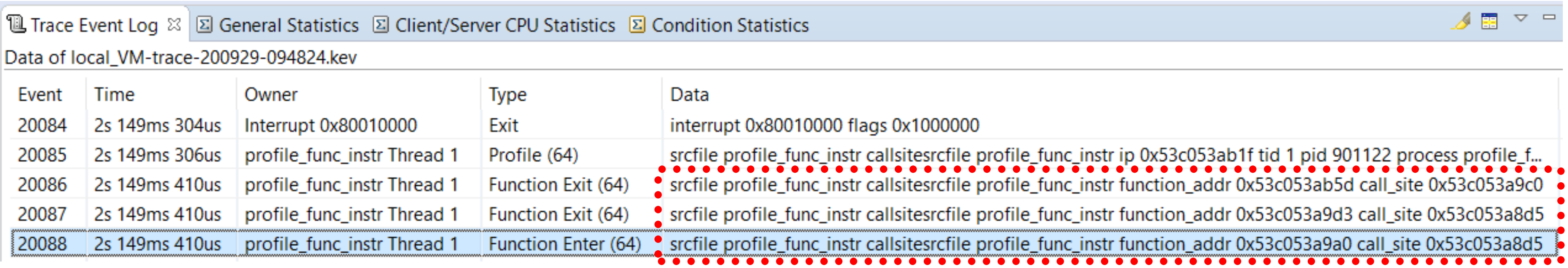
- Function Tracing is primarily an Application Profiling tool
  - it can be used with System Profiling
    - the data output must be configured to go to the kernel trace log
  - data from multiple applications can be collected in the same log
  - all source that you wish traced must be specially compiled
  - all applications must be linked to the profiling library

Preparing for Function Tracing:

- compile your source files to generate the tracing code:
  - `CFLAGS += -finstrument-functions`
  - this adds code to the start and end of every function call that generates data
- link against the tracing library:
  - `-lprofilingS`
  - this is a library, it must be after the objects on the link line, e.g.
  - `$(LD) $(LDFLAGS) main.o funcs.o -lprofilingS -o my_prog`
- tell the tracing library to send its data to the kernel event log:
  - set the environment variable **`QPROF_KERNEL_TRACE=1`**, e.g.:
  - `QPROF_KERNEL_TRACE=1 my_prog &`
  - you may wish to export this early or always set it, since it won't affect uninstrumented applications

Function Tracing

In the log it looks like:



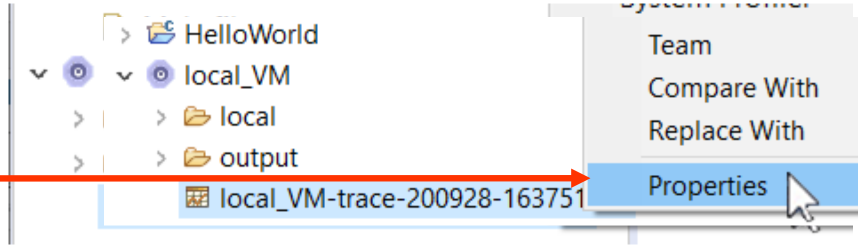
Data of local_VM-trace-200929-094824.kev

Event	Time	Owner	Type	Data
20084	2s 149ms 304us	Interrupt 0x80010000	Exit	interrupt 0x80010000 flags 0x1000000
20085	2s 149ms 306us	profile_func_instr Thread 1	Profile (64)	srcfile profile_func_instr callsitesrcfile profile_func_instr ip 0x53c053ab1f tid 1 pid 901122 process profile_f...
20086	2s 149ms 410us	profile_func_instr Thread 1	Function Exit (64)	srcfile profile_func_instr callsitesrcfile profile_func_instr function_addr 0x53c053ab5d call_site 0x53c053a9c0
20087	2s 149ms 410us	profile_func_instr Thread 1	Function Exit (64)	srcfile profile_func_instr callsitesrcfile profile_func_instr function_addr 0x53c053a9d3 call_site 0x53c053a8d5
20088	2s 149ms 410us	profile_func_instr Thread 1	Function Enter (64)	srcfile profile_func_instr callsitesrcfile profile_func_instr function_addr 0x53c053a9a0 call_site 0x53c053a8d5

- – for finding these events, they are in the System category
- – but we only have raw function addresses
  - if the application was built with symbol information we can translate this to function names and lines
  - we can do this…

To add source code information:

In the Project Explorer view, bring up the Properties of the kev file

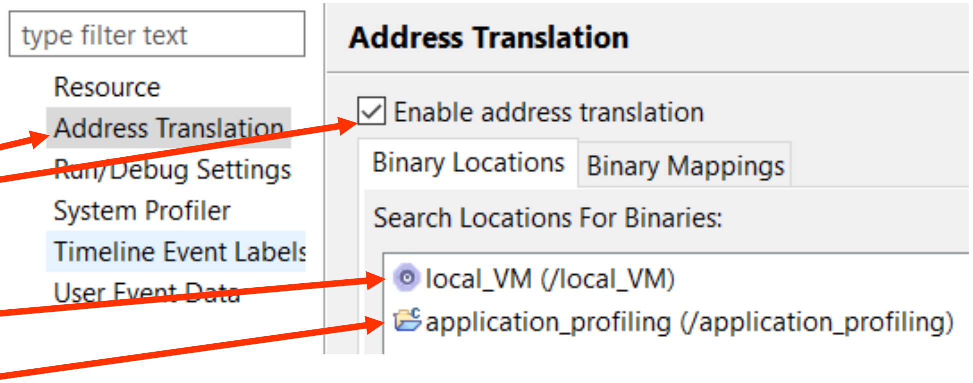


Select Address Translation

Click to Enable

Remove the Target Project

And add source locations



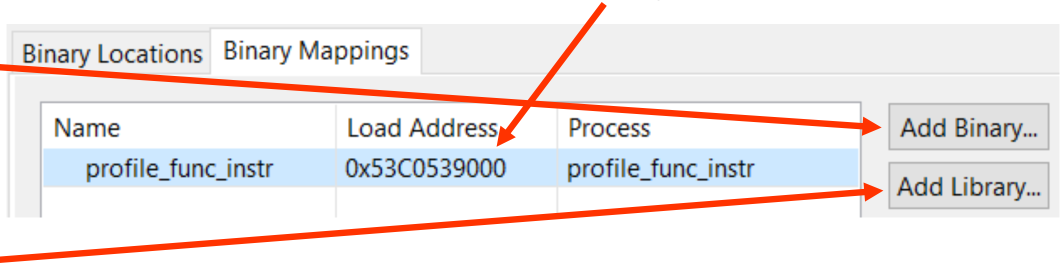
☑ Enable address translation

Search Locations For Binaries:

- local_VM (/local_VM)
- application_profiling (/application_profiling)

Enter binary load address

Then add the binaries for which you want translations

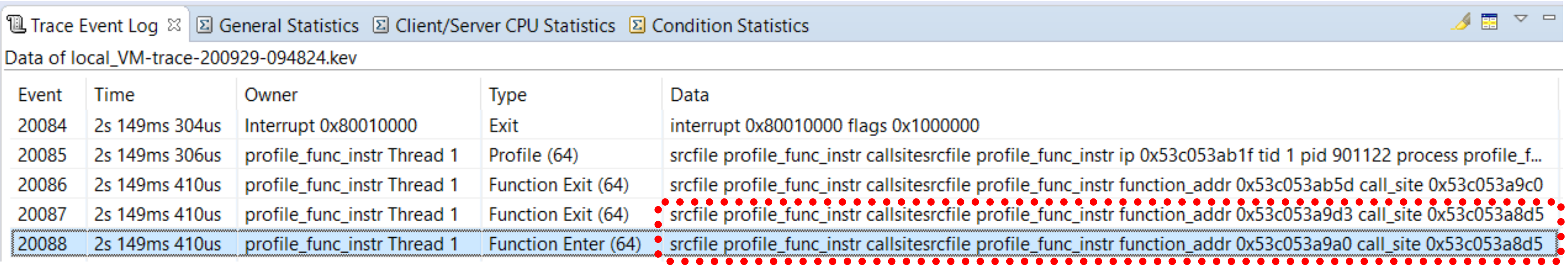


Name	Load Address	Process
profile_func_instr	0x53C0539000	profile_func_instr

You can add libraries to a binary as well

Function Tracing

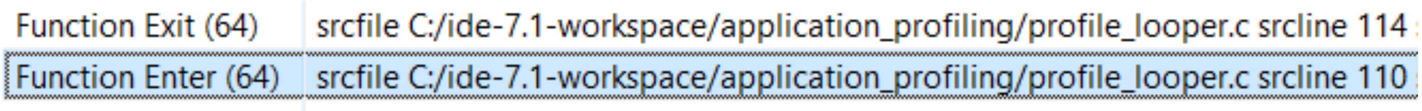
So, after adding translation:



Data of local_VM-trace-200929-094824.kev

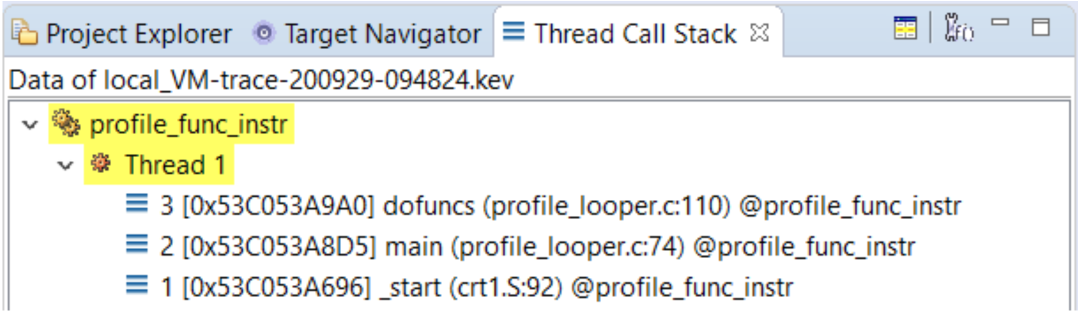
Event	Time	Owner	Type	Data
20084	2s 149ms 304us	Interrupt 0x80010000	Exit	interrupt 0x80010000 flags 0x1000000
20085	2s 149ms 306us	profile_func_instr Thread 1	Profile (64)	srcfile profile_func_instr callsitesrcfile profile_func_instr ip 0x53c053ab1f tid 1 pid 901122 process profile_f...
20086	2s 149ms 410us	profile_func_instr Thread 1	Function Exit (64)	srcfile profile_func_instr callsitesrcfile profile_func_instr function_addr 0x53c053ab5d call_site 0x53c053a9c0
20087	2s 149ms 410us	profile_func_instr Thread 1	Function Exit (64)	srcfile profile_func_instr callsitesrcfile profile_func_instr function_addr 0x53c053a9d3 call_site 0x53c053a8d5
20088	2s 149ms 410us	profile_func_instr Thread 1	Function Enter (64)	srcfile profile_func_instr callsitesrcfile profile_func_instr function_addr 0x53c053a9a0 call_site 0x53c053a8d5

Now looks like:



Function Exit (64)	srcfile C:/ide-7.1-workspace/application_profiling/profile_looper.c srcline 114
Function Enter (64)	srcfile C:/ide-7.1-workspace/application_profiling/profile_looper.c srcline 110

And the Thread Call Stack view can tell you where you are:



Data of local_VM-trace-200929-094824.kev

- profile_func_instr
  - Thread 1
    - 3 [0x53C053A9A0] dofuncs (profile_looper.c:110) @profile_func_instr
    - 2 [0x53C053A8D5] main (profile_looper.c:74) @profile_func_instr
    - 1 [0x53C053A696] _start (crt1.S:92) @profile_func_instr

Naming Threads

You can also name your threads:

- “interrupt_thread” is more useful than “Thread 2”
- for example:

```
pthread_setname_np(0,“my_thread_name” );
```

- this will also name the thread for the System Information Perspective

Topics:

- Overview
- Creating a Log
- Log Summary
- A Quick Tour
- Filtering Events and Event Owners
- Navigating through a log with the Timeline Pane
- Statistics
- CPU Activity Pane and CPU Usage Pane
- Tying The Trace to Your Code
  - → – Exercise
- Multi-core Related Features
- Adaptive Partitioning: Partition Summary
- Conclusion

EXERCISE – trace_log*() and naming threads

Experiment with *trace_log*()* and naming threads:

- modify a program to add some *trace_log*()* calls to insert traces into the log
    - the compiler built-in variables **`__LINE__`** and **`__FILE__`** may be useful
- also name any thread or threads in the program
- run the program and get a log
- find those events in the resulting log

- rebuild a program with function tracing as well, or run a program already built that way, and get that data

System Profiling

Topics:

Overview

Creating a Log

Log Summary

A Quick Tour

Filtering Events and Event Owners

Navigating through a log with the Timeline Pane

Statistics

CPU Activity Pane and CPU Usage Pane

Tying The Trace to Your Code

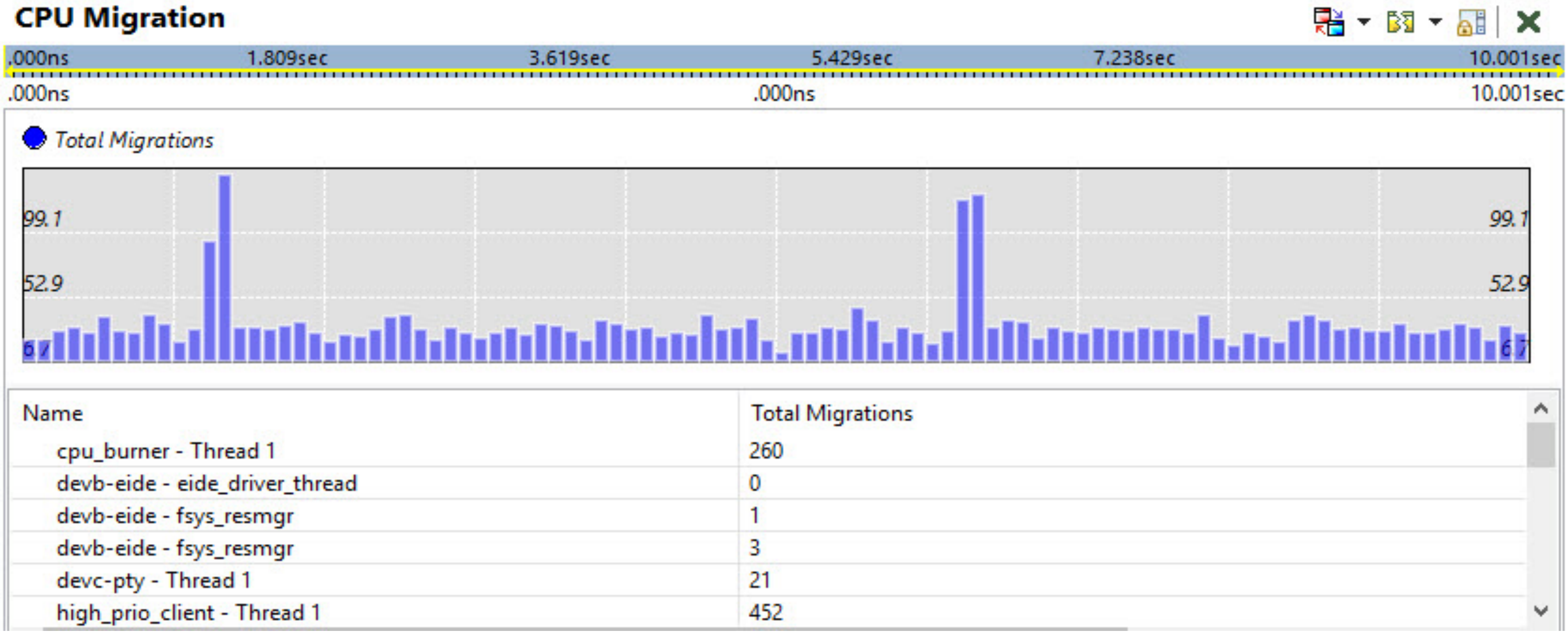
→ Multi-core Related Features

Adaptive Partitioning: Partition Summary

Conclusion

Migration of a thread between CPUs:

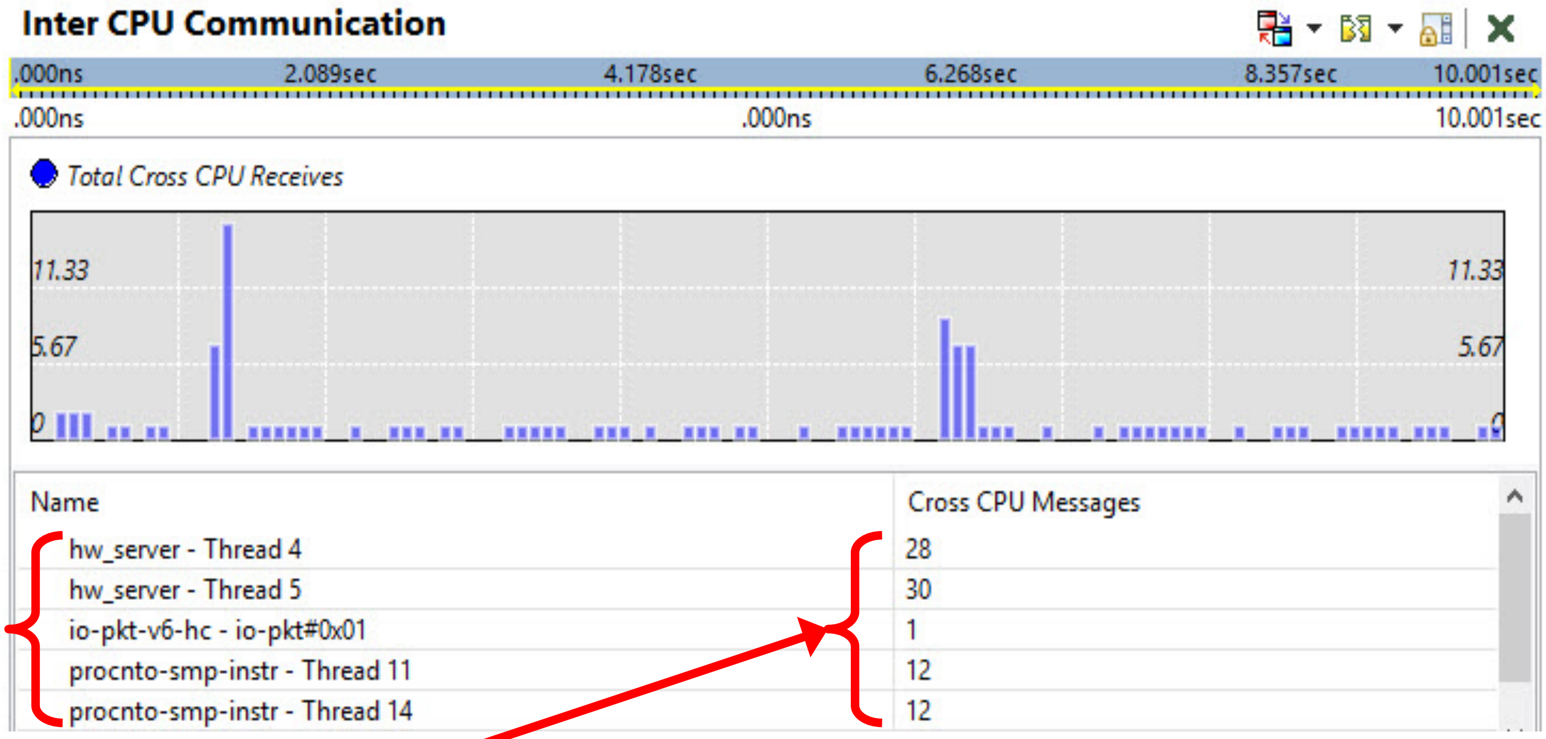
- allows a thread to run when it would not have the opportunity if there was no migration
- may cause a performance penalty due to inefficient use of CPU cache
- CPU Migration Pane will indicate how many migrations occurred



Name	Total Migrations
cpu_burner - Thread 1	260
devb-eide - eide_driver_thread	0
devb-eide - fsys_resmgr	1
devb-eide - fsys_resmgr	3
devc-pty - Thread 1	21
high_prio_client - Thread 1	452

Inter-CPU communication Pane:

- cross-CPU message passing also can cause an inefficient utilization of the CPU cache



Inter CPU Communication

Name	Cross CPU Messages
hw_server - Thread 4	28
hw_server - Thread 5	30
io-pkt-v6-hc - io-pkt#0x01	1
procnto-smp-instr - Thread 11	12
procnto-smp-instr - Thread 14	12

servers

number of times clients and servers were scheduled on a different CPU

Topics:

Overview

Creating a Log

Log Summary

A Quick Tour

Filtering Events and Event Owners

Navigating through a log with the Timeline Pane

Statistics

CPU Activity Pane and CPU Usage Pane

Tying The Trace to Your Code

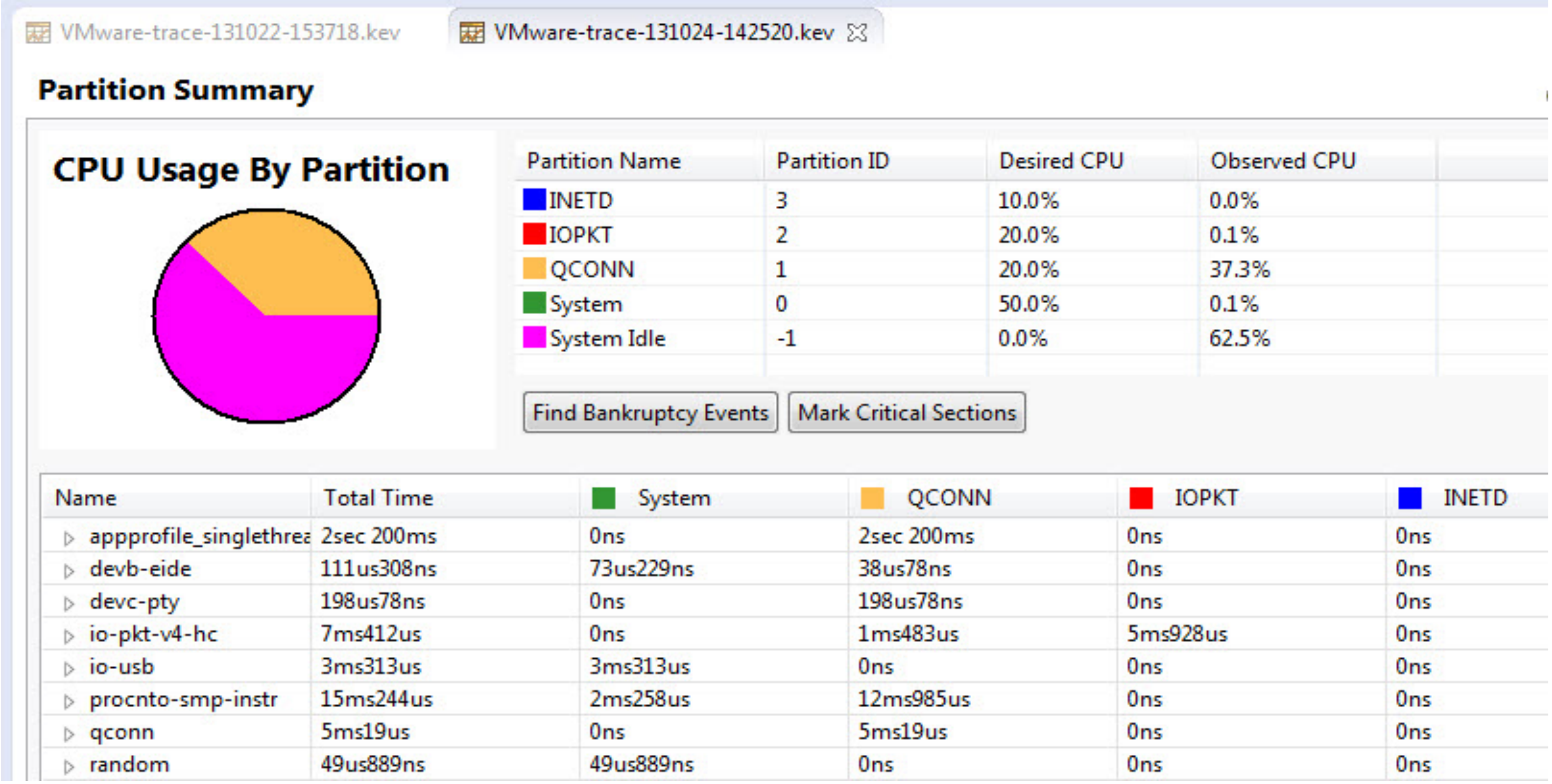
Multi-core Related Features

→ **Adaptive Partitioning: Partition Summary**

Conclusion

Partition Summary Pane:

- lists partitions and CPU usage within
- helps to find problem areas where critical threads didn’t get enough CPU time



VMware-trace-131022-153718.kev
VMware-trace-131024-142520.kev

Partition Summary

CPU Usage By Partition

Partition Name	Partition ID	Desired CPU	Observed CPU
🟦 INETD	3	10.0%	0.0%
🟥 IOPKT	2	20.0%	0.1%
🟧 QCONN	1	20.0%	37.3%
🟩 System	0	50.0%	0.1%
🟪 System Idle	-1	0.0%	62.5%

Find Bankruptcy Events
Mark Critical Sections

Name	Total Time	🟩 System	🟧 QCONN	🟥 IOPKT	🟦 INETD
▷ appprofile_singlethrea	2sec 200ms	0ns	2sec 200ms	0ns	0ns
▷ devb-eide	111us308ns	73us229ns	38us78ns	0ns	0ns
▷ devc-pty	198us78ns	0ns	198us78ns	0ns	0ns
▷ io-pkt-v4-hc	7ms412us	0ns	1ms483us	5ms928us	0ns
▷ io-usb	3ms313us	3ms313us	0ns	0ns	0ns
▷ procnto-smp-instr	15ms244us	2ms258us	12ms985us	0ns	0ns
▷ qconn	5ms19us	0ns	5ms19us	0ns	0ns
▷ random	49us889ns	49us889ns	0ns	0ns	0ns

Topics:

- Overview
- Creating a Log
- Log Summary
- A Quick Tour
- Filtering Events and Event Owners
- Navigating through a log with the Timeline Pane
- Statistics
- CPU Activity Pane and CPU Usage Pane
- Tying The Trace to Your Code
- Multi-core Related Features
- Adaptive Partitioning: Partition Summary
- → Conclusion

Conclusion

You learned:

- how the instrumented kernel, tracelogger and the IDE can be used to gather events
- how to use *trace_log*()* to log your own data
- how to control logging of this data from:
  - the IDE
  - the command line using **`tracelogger`** and
  - from your own code
- how to analyze this data using the IDE