Linux & Windows Perf Analysis using ~~WPA~~

Ivan Berg ivberg@microsoft.com
Tristan Gibeau Tristan.Gibeau@microsoft.com

/w Nicolas De Carli De.Nicolas@microsoft.com

Microsoft -> Cloud + AI -> COSINE (Core OS Group)

Agenda

- History
  - Windows: ETW, XPerf, WPA
- Why? What? How?
  - Environment: Windows, Linux, Internet of Things
  - Tracing Types: LTTng, Linux Logs, ETL
- WPA SDK
- Demo
- Final Thoughts
  - Open Source, Community & Future

Short History/Context

Event Tracing for Windows (ETW) – since Windows 2000

- Kernel & User Instrumentation platform for Windows
- Built into the kernel. High performance (thousands events/s)
- Realtime & memory/file backed tracing
- Global & In Process Capture

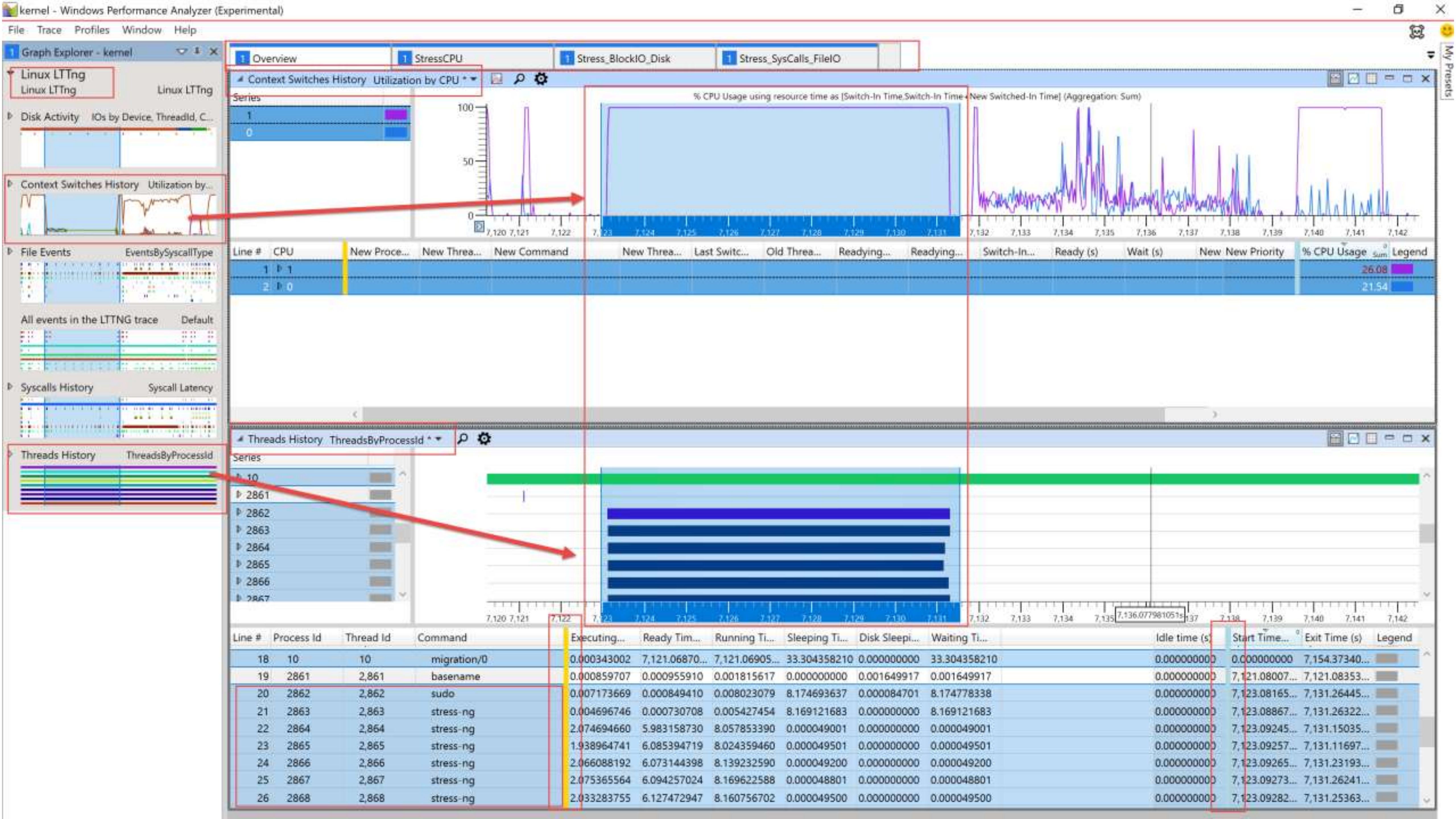
XPerf/Windows Performance Analyzer (WPA)

- Excellent support for full system tracing and full analysis on another box (offline)
- Processes Event Trace Log (ETL) -> Rebuild Kernel/User State (Correlation)
- Commandline -> Simple Tables -> Timeline Graph Views

Why? What? How?

- WPA is a popular free tool used outside/inside Microsoft
- We already have a large WPA perf analyst community.
- We want to leverage that tooling expertise when looking into Linux and Azure performance.

- ETW very successful in Windows at solving many hard perf problems
- Existing Linux Tooling & GUIs – Trace Compass, BabelTrace
- How did we arrive at system-level tracing / offline?
- “No problem can hide”


kernel - Windows Performance Analyzer (Experimental)
File Trace Profiles Window Help
Graph Explorer - kernel
Linux LTTng
Linux LTTng
Linux LTTng
Disk Activity
IOs by Device, ThreadId, C...
Context Switches History
Utilization by...
File Events
EventsBySyscallType
All events in the LTTNG trace
Default
Syscalls History
Syscall Latency
Threads History
ThreadsByProcessId
Overview
StressCPU
Stress_BlockIO_Disk
Stress_SysCalls_FileIO
Context Switches History
Utilization by CPU *
Series
% CPU Usage using resource time as [Switch-In Time,Switch-In Time+New Switched-In Time] (Aggregation: Sum)
Line #
CPU
New Proce...
New Threa...
New Command
New Threa...
Last Switc...
Old Threa...
Readying...
Readying...
Switch-In...
Ready (s)
Wait (s)
New
New Priority
% CPU Usage
Legend
1 1 26.08
2 0 21.54
Threads History
ThreadsByProcessId *
Series
10
2861
2862
2863
2864
2865
2866
2867
7,136.0779810515
Line # | Process Id | Thread Id | Command | Executing... | Ready Tim... | Running Ti... | Sleeping Ti... | Disk Sleepi... | Waiting Ti... | Idle time (s) | Start Time... | Exit Time (s) | Legend
18 | 10 | 10 | migration/0 | 0.000343002 | 7,121.06870... | 7,121.06905... | 33.304358210 | 0.000000000 | 33.304358210 | 0.000000000 | 0.000000000 | 7,154.37340...
19 | 2861 | 2,861 | basename | 0.000859707 | 0.000955910 | 0.001815617 | 0.000000000 | 0.001649917 | 0.001649917 | 0.000000000 | 7,121.08007... | 7,121.08353...
20 | 2862 | 2,862 | sudo | 0.007173669 | 0.000849410 | 0.008023079 | 8.174693637 | 0.000084701 | 8.174778338 | 0.000000000 | 7,123.08165... | 7,131.26445...
21 | 2863 | 2,863 | stress-ng | 0.004696746 | 0.000730708 | 0.005427454 | 8.169121683 | 0.000000000 | 8.169121683 | 0.000000000 | 7,123.08867... | 7,131.26322...
22 | 2864 | 2,864 | stress-ng | 2.074694660 | 5.983158730 | 8.057853390 | 0.000049001 | 0.000000000 | 0.000049001 | 0.000000000 | 7,123.09245... | 7,131.15035...
23 | 2865 | 2,865 | stress-ng | 1.938964741 | 6.085394719 | 8.024359460 | 0.000049501 | 0.000000000 | 0.000049501 | 0.000000000 | 7,123.09257... | 7,131.11697...
24 | 2866 | 2,866 | stress-ng | 2.066088192 | 6.073144398 | 8.139232590 | 0.000049200 | 0.000000000 | 0.000049200 | 0.000000000 | 7,123.09265... | 7,131.23193...
25 | 2867 | 2,867 | stress-ng | 2.075365564 | 6.094257024 | 8.169622588 | 0.000048801 | 0.000000000 | 0.000048801 | 0.000000000 | 7,123.09273... | 7,131.26241...
26 | 2868 | 2,868 | stress-ng | 2.033283755 | 6.127472947 | 8.160756702 | 0.000049500 | 0.000000000 | 0.000049500 | 0.000000000 | 7,123.09282... | 7,131.25363...
My Presets

Linux Tracing Overview

Many diverse Linux Tracing Tools

They all have their use and purpose

Tooling Use-Case: Offline / Online ?

- **Online** – meant to be largely used in real-time on the box
- **Offline** – meant to be largely used to record data, and then optionally analyzed “offline” on another box with an analysis toolset

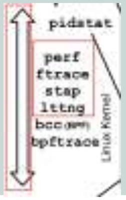
System-level or Targeted ?

- **Targeted** – Looks at one sub-system (File System, SysCalls, Sockets, etc)
- **System** – Can be targeted but captures across a wide variety of subsystems

LTTng use-case works well for an offline, system-level tracing

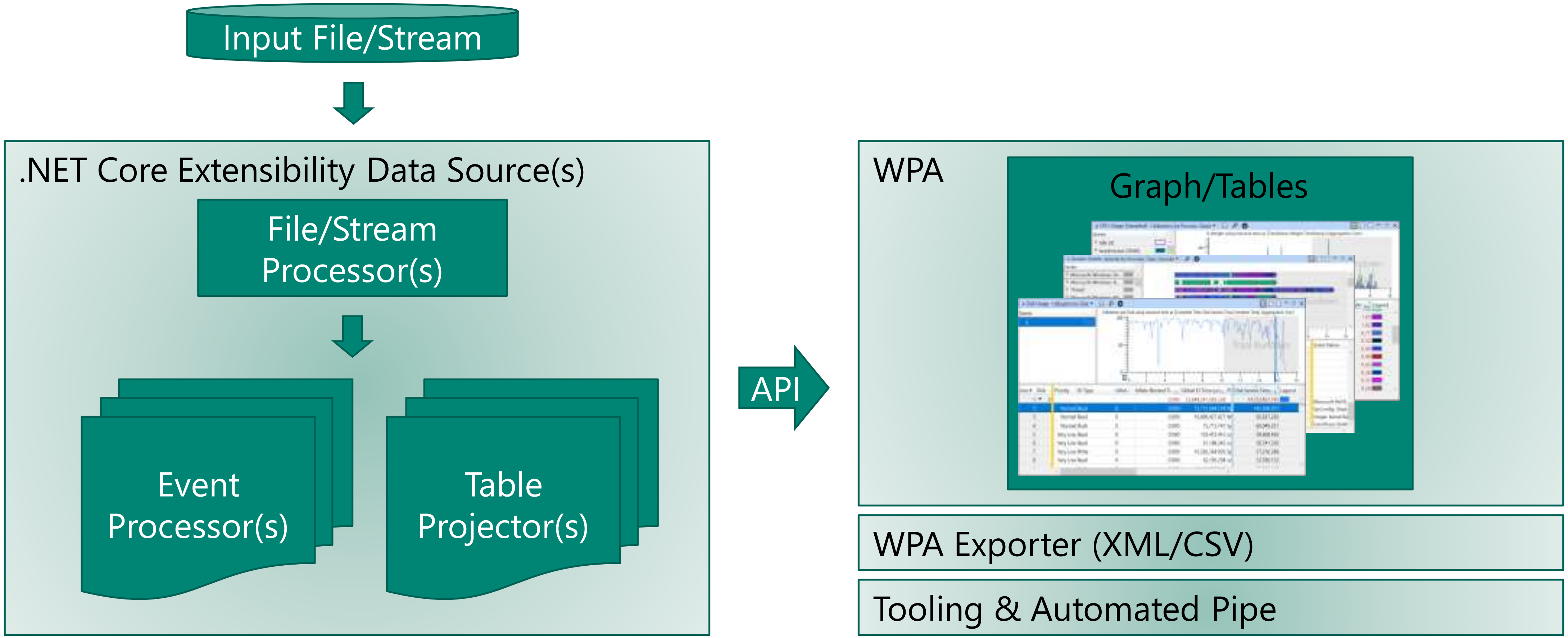
- Scales well for large scale data collection and analysis
- **Challenge: Comprehensive - a large amount of data is collected. You need good tooling to analyze and sift through data.**

Perf & Tracing Categories

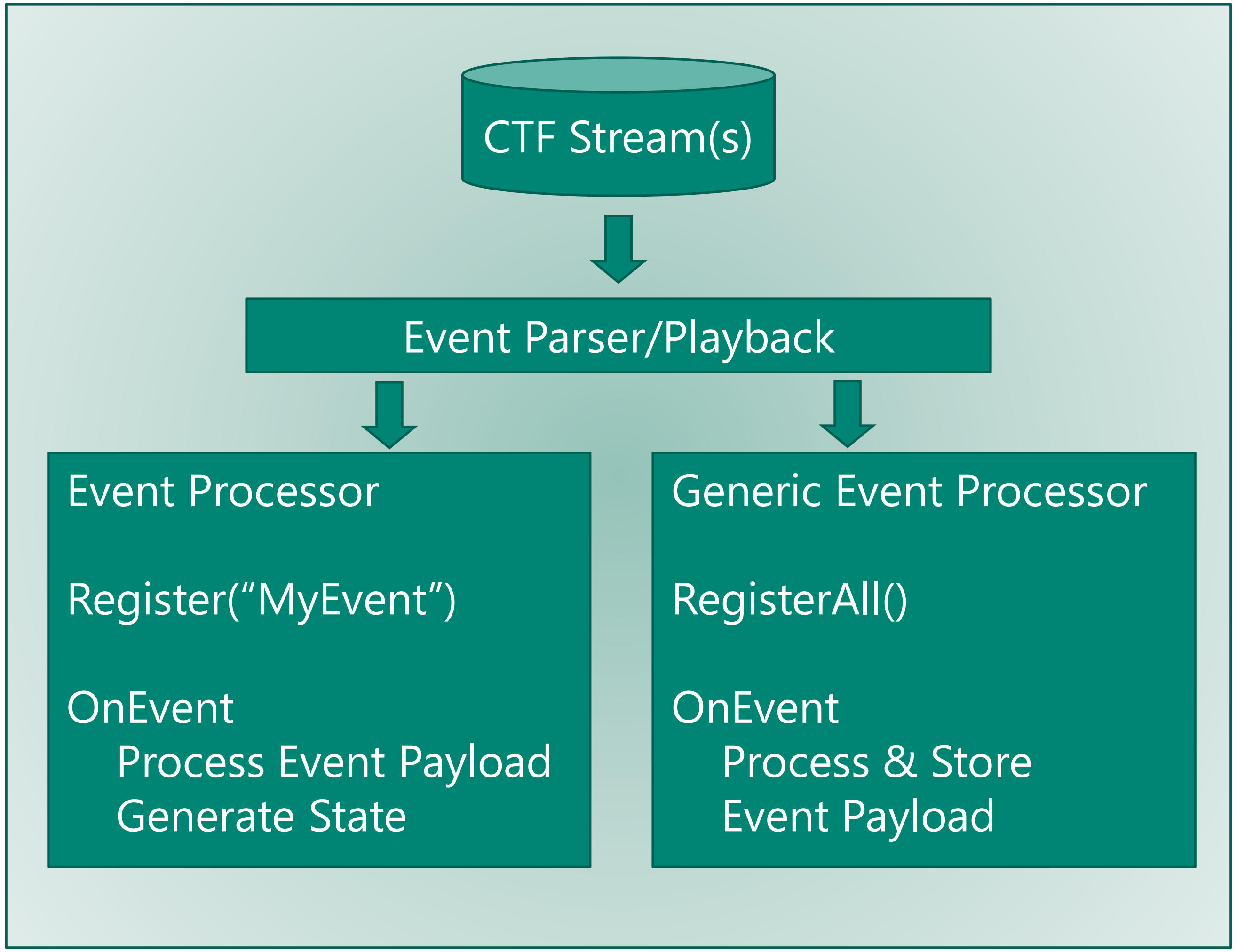
	Linux		Windows
Perf Tools	Strace, netstat, etc		Task Mgr, PerfMon, Resource Monitor, SysInternals Suite
Custom System Observability	eBPF (4.4 kernel, Ubuntu 16.04+)		ETW, Dtrace (Win 10 18342+)
Offline system-level tracing	LTTng, perf		ETW, XPerf/WPA

Linux Images Credit: Brendan Gregg

Processing SDK Overview



Common Trace Format Data Source



Plugin Sample Source

```
// Process Log to data structure
public override Task ProcessAsync(
       ISourceDataProcessor<LogEntry> dataProcessor,
       ILogger logger,
       IProgress<int> progress,
       CancellationToken cancellationToken)
{
foreach (var path in this.filePaths)
{
    while ((line = file.ReadLine()) != null)
    {
        var entry = new LogEntry();
        // Process log ....
        dataProcessor.ProcessDataElement(entry);
    }
}
this.timeInterval = new DataSourceInfo(0,
offsetEndTimestamp.ToNanoseconds, fileStartTime);
}
```

```
// GUI Table Configuration
new TableDescriptor (GUID, Name, Desc);
new ColumnConfiguration (new ColumnMetadata (GUID,
Name, Desc), new UIHints (…))

void Build (ITableBuilder tableBuilder,
            IDataExtensionRetrieval data)
{
    var timeData = data.QueryOutput<TimeData>();
    var timestampProj = timeData.Compose(
                            x => x.timestamp);

    var config = new TableConfiguration("Default"){…}

    tableBuilder.AddTableConfiguration (config)
                .AddColumn (TimeColumn, timeProj);
}
```

What Plugins are currently supported?

Linux
• LTTng (system-level tracing) • Dmesg • Cloud-Init (part of Azure VM Provisioning) • WaAgent (part of Azure VM Provisioning)

LTTng Linux Kernel WPA Plugin

- Analyzes “offline” events recorded during a tracing session of LTTng

- The plugin will parse and provide information about the following topics:
  - Threads and Processes
  - Context Switches / CPU Usage
  - Syscalls
  - File related events
  - Block IO / Disk Activity
  - Diagnostic Messages

- **Performs secondary processing and heuristics that correlate and enrich data**

- Utilize **rich graphing** and other support in **WPA** that empowers the data

Demo

Ivan Berg

How to record an LTTng trace

1. Install the tracing software:

```
$ sudo apt-get install lttng-tools lttng-modules-dkms liblttng-ust-dev
```

2. Create a session:

```
$ sudo lttng create my-kernel-session --output=lttng-kernel-trace
```

3. Add the desired events to be recorded:

```
$ sudo lttng enable-event --kernel
block_rq_complete,block_rq_insert,block_rq_issue,printk_console,sched_wak*,sched_switc
h,sched_process_fork,sched_process_exit,sched_process_exec,lttng_statedump*
$ sudo lttng enable-event --kernel --syscall --all
```

4. Optionally, add context fields to the channel:

```
$ sudo lttng add-context --kernel --channel=channel0 --type=tid
$ sudo lttng add-context --kernel --channel=channel0 --type=pid
$ sudo lttng add-context --kernel --channel=channel0 --type=procname
```

5. Start the recording:

```
$ sudo lttng start
```

6. Save the session:

```
$ sudo lttng regenerate statedump <- Better correlation / info in WPA
$ sudo lttng stop
$ sudo lttng destroy
```

More info at https://lttng.org/docs/v2.10/#doc-tracing-the-linux-kernel

Demo Setup / Context

1. Demo 1: Linux VM – Multiple Plugins /w WPA **Unified Timeline**
2. Demo 2: **Some Load Applied (Stress)**
3. Install the tracing software:

   ```
   $ sudo apt-get install stress-ng
   ```

4. Stress CPU

   ```
   $ sudo stress-ng --cpu 8 --timeout 8 --metrics-brief
   ```

5. Stress Block IO Device / Disk

   ```
   $ sudo stress-ng --hdd 5 --hdd-ops 50000
   ```

6. Stress Filesystem and Syscalls

   ```
   $ sudo stress-ng --sequential 1 --class filesystem -t 1s --times --timeout 1s
   $ Ctrl-C After 1s
   ```

Demo Contents

- Watch the final video of the talk
  - OR
- See Appendix Slides for details on the WPA LTTng Views
- OR
- Two pre-recorded demo videos
  - WPA Unified Timeline
  - Stress – Some load applied

Final Thoughts

- We are open to ideas and comments from the community
- We will be open-sourcing LTTng / Linux plugins
  http://aka.ms/TracingSummit2019
- This is our first early pass at Linux Tracing tooling for WPA
  - WPA tool is stable. LTTng Plugins are "Beta" quality
  - Heuristics may need work or be wrong. Let us know or contribute!
- We would love for other scenarios / logs to work as well. E.g.
  - Offline CPU Sampling : cpu-clock via perf->CTF or even LTTng ?
  - Stacks?
  - Memory?
  - Windows Subsystem for Linux 2 (WSL2) LTTng kernel module ?
  - Systemd "Log" PlugIn?

Question & Answer

Microsoft

Ivan Berg ivberg@microsoft.com

Tristan Gibeau Tristan.Gibeau@microsoft.com

/w Nicolas De Carli De.Nicolas@microsoft.com

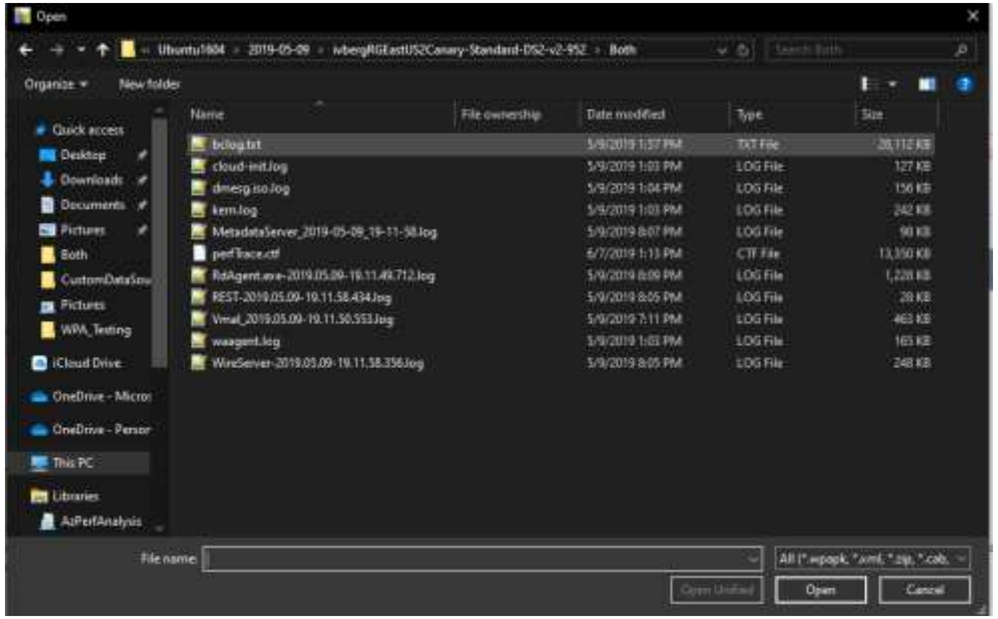
http://aka.ms/TracingSummit2019

Opening a LTTng Common Trace Format (CTF)

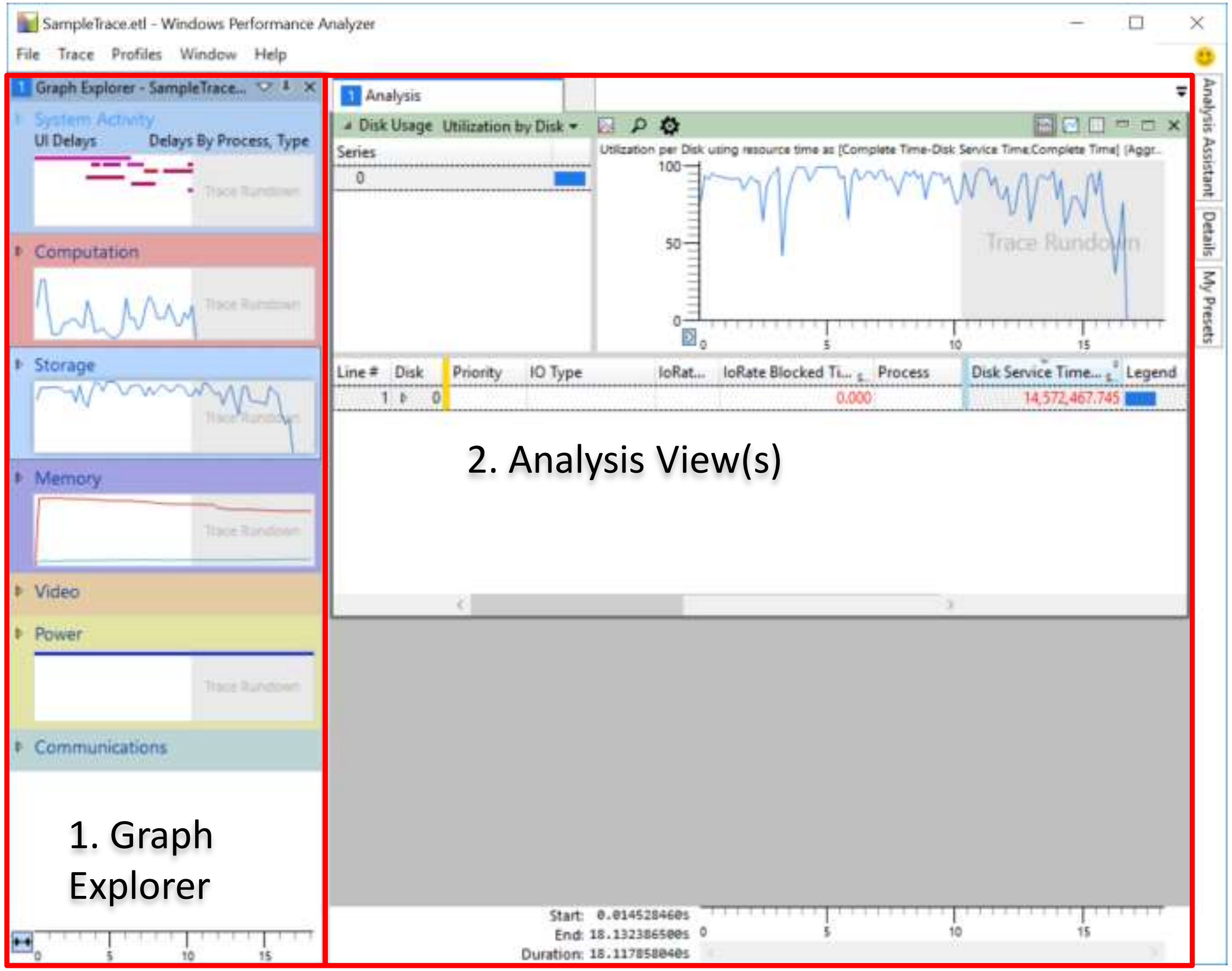
WPA – Two ways to load LTTng CTF

- Just LTTng CTF Trace –
  - File -> Open Folder
- WPA Unified Open (everything in the same session with single timeline)
  - Workaround:
    1) Zip LTTng folder and rename to .ctf extension
    2) Copy all files to a single folder including .ctf file
    3) File -> Open

Unified Demo

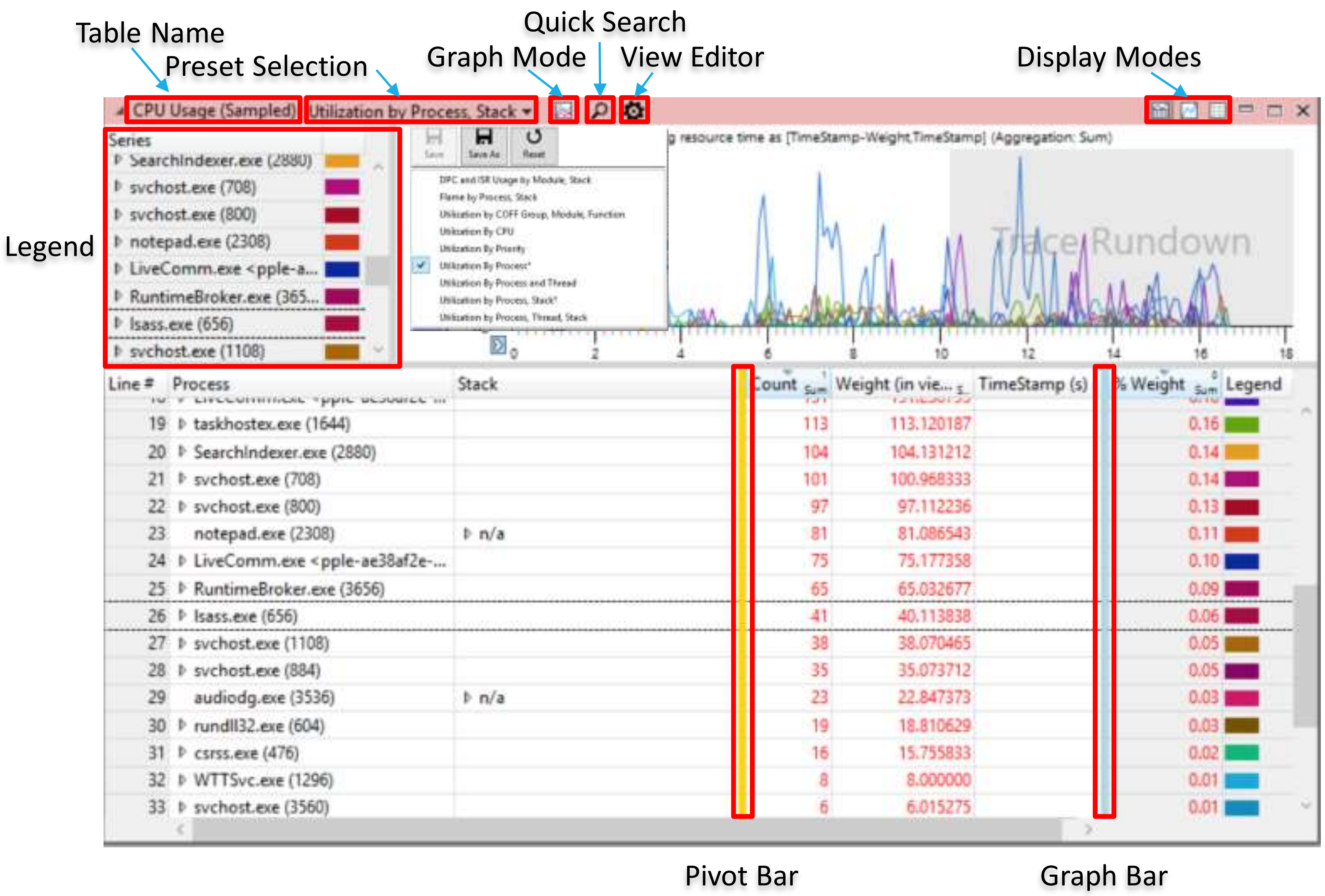


WPA Layout



1. Graph Explorer shows KPIs (Key Performance Indicator)
2. Drag and Drop from Graph Explorer to Analysis View

Table Layout



Preset Selection

- Switch & Save Presets

Graph Mode

- Line
- Stacked Line/Bar
- Flame

Quick Search

- Search Across Columns in Table

Display Modes

- Graph Only
- Table Only
- Split

Pivot Bar (Gold)

- Group Similar Data

Graph Bar (Blue)

- Graphed Data

LTTng is an open source tracing framework for Linux

It provides Kernel modules to trace the Linux kernel

A tracing session has a set of channels, which are a stream of events

Each event belongs to a certain kind, which is identified by a name and an id

An event contains a dictionary called Payload, which contains all the information related to the event

A context is provided with each event, for instance, it can contain the CPU on which the event occurred

Event Example:

Name	Id	CPU	_parent_comm (Field 1)	_parent_tid (Field 2)	_parent_pid (Field 3)	_parent_ns_inum (Field 4)	_child_comm (Field 5)	_child_tid (Field 6)	__vtids_length (Field 7)	_vtids (Field 8)	Timestamp (s)
sched_process_fork	1,169	1	systemd	517	517	4026531836	systemd	518	1	[[0] = 518]	6.100463900

We are going to present a WPA plugin that shows profiling information of the Linux kernel

Analyzes events recorded during a tracing session of LTTng

The plugin will parse and provide information about the following topics:

- Threads and Processes
- Context Switches
- Syscalls
- File related events
- Disk Activity
- Diagnostic Messages

Syscall view

Lists every syscall that occurred during the trace, specifying for each one:

- Name
- Arguments used
- Return value
- Thread Id of the caller
- Process Id of the caller
- Start Time
- End Time
- Duration

Process Id	Thread Id	Syscall Name	Arguments	Return Value	Duration (μs) Max	Start Time (s)	End Time (s)
622	622				137,479.000		
		ioctl			1.100		
			{fd: 7, cmd: 2149074240, arg: 140731002831264}	-22	1.100	7.268445600	7.268446700
			{fd: 7, cmd: 2149074240, arg: 140731002831264}	-22	0.900	7.355870700	7.355871600
			{fd: 15, cmd: 2148012658, arg: 94139984172208}	0	0.700	7.537052300	7.537053000
		fadvise64	{fd: 15, offset: 0, len: 0, advice: 1}	0	0.700	7.537043700	7.537044400
		read			56.800		
			{fd: 7, count: 4096}	22	2.700	6.719437300	6.719440000
			{fd: 7, count: 4096}	0	0.600	6.719440700	6.719441300

Syscall Name	Syscall Name Count	Duration (μs) Max	Duration (μs) Min	Duration (μs) Avg	Duration (μs) Sum
poll	3,523	33,952,690.580	0.000	110,474.643	389,202,169.150
epoll_wait	23,370	52,465,305.490	0.000	12,716.469	297,183,883.744
select	6,859	30,201,977.070	0.500	25,138.836	172,427,276.432
read	135,864	33,404,188.934	0.000	1,153.068	156,660,449.545
nanosleep	50	15,000,121.800	1,000,063.206	1,991,744.438	99,587,221.902
futex	18,070	15,000,174.100	0.299	5,338.919	96,474,279.783
recvfrom	457	30,227,588.800	0.500	66,147.092	30,229,221.486
wait4	1,974	5,717,513.700	0.300	14,394.160	28,414,073.773
ppoll	1,613	5,004,732.200	0.600	7,825.012	12,621,744.658
write	20,656	781,805.300	0.000	374.429	7,734,217.406

Threads View

Contains an entry for every thread that was alive during any moment of the tracing session.

It has 14 columns, 5 displaying attributes about the thread and 9 specifying how much time the thread spent in different states.

The attributes being shown are:

- Thread Id
- Process Id
- Command (Executable name)
- Start Time
- Exit Time

Threads View

The states a thread can be in are declared in sched.h.

The kernel defines a user-friendly translation in array.c, as follows:

State	Translation
TASK_RUNNING	R (running)
TASK_INTERRUPTIBLE	S (sleeping)
TASK_UNINTERRUPTIBLE	D (disk sleep)
__TASK_STOPPED	T (stopped)
__TASK_TRACED	t (tracing stop)
TASK_PARKED	P (parked)
TASK_DEAD	(Varies depending on the thread's exit state)
TASK_WAKEKILL	R (running)
TASK_WAKING	R (running)
TASK_NOLOAD	R (running)
TASK_NEW	R (running)
TASK_STATE_MAX	R (running)
TASK_KILLABLE	D (disk sleep)
TASK_STOPPED	T (stopped)
TASK_TRACED	t (tracing stop)
TASK_IDLE	I (idle)

Threads View

A column for every of the following translations is presented:

- Running Time
- Sleeping Time
- Disk Sleeping Time
- Stopped Time
- Parked Time
- Idle Time

Each one shows the time spent in any state of such translation

Additionally, the following columns are provided:

- Executing Time – Total time spent executing on any CPU
- Ready Time – The thread was able to run but not scheduled on any CPU
- Waiting Time – Sum of Sleeping Time and Disk Sleeping Time

Threads View

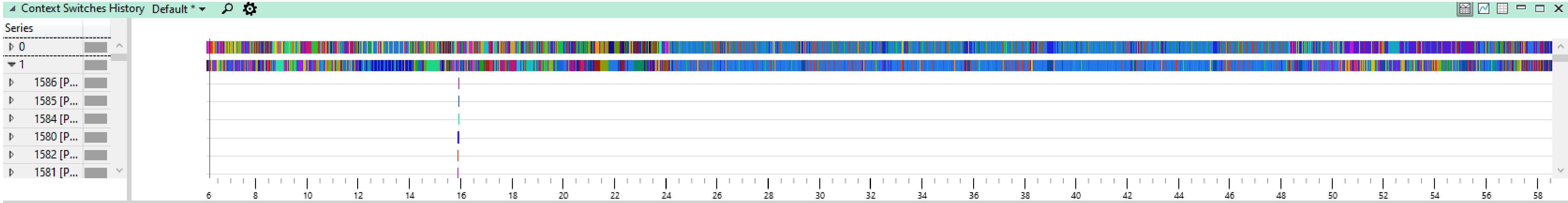
Process Id	Thread Id	Command	Executing Time (s) Sum	Ready Time (s) Avg	Running Time (s) Avg	Sleeping Time (s) Avg	Disk Sleeping Time (s) Avg	Waiting Time (s) Avg	Idle time (s) Avg	Parked Time (s) Avg	Stopped Time (s) Avg	Start Time (s) Min	Exit Time (s) Max
1394			0.236232334	0.105162237	0.164220320	42.814159944	0.040428291	42.854588235	0.000000000	0.000000000	0.000000000	15.257695400	58.545642681
	1,394	rsyslogd	0.004665400	0.006181700	0.010847100	43.118727881	0.158372300	43.277100181	0.000000000	0.000000000	0.000000000	15.257695400	58.545642681
	1,500	rsyslogd	0.006743999	0.006090498	0.012834497	42.915036284	0.001232500	42.916268784	0.000000000	0.000000000	0.000000000	15.616539400	58.545642681
	1,501	rsyslogd	0.083239590	0.123150078	0.206389668	42.721260948	0.001402065	42.722663013	0.000000000	0.000000000	0.000000000	15.616590000	58.545642681
	1,499	rsyslogd	0.141583345	0.285226672	0.426810017	42.501614664	0.000706300	42.502320964	0.000000000	0.000000000	0.000000000	15.616511700	58.545642681
1520	1,520	mdadm	0.000189200	0.000111700	0.000300900	42.794402081	0.000000000	42.794402081	0.000000000	0.000000000	0.000000000	15.750939700	58.545642681
1446	1,446	systemd-logind	0.019282654	0.036450865	0.055733519	42.774932294	0.252495568	43.027427862	0.000000000	0.000000000	0.000000000	15.462481300	58.545642681
1423	1,423	dbus-daemon	0.142841775	0.267868817	0.410710592	42.699565189	0.126150700	42.825715889	0.000000000	0.000000000	0.000000000	15.309216200	58.545642681
1556	1,556	agetty	0.005882500	0.007467300	0.013349800	42.699145981	0.013679400	42.712825381	0.000000000	0.000000000	0.000000000	15.819467500	58.545642681
1526	1,526	apport	0.003079900	0.019488700	0.022568600	42.690403381	0.061606600	42.752009981	0.000000000	0.000000000	0.000000000	15.771064100	58.545642681
1555	1,555	agetty	0.013042600	0.031357100	0.044399700	42.682482881	0.000076900	42.682559781	0.000000000	0.000000000	0.000000000	15.818683200	58.545642681
1572	1,572	irqbalance	0.004161177	0.005764862	0.009926039	42.681844042	0.000000000	42.681844042	0.000000000	0.000000000	0.000000000	15.853872600	58.545642681
1617	1,617	ondemand	0.002430600	0.000142500	0.002573100	42.601215581	0.000000000	42.601215581	0.000000000	0.000000000	0.000000000	15.941854000	58.545642681
1628	1,628	sleep	0.000481700	0.000095700	0.000577400	42.581201181	0.000476300	42.581677481	0.000000000	0.000000000	0.000000000	15.963387800	58.545642681
1419			0.531520200	0.033839822	0.092897622	40.978836092	0.306572211	41.285408303	0.000000000	0.000000000	0.000000000	15.298814200	58.545642681
	1,678	snapd	0.000136000	0.000112000	0.000248000	41.876888881	0.000000000	41.876888881	0.000000000	0.000000000	0.000000000	16.668505800	58.545642681
	1,676	snapd	0.012594700	0.110298200	0.122892900	41.762651281	0.000000000	41.762651281	0.000000000	0.000000000	0.000000000	16.660098500	58.545642681
	1,679	snapd	0.007774600	0.103195300	0.110969900	41.413983881	0.352126600	41.766110481	0.000000000	0.000000000	0.000000000	16.668562300	58.545642681
	1,419	snapd	0.025907500	0.022427600	0.048335100	41.160888181	2.037605200	43.198493381	0.000000000	0.000000000	0.000000000	15.298814200	58.545642681
	1,709	snapd	0.011488000	0.004761500	0.016249500	41.051434581	0.000000000	41.051434581	0.000000000	0.000000000	0.000000000	17.477958600	58.545642681
	1,701	snapd	0.005328700	0.005428800	0.010757500	41.027079081	0.075262400	41.102341481	0.000000000	0.000000000	0.000000000	17.432543700	58.545642681
	1,799	snapd	0.026583500	0.003313900	0.029897400	40.308179181	0.000000000	40.308179181	0.000000000	0.000000000	0.000000000	18.207566100	58.545642681
	1,784	snapd	0.332849400	0.032305200	0.365154600	40.264321481	0.031776800	40.296098281	0.000000000	0.000000000	0.000000000	17.884389800	58.545642681
	1,800	snapd	0.108857800	0.022715900	0.131573700	39.944098281	0.262378900	40.206477181	0.000000000	0.000000000	0.000000000	18.207591800	58.545642681
1767	1,767	sshd	0.007988693	0.013130622	0.021119315	40.694523066	0.000003800	40.694526866	0.000000000	0.000000000	0.000000000	17.829996500	58.545642681

Context Switch View

- Lists every context switch that occurred during the tracing session
  - Similar to the “Timeline by CPU” view, under the “CPU Usage (Precise)” category displayed in WPA when analyzing WPR traces
- Has 17 columns, detailed as follows:

Column Name	Description
CPU	CPU on which the context switch occurred
New Process Id	Process Id of the thread that is being switched in
New Thread Id	Thread Id of the thread that is being switched in
New Command	Command Id of the thread that is being switched in
Old Process Id	Process Id of the thread that is being switched out
Old Thread Id	Thread Id of the thread that is being switched out
Old Command	Command Id of the thread that is being switched out
Last Switch Out Time	Last time the new thread was switched out from a CPU
New Thread’s Previous State	State of the new thread before being ready for execution
Readying Process Id	Process Id of the thread that caused the new thread to be ready
Readying Thread Id	Thread Id of the thread that caused the new thread to be ready
Ready	Time the new thread spent ready for execution before it was switched in
Wait	Time between the new thread’s last switch out time and when it became ready
New Switched-In Time	Time the new thread spent executing immediately after it was switched in
New Priority	Execution priority of the thread that is being switched in
Switch-In Time	Time when the context switch happened
Next Switch-Out Time	Time when the new thread will be switched out

Context Switch View

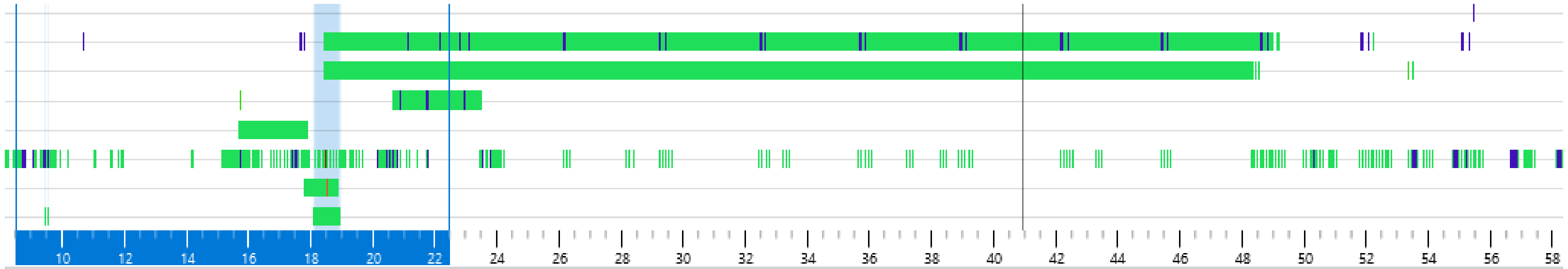


Line #	CPU	New Process Id	New Thread Id	New Command	Old Thread Id	Old Process Id	New Thread's Previous State	Last Switch-Out Time (s)	Old Command	Ready (µs)	Wait (µs)	New Switched-In Time (ms)	New Priority	Switch-In T...	Next Switch-Out Time...	Legend
1	▷ 0									15,893,795.200		255.993468				
2	▼ 1									15,888,246.700		422.429200				
3		1586 [Probably]	1,586	nscd	609	609		0.000000000	systemd-udevd	15,888,246.700	0.000	0.021100	20	15.888246700	15.888267800	
4		1585 [Probably]	1,585	nscd	1,553	1553		0.000000000	systemd-udevd	15,887,019.900	0.000	0.019300	20	15.887019900	15.887039200	
5		1584 [Probably]	1,584	nscd	1,550	1550		0.000000000	systemd-udevd	15,885,393.200	0.000	0.025200	20	15.885393200	15.885418400	
6		1580 [Probably]	▷ 1,580							15,875,482.100		0.568400				
7		1582 [Probably]	1,582	perfkill	1,583	1583		0.000000000	xargs	15,873,369.100	0.000	0.069100	20	15.873369100	15.873438200	
8		1581 [Probably]	1,581	perfkill	666	666 [Probably]		0.000000000	kworker/1:3	15,872,721.700	0.000	0.528300	20	15.872721700	15.873250000	
9		1583	▼ 1,583							15,872,290.700		0.405500				
10				perfkill	555	555		0.000000000	systemd-journal	15,872,290.700	0.000	0.405500	20	15.872290700	15.872696200	
11				xargs	1,602	1602	R (running)	15.928921500	systemctl	8,024.100	0.000	0.005200	20	15.936945600	15.936950800	
12				xargs	1,501	1394	R (running)	15.936950800	rs:main Q:Reg	665.000	0.000	0.081400	20	15.937615800	15.937697200	
13				xargs	1,581	1583	R (running)	15.872696200	grep	553.800	0.000	0.119100	20	15.873250000	15.873369100	
14		1577 [Probably]	1,577	kworker/u4:2	16	16 [Probably]		0.000000000	ksoftirqd/1	15,872,179.400	0.000	0.009600	20	15.872179400	15.872189000	
15		1573 [Probably]	▷ 1,573							15,862,933.300		0.794200				
16		776 [Probably]	776	sh	710	710		0.000000000	console-setup-t	7,724,412.500	0.000	0.706100	20	7.724412500	7.725118600	
17		769	▷ 769							7,712,904.400		0.413000				
18		770	▷ 770							7,696,782.100		0.657800				
19		764	▷ 764							7,688,820.200		0.586000				
20		760	▷ 760							7,682,013.500		0.640600				
21		759	▷ 759							7,681,426.000		0.474500				
22		756	▷ 756							7,660,096.400		0.157200				

File Events View

- Lists the following file-related syscalls:
  - create, fallocate, ftruncate, lseek, memfd_create, mknod, mknodat, name_to_handle_at, open, open_by_handle, openat, pread, preadv, pwrite, pwritev, read, readv, rename, renameat, renameat2, sendfile, truncate, write, writev

- The following information is specified for each entry:
  - Name of the syscall
  - Thread Id of the caller
  - Process Id of the caller
  - Size of the operation
  - File involved
  - Duration
  - Start Time
  - End Time



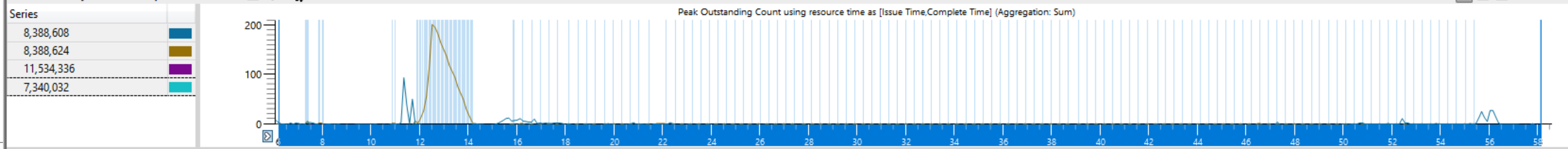
File	Event	Thread Id	Process Id	Size (KB)	Duration (s)	Start Time (s)	End Time (s)	Legend
/dev/console				16,143.301	39.753729781			
/usr/lib/python3.5/__pycache__/_strptime.cpython-35.pyc				15.557	33.404311330			
	read			15.494	33.404293630			
	open	2,301	2301	0.000	0.000010100	55.436985820	55.436995920	
	write			0.063	0.000007000			
	lseek	2,301	2301	0.000	0.000000600	55.437012019	55.437012619	
/var/lib/waagent/Incarnation				2.738	30.472072694			
/var/lib/waagent/history				4.873	29.935349033			

Disk Activity View

- Lists every blocking I/O request sent to a disk, specifying the following for each one:
  - Device Id
  - File Involved in the operation
  - Thread Id of the thread which made the request
  - Process Id of the thread which made the request
  - Disk’s Sector Number where the data involved in the operation resides
  - Disk’s Offset of the data involved in the operation
  - Size of the operation
  - IO Time
  - Error number of the operation
  - Request’s Insert Time
  - Request’s Issue Time
  - Request’s Complete Time

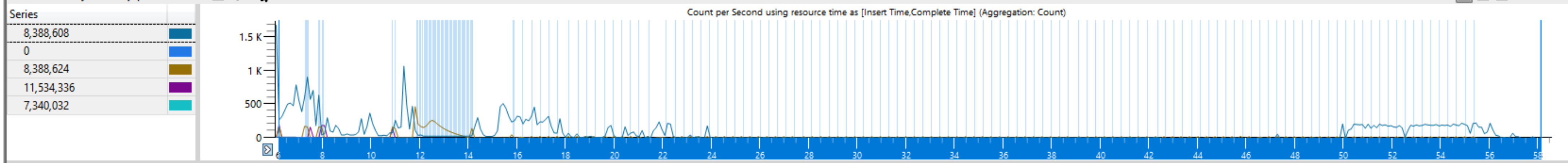
Many useful view arrangements are presented with the

Disk Activity OutStandingIOsOverTime



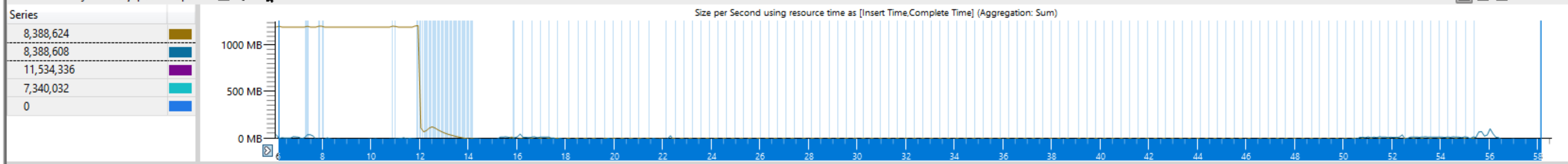
Line #	Device Id	Sector Number	Disk Offset (B)	Size (MB) Sum	Size (MB) Max	Insert Time (s)	Insert Time...	Issue Time (s)	Issue Time...	Complete Tim...	Complete...	IO Time (ms) Max	IO Time (ms) Avg	IO Time (ms)	Count Sum	Legend
1	8,388,608			294.023	0.500							241.542499	9.059533	33,520.274884	3,700	
2	8,388,624			307.207	0.500							1,583.143400	225.829339	188,115.839434	833	
3	11,534,336			0.416	0.004							9.830300	1.438568	153.926800	107	
4	7,340,032	0	0	0.004	0.004	17.432340100	2019/05/09 2...	17.432341100	2019/05/09 2...	17.432355400	2019/05/09 2...	0.015300	0.015300	0.015300	1	

Disk Activity Throughput - IOPs



Line #	Device Id	Sector Number	Disk Offset (B)	Size (MB) Sum	Size (MB) Max	Insert Time (s)	Insert Time...	Issue Time (s)	Issue Time...	Complete Tim...	Complete...	IO Time (ms) Max	IO Time (ms) Avg	IO Time (ms)	Count Count	Legend
1	8,388,608			316.325	2.238							241.542499	6.224749	33,520.274884	5,385	
2	0			0.001	0.000							0.000000	0.000000	0.000000	211	
3	8,388,624			14,700.921	2,048.000							1,583.143400	185.153385	188,115.839434	1,016	
4	11,534,336			0.483	0.004							9.830300	0.993076	153.926800	155	
5	7,340,032			0.008	0.004							0.015300	0.003060	0.015300	5	

Disk Activity Throughput - MBps



Line #	Device Id	Sector Number	Disk Offset (B)	Size (MB) Max	Insert Time (s)	Insert Time...	Issue Time (s)	Issue Time...	Complete Tim...	Complete...	IO Time (ms) Max	IO Time (ms) Avg	IO Time (ms) Sum	Count	Size (MB) Sum	Legend
1	8,388,624			2,048.000							1,583.143400	185.153385	188,115.839434	1	14,700.921	
2	8,388,608			2.238							241.542499	6.224749	33,520.274884	5	316.325	
3	11,534,336			0.004							9.830300	0.993076	153.926800		0.483	
4	7,340,032			0.004							0.015300	0.003060	0.015300		0.008	
5	0			0.000							0.000000	0.000000	0.000000		0.001	

Diagnostic Messages View

- Lists all the diagnostic messages of the kernel that were logged during the tracing session

- Alongside the message, a timestamp of when it was created is displayed

Line #	Message	Timestamp (s)
1	Loading iSCSI transport class v2.0-870.	6.406671000
2	iscsi: registered transport (tcp)	6.434802000
3	EXT4-fs (sda1): re-mounted. Opts: discard	6.437460000
4	iscsi: registered transport (iser)	6.477262000
5	random: crng init done	6.573536000
6	random: 7 urandom warning(s) missed due to ratelimiting	6.576005000
7	serial8250: too much work for irq4	15.052073000

All Events View

- Lists all the events of the trace, in a raw format.

- For each entry, the following information is provided:
  - Name of the event
  - Id of the event
  - CPU where the event occurred
  - Timestamp
  - Payload

About the current heuristics

An event is logged when a syscall starts, and a different one is logged when it ends

- There is no direct way to know which opening event belongs to each closing one
- We match them by name and the thread id related to the events, that is, the thread id of the caller
- If a thread issues a syscall of a certain kind while another one of the same type is ongoing, we have no way of knowing to which syscall the following exiting events belong to. In this case, both syscalls will be logged with duration zero because we don’t know when they ended.

If the thread id is not in the context of an event, it can be inferred by tracking context switch events

- We always know the CPU on which the event occurred. We need to check the latest context switch on that CPU to find out which thread was being executed and therefore generated the event.
- Context switches are recorded by LTTng by Pluging *`sched_switch`* events to the session

About the current heuristics

The process id of a thread can be inferred when is created by listening to fork, vfork and clone syscalls.

- If fork or vfork are called, the child utilizes its thread id as process id.
- If clone is used, a bit of one of the arguments indicates if the process id has to be inherited, or if the child's thread id should be used as process id instead.
- This heuristic is more tolerant to having multiple ongoing syscalls
  - If we spot several consecutive fork or vfork entry events, if all the corresponding exit events state that the operation was successful, although we won't know which exit event belongs to each entry event, since the child's thread id is noted in the entry event, we will be able assign the thread as process id to all the new threads
  - With clone the situation is similar, although we also must check that the bit we are interested in has the same value on every clone entry event. If that's the case, we can confidently utilize the same behavior on every new thread, whether it is to inherit the process id or utilize its thread id.
- For the processes that were running when the trace started and for those that the syscall inferring process failed, we will guess its process id is its thread id and place the "[Probably]" placeholder next to the process id.

getpid syscalls are listened to capture the process id of threads for which we are not sure of its process id

- When a process id is discovered in this way, all the threads of that process are updated

About the current heuristics

We infer the file involved in each file IO operation

- File IO syscalls have file descriptors as arguments
- We track syscalls that create or open files to know the filepath each descriptor points to
- Rename syscalls must also be tracked to update filepaths when a file is renamed
  - When we fail to parse a syscall of this kind, both possible filepaths will appear on the file column, with the placeholder “(maybe renamed to)” in between them

Tracking IO operations allows us to know the file being used when a disk activity occurs

- If a disk request is issued by a given thread, and that thread has only one ongoing file IO syscall, we infer that the file being accessed by the activity is the one involved in the syscall.
- After a successful match, we know on which device the file is on. We can use this information for future guesses.
  - If many file IO operations are ongoing when a disk request is placed, but only one is related to a file that is on the device of the request, we know that’s the file involved in the disk activity.