What's Slowing You Down?

Using Kernel Event Tracing to Uncover Performance Issues

Malte Mundt, Senior Field Application Engineer, BlackBerry QNX
mmundt@blackberry.com

Agenda

01

Overview: What does Kernel Event Tracing mean?

02

System Profiler: Visualization and Interpretation

03

Finding out about… Processing Time

04

Finding out about… Dependencies

05

Finding out about… Bottlenecks

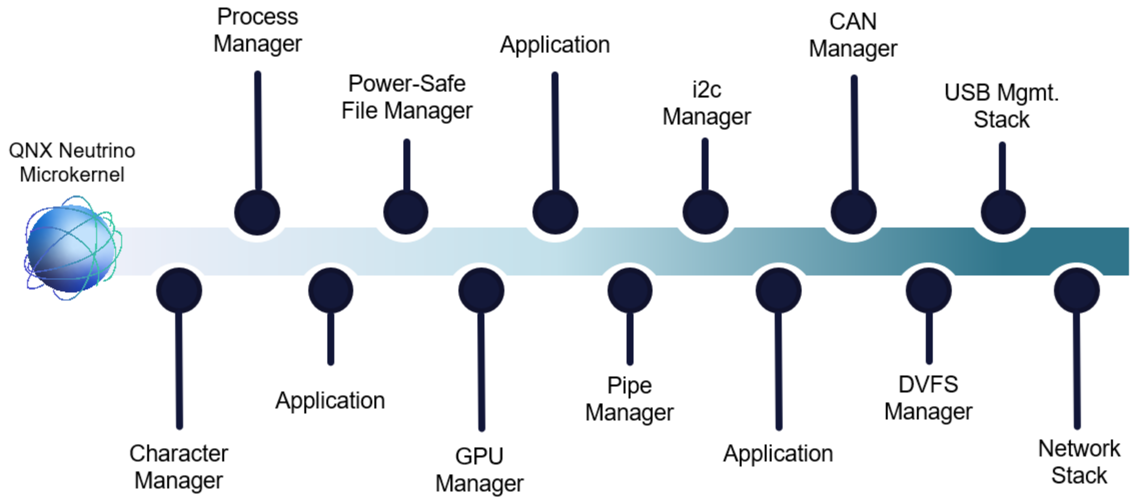
06

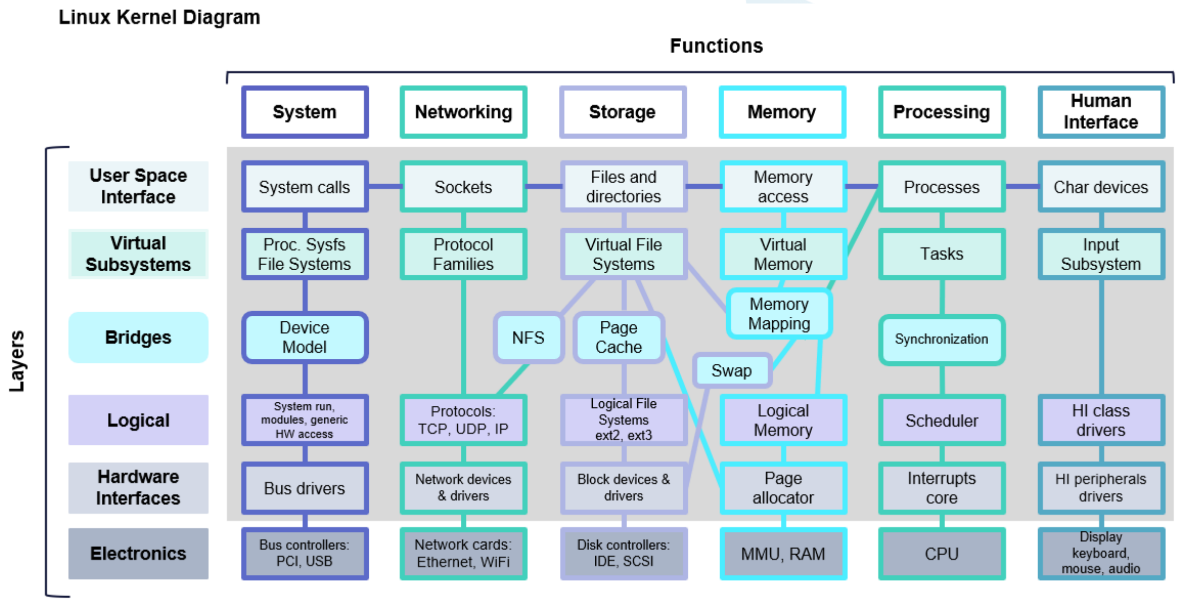
Conclusion
Q&A

What does Kernel Event Tracing mean?

Overview

What Kernel Tracing means in a Microkernel environment





- With the QNX Microkernel, everything is a process with threads: Drivers, Stacks, Services, Applications
- All processes can use the same APIs, i.e. ANSI C / C99 / C11, POSIX PSE54 etc.
- Kernel Event Tracing is logging kernel events, System Profiling visualizes kernel event log
- The QNX microkernel can be extended to become a Hypervisor: VMs are special processes

Performance analysis from the kernel's point of view

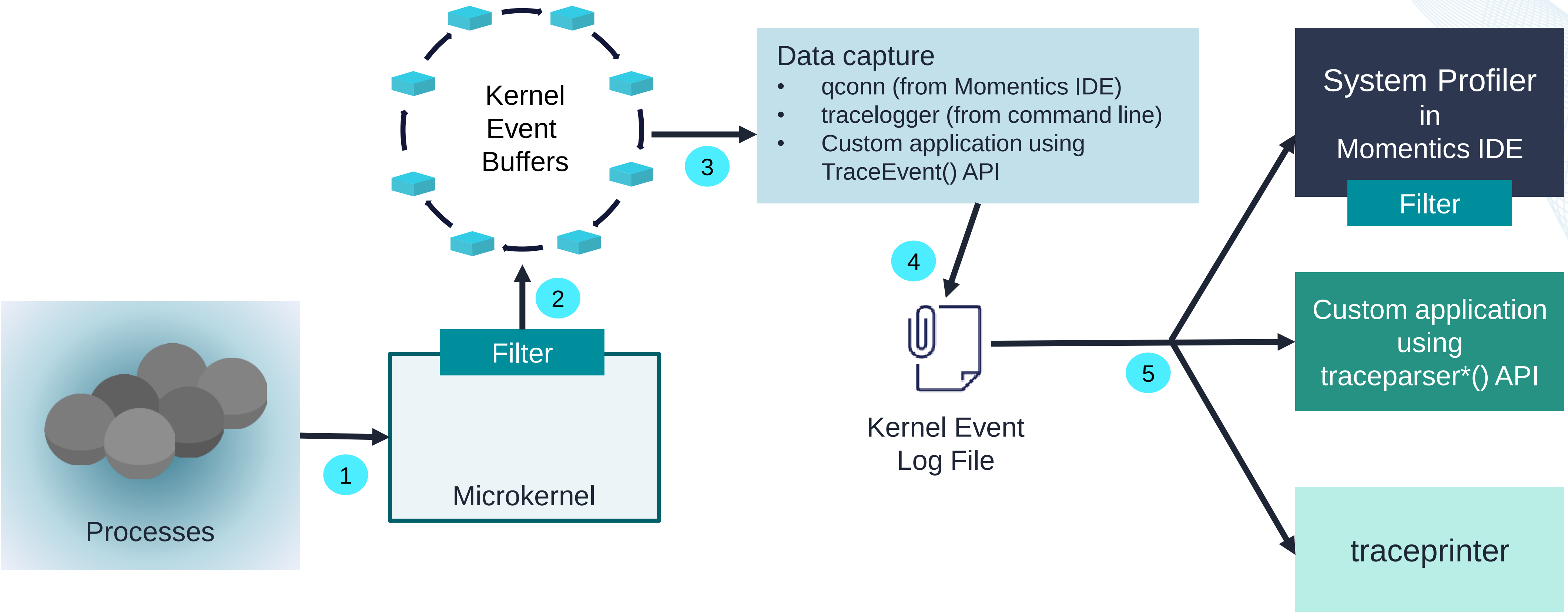
Kernel Event Tracing… is using the instrumentation built into the QNX kernel for logging of events:

- Creation and Exit of Threads, Processes, Virtual Machines
- Kernel calls made by Threads
- Message Passing based IPC, Thread Synchronization
- Thread State Changes and Scheduling
- Guest Exit/Guest Entry (Hypervisor)
- Interrupt Handling
- User-defined trace events

… gives you a holistic view on overall system performance

… lets you drill down to the details

QNX Kernel Event Tracing Data Flow



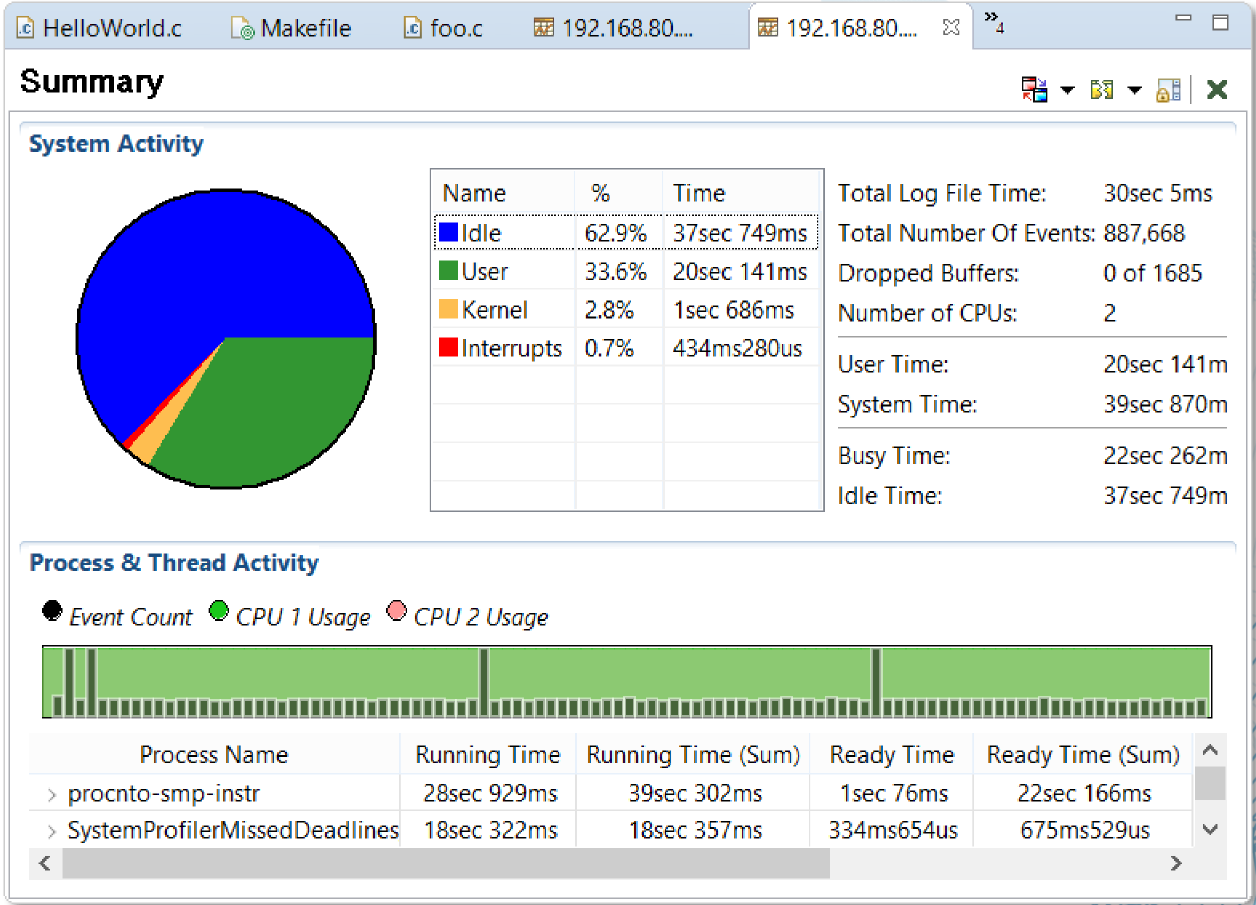
Kernel Event Tracing - Visualization and Interpretation

System Profiler

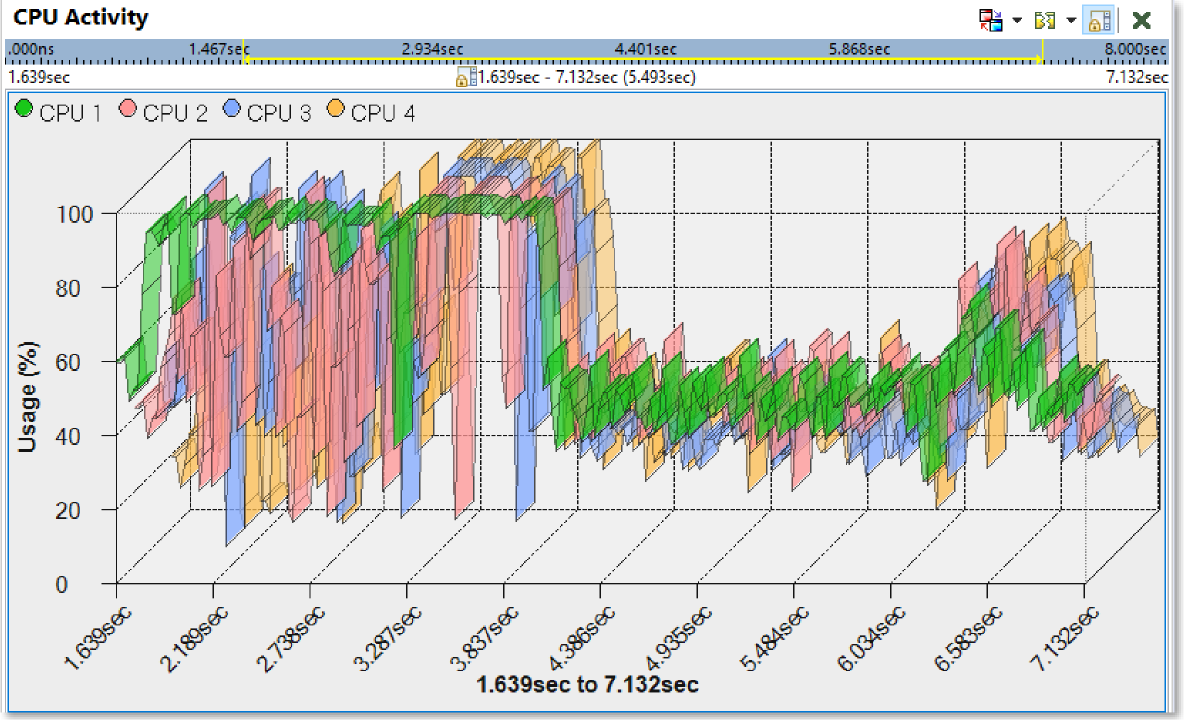
BlackBerry QNX

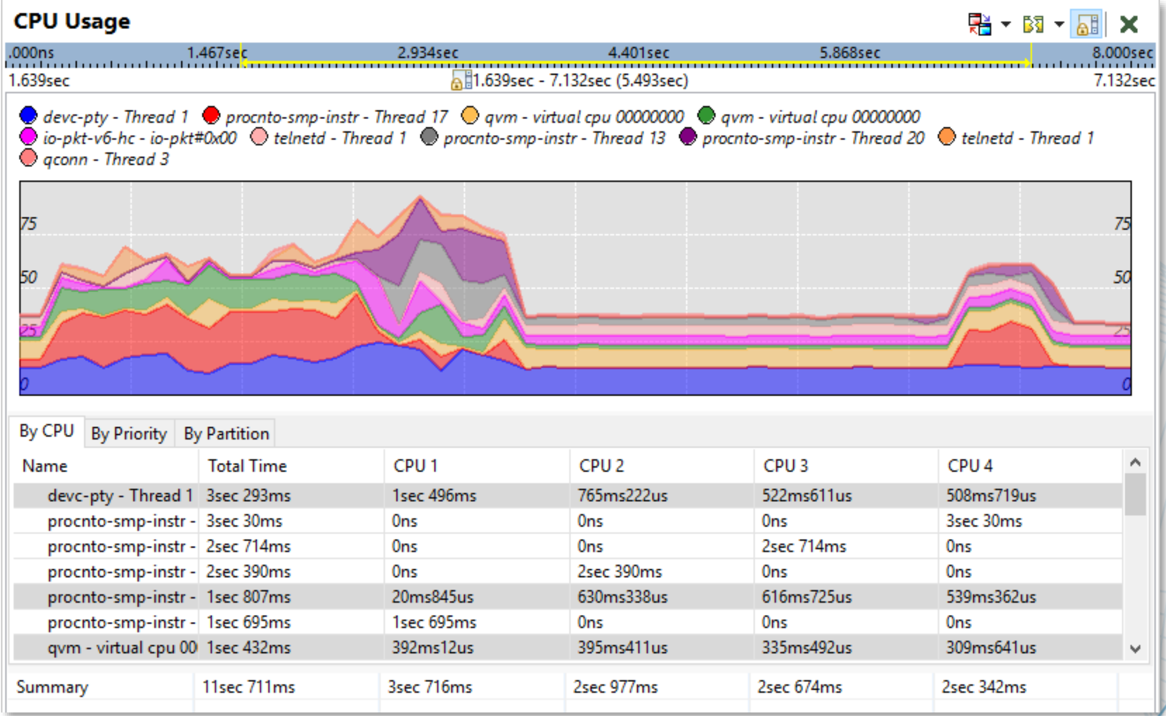
Find out overall CPU usage level

- Events have microsecond resolution timestamps
  - Number of events displayed in bar graph
- System Profiler calculates CPU time
  - Idle
  - User Space processes
  - Kernel
  - Interrupt handlers
- Most active processes also shown in Summary



Identify sources of CPU load



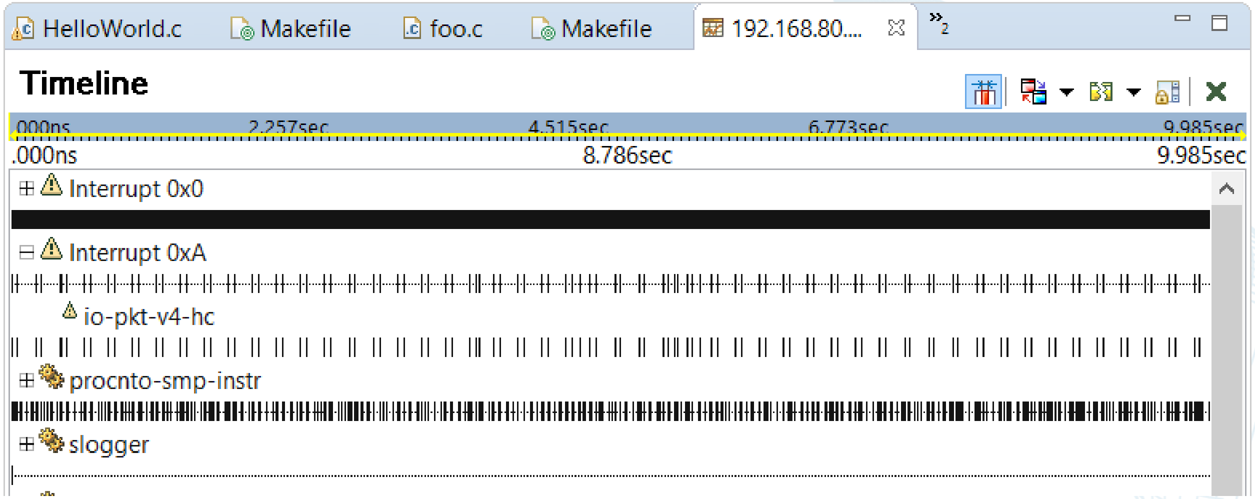


Name	Total Time	CPU 1	CPU 2	CPU 3	CPU 4
devc-pty - Thread 1	3sec 293ms	1sec 496ms	765ms222us	522ms611us	508ms719us
procnto-smp-instr -	3sec 30ms	0ns	0ns	0ns	3sec 30ms
procnto-smp-instr -	2sec 714ms	0ns	0ns	2sec 714ms	0ns
procnto-smp-instr -	2sec 390ms	0ns	2sec 390ms	0ns	0ns
procnto-smp-instr -	1sec 807ms	20ms845us	630ms338us	616ms725us	539ms362us
procnto-smp-instr -	1sec 695ms	1sec 695ms	0ns	0ns	0ns
qvm - virtual cpu 00	1sec 432ms	392ms12us	395ms411us	335ms492us	309ms641us
Summary	11sec 711ms	3sec 716ms	2sec 977ms	2sec 674ms	2sec 342ms

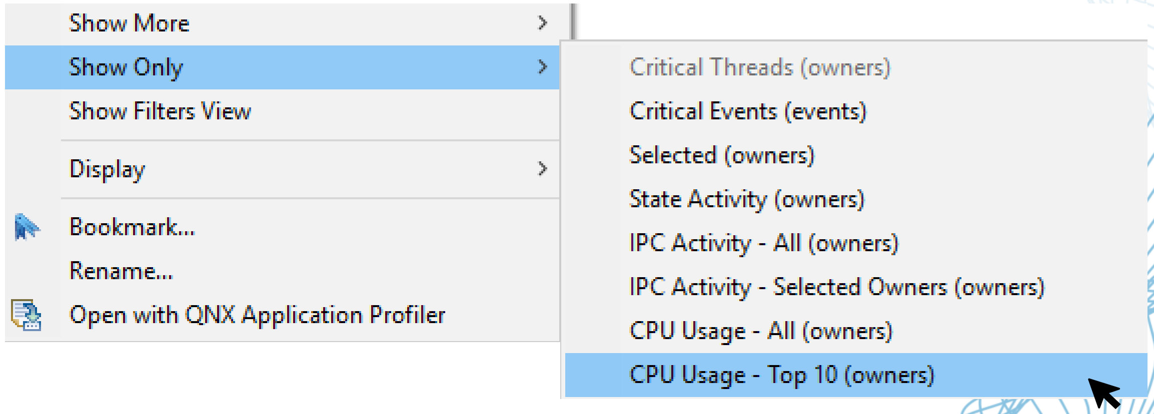
- Microsecond-accurate CPU activity graph per core – identify peaks, verify actual usage
- Zoom in to see CPU toggling between active and idle
- Identify processes causing base load, peaks
- Select/deselect individual threads to see CPU load accumulating

Maximum Detail: The Time Microscope

- Timeline view sorts all events by source (processes and threads) to represent their exact timing
- For each event source a timeline is drawn
  - Events represented by vertical tick marks
  - Interrupts listed at the top



- Large systems can have hundreds of processes, thousand of threads – filtering is key
  - Filter out unneeded event classes when logging
  - Event Display filtering: Show only top 10 CPU users, filter certain process names, event classes…



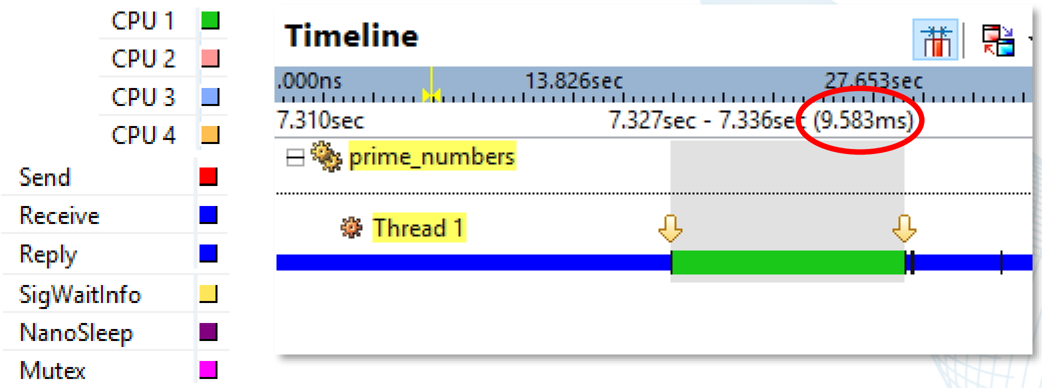
What can you find out with a Kernel Event Trace?

Processing Time

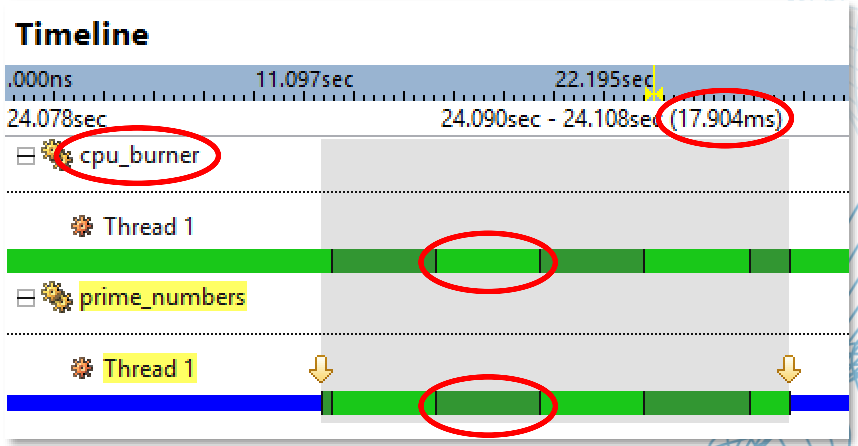
BlackBerry QNX

Exact measurement of processing times

- Threads that want to use CPU are *Ready*
  - Scheduler makes them *Running* on available cores
- Threads waiting for something are Blocked
  - Color shows why the thread is not running
  - All thread states are documented

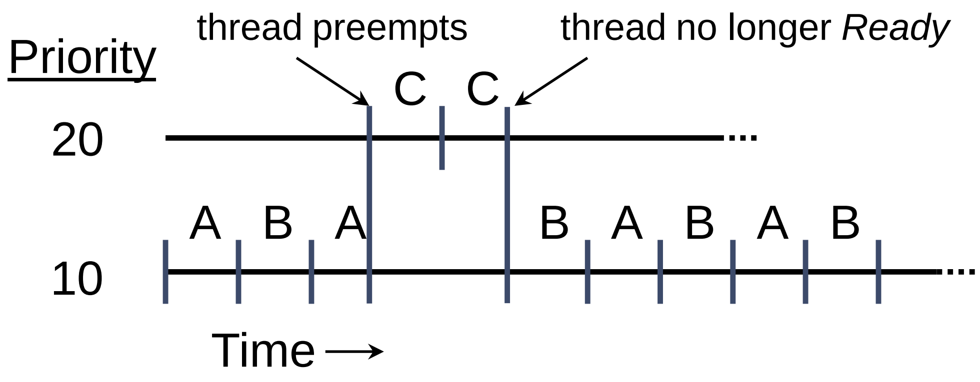


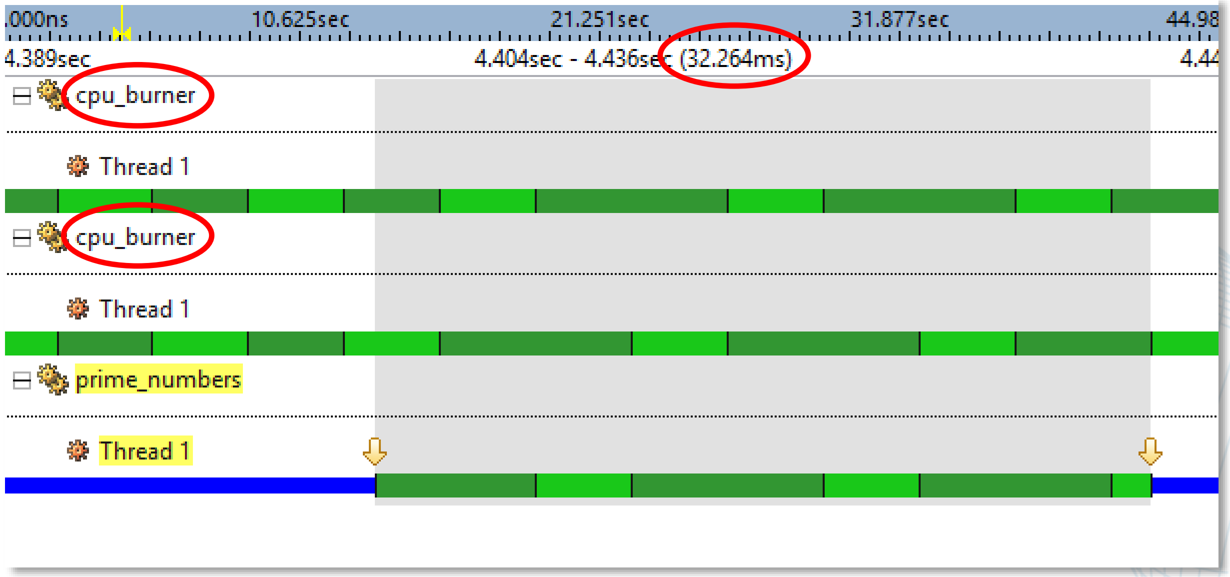
- Example: An event happens, you have to quickly calculate something – Timeline shows you exactly how long it takes
- But when CPU is fully loaded…
  - … suddenly your calculation takes much longer
- Timeline over multiple threads helps understanding why

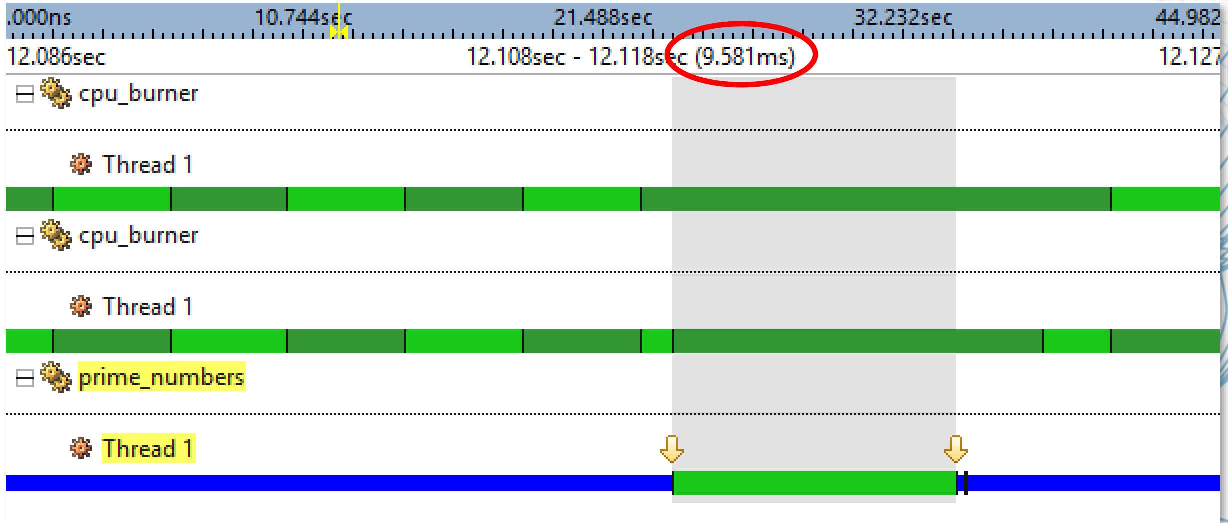


Analyze scheduler actions, adjust priorities

- More threads *Ready* – calculation slower…
- With Round Robin only, Ready Queue can get long
- Scheduler evaluates thread Priority first
- Let's increase the priority of our calculation
  - Now it's back to its original speed
- Same total load, drastically different behavior

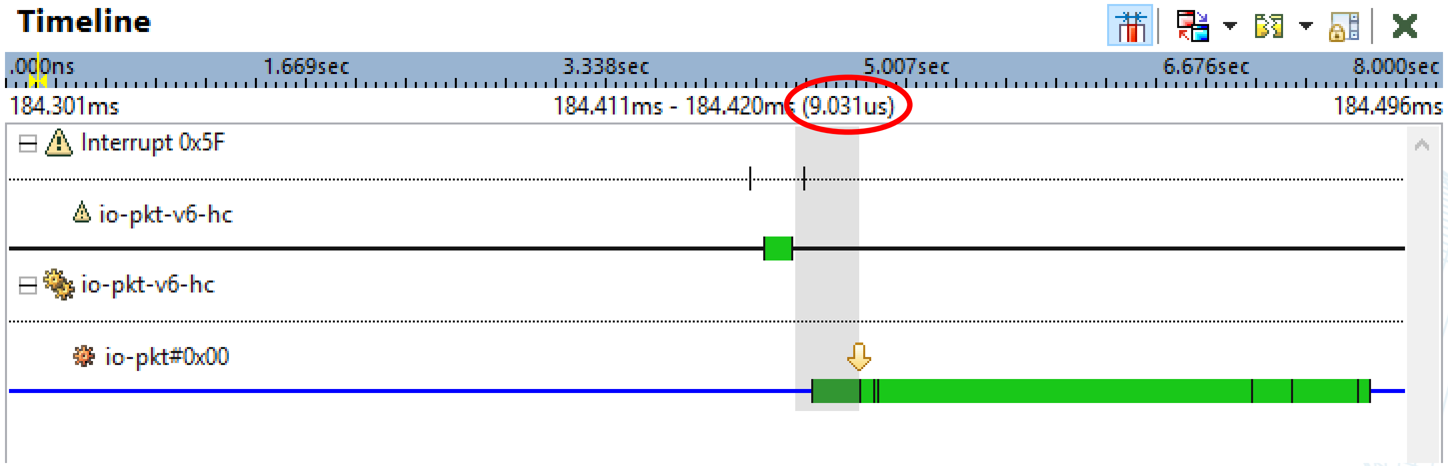






Interrupt frequency and handling times

- Find out actual frequency of interrupts
- How long until handler gets called?
- IRQ handler execution time?
- Thread to be scheduled?



- How long does it take until the thread is made *Running* ? Depends on priority…
- Thread woken up on behalf of interrupt handler is executing in full POSIX environment
  - On any available core and in your configured priority

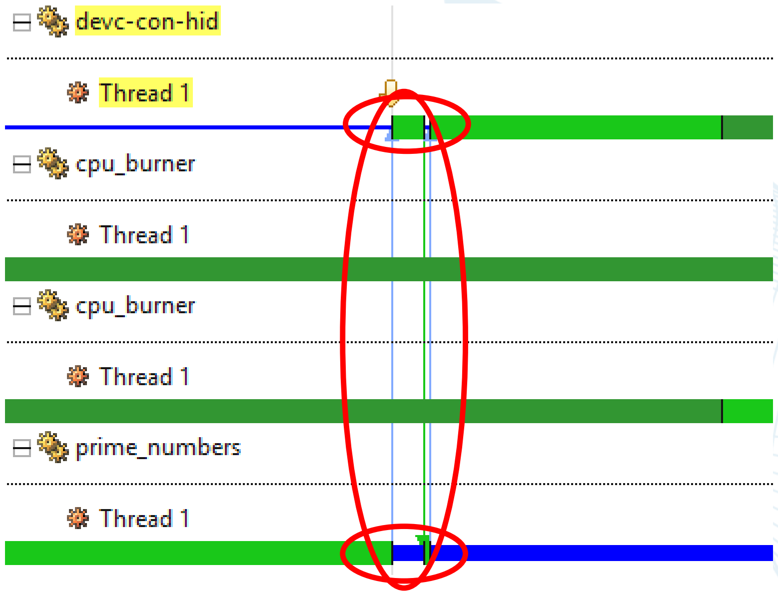
- Optimize? Use only a handler or only a thread

What can you find out with a Kernel Event Trace?

Dependencies

Performance dependencies – Clients and Servers

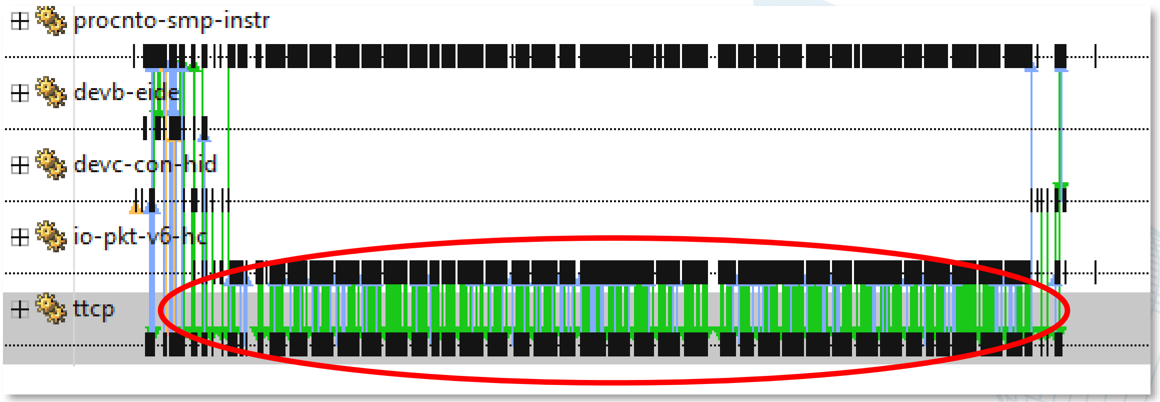
- Our CPU load example prints to console when done – it's IPC
- Visualize Message Passing based IPC to see dependencies
  - Servers and clients are both processes, interaction is IPC
  - Drivers and Stacks (Network, USB, Audio) are server processes
- Application performance depends on servers they are interacting with
  - CPU load of servers induced by their clients



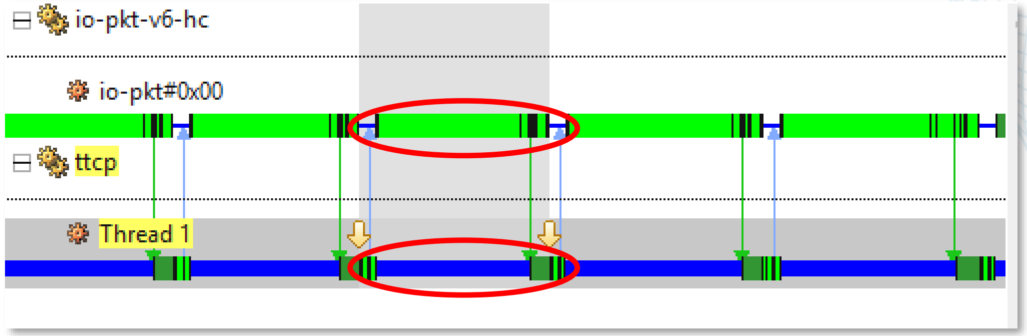
- System Profiler enables you to identify and measure impact of servers on your application
  - Measure how long your thread is blocked when it's a client waiting for a server
  - Look at server threads' processing time
  - Find other CPU-heavy threads that slow down server

Visualize and analyze IPC interactions

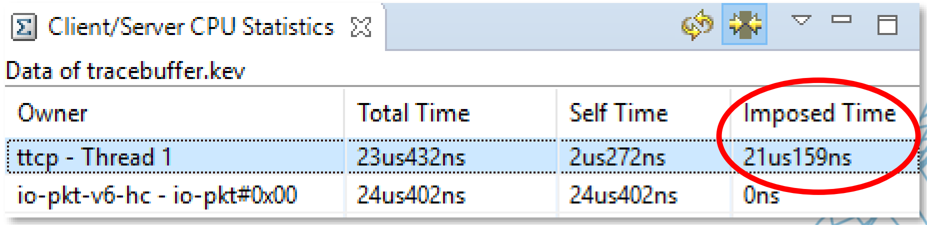
- Example: Application sends data out via TCP/IP
- Select process of interest, filter on IPC partners
- Our test app is client of various servers
  - Heavy dependency on network stack is obvious



- Server spends a lot of time for application
- Client spends a lot of time waiting for sever



- Client/Server CPU statistics reveal *Imposed Time*

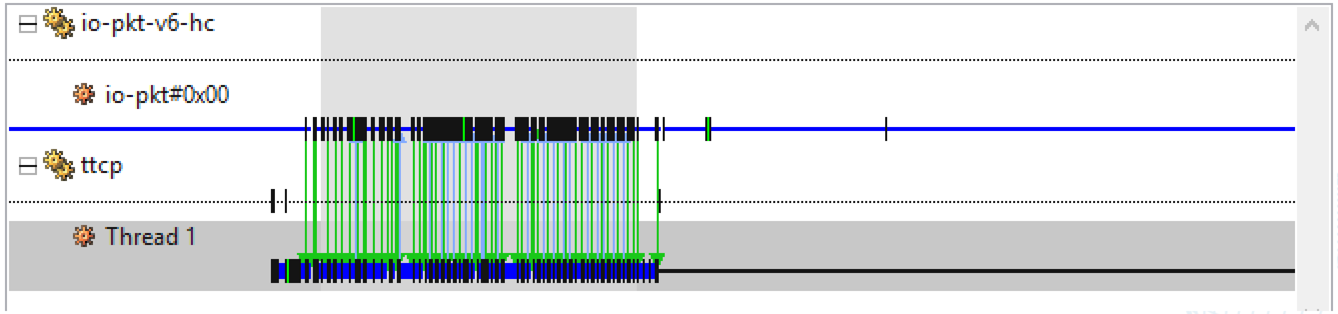


Owner	Total Time	Self Time	Imposed Time
ttcp - Thread 1	23us432ns	2us272ns	21us159ns
io-pkt-v6-hc - io-pkt#0x00	24us402ns	24us402ns	0ns

Identify IPC frequency, Client wait times

- Client needs time to prepare buffer
- Server process time usually is sum of
  - Fixed time needed to process request
  - Variable time depending on the size of the data
  - Both include hardware interaction
- Statistics View shows Client wait time

- Determine Optimization options:
  - Server processing, Client calls, IPC parameters, Shared Memory

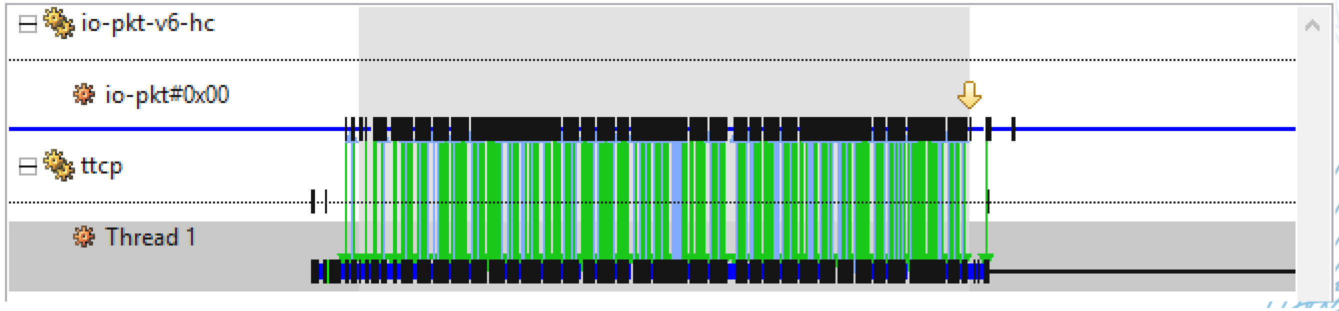


States

Event	Occu...	Avg Dur...	Max Dura...	Min Dur...	Total Duration
Reply	114	140us50...	743us188ns	321ns	16ms18us

Events

Event	Occurrences
Send Message	113



States

Event	Occu...	Avg Dur...	Max Dura...	Min Dur...	Total Duration
Reply	814	33us755ns	549us317ns	224ns	27ms477us

Events

Event	Occurrences
Send Message	813

What can you find out with a Kernel Event Trace?

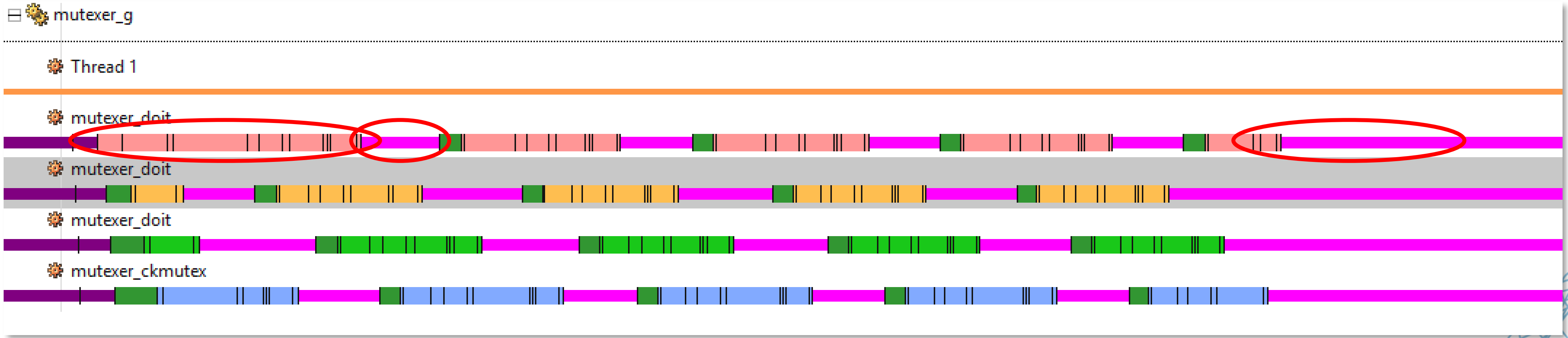
Bottlenecks

Hey, we are waiting for you! The absence of full CPU load

- Scenario: Something is too slow (i.e. button press, reaction takes 2s) – but CPU is not fully loaded
- Resource contention may slow you down – waiting for a Mutex, Condvar, Semaphore
- System Profiler will help you identify:
  - Wait time – Average and Maximum duration
  - Threads which are holding Mutex longest while others wait
  - Unveil thread that does not release Mutex at all (Deadlock!)

States

Event	Occu...	Avg Duration	Max Duration	Min Duration	Total Duration
Running	5	85us969ns	104us575ns	37us651ns	429us863ns
Ready	5	15us121ns	18us90ns	13us833ns	75us606ns
Mutex	5	292us515ns	1ms206us	51us560ns	1ms462us
NanoSleep	1	21us757ns	21us757ns	21us757ns	21us757ns



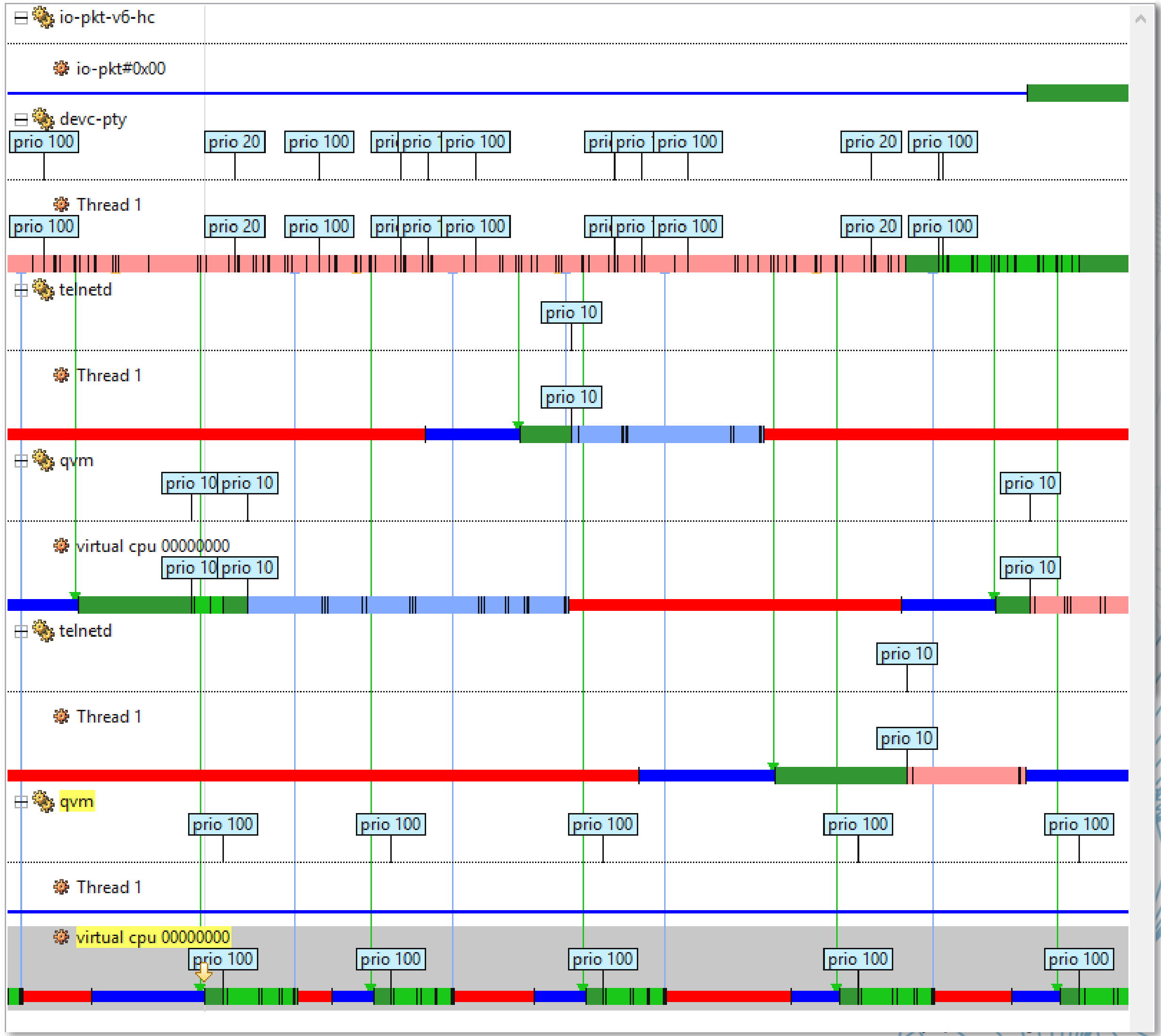
Servers can be bottlenecks – how priority inheritance helps

- Example: QNX Hypervisor
- I telnet in twice, and from each console I start a virtual machine
  - Linux guest at priority 10
  - QNX guest at priority 100
- Linux guest prints out text to console …
- … then QNX guest also prints text simultaneously
- Servers managing shared resources will inherit client priority – wait time minimized

Unveiling another kind of bottleneck

- Everyone seems busy – but CPU not fully loaded
- Long red lines are eye-catching: Send-Blocked!
  - Meaning: Receiver busy, can not receive
- Multiple threads are Send-Blocked

- Priority Labels show scheduling priority…
  - “devc-pty” priority is changing while it’s running
  - It is bombarded with requests
  - It is running the entire time
  - It’s single-threaded! 😯



What’s Slowing You Down?

Conclusion

BlackBerry | QNX

What‘s Slowing You Down? – Conclusion

Processing Time

- Sources of CPU load
- Priorities, Ready Queue
- Interrupts

Dependencies

- Client/Server Relationships
- Imposed Time
- IPC Frequency

Bottlenecks

- Threads waiting for each other
- Priority Inheritance
- Send-Blocked clients

Kernel Event Tracing and System Profiling support you to:

- Understand your system, realize how all its components are intertwined and play together
- Inspect any individual processes regarding its CPU usage, IPC, priorities, resource contentions
- Make the right decisions re performance improvement and optimization strategies

Q&A

Malte Mundt
Senior Field Application Engineer
mmundt@blackberry.com

BlackBerry | QNX

Thank you!