



# Tracealyzer for Linux

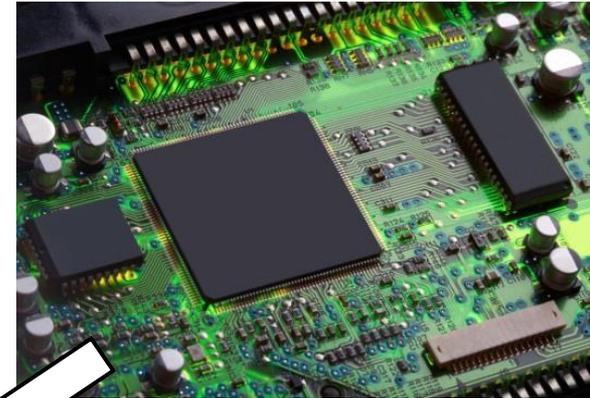
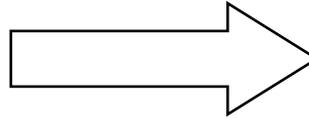
## Overview and Getting started



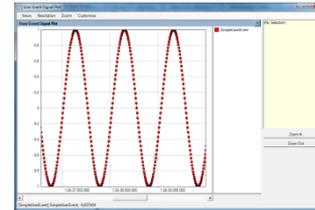
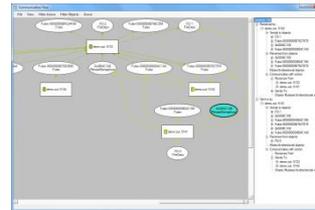
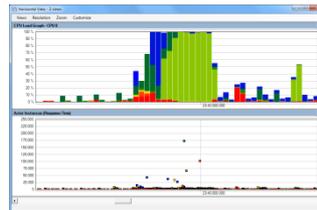
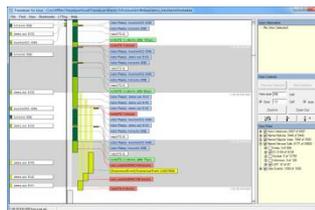


# Tracealyzer shows what's going on!

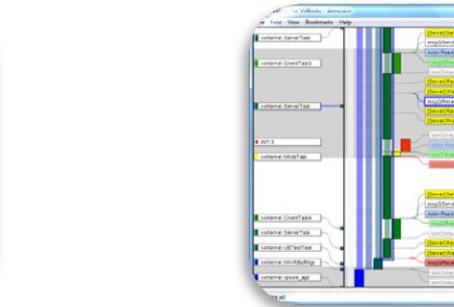
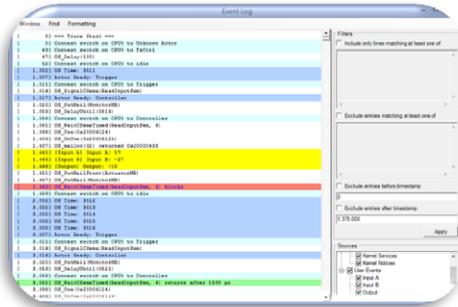
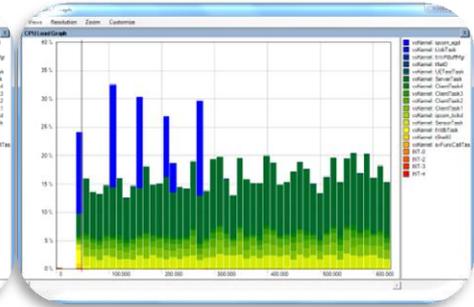
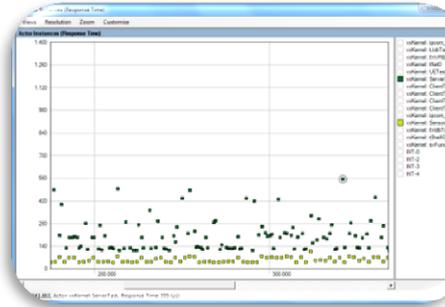
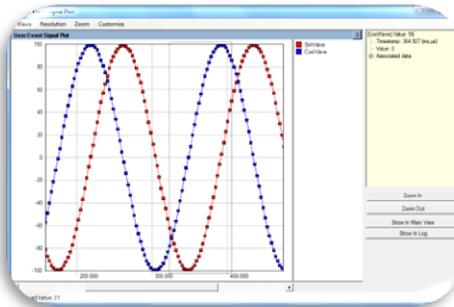
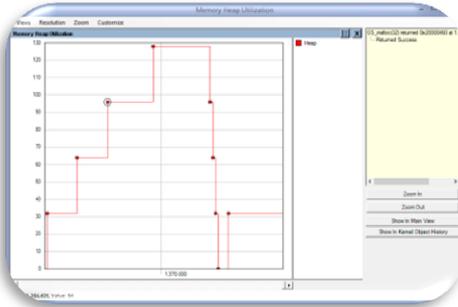
```
if (liczba_binarny[0] == 0)
    ulamek_binarny[0] = 0;
    counter = 1;
    counter_c = 1;
}
for (int i = 0; liczba != 0; i++) {
    int bit = Math.abs(liczba % 2);
    liczba = liczba / 2;
    binarna[i] = bit;
    counter_c++;
}
ulamek = Math.abs(ulamek);
for (int i = 0; ulamek != 0; i++) {
    if (ulamek % 2 == 1)
        ulamek_binarny[i] = 1;
        ulamek = ulamek / 2;
    else
        ulamek_binarny[i] = 0;
}
```



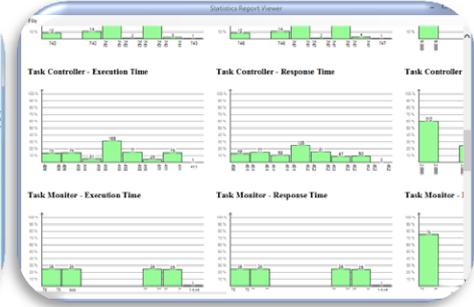
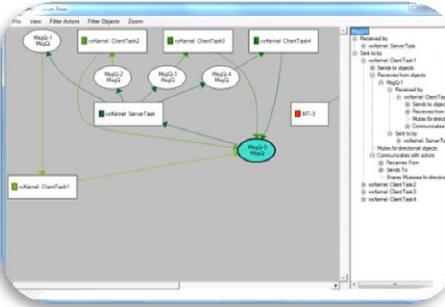
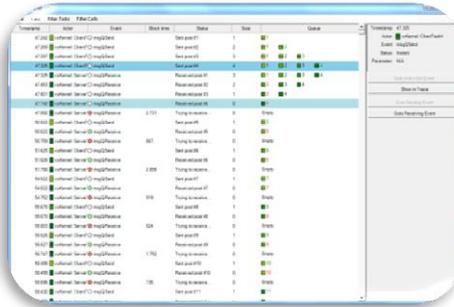
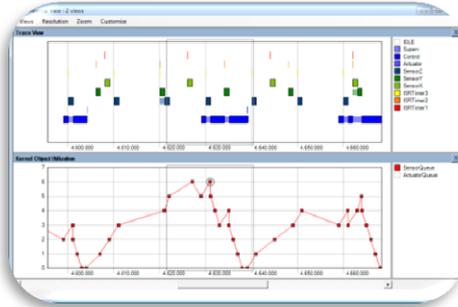
**Software-Defined Tracing**  
+ Easy to use  
+ Flexible/configurable  
+ Always applicable



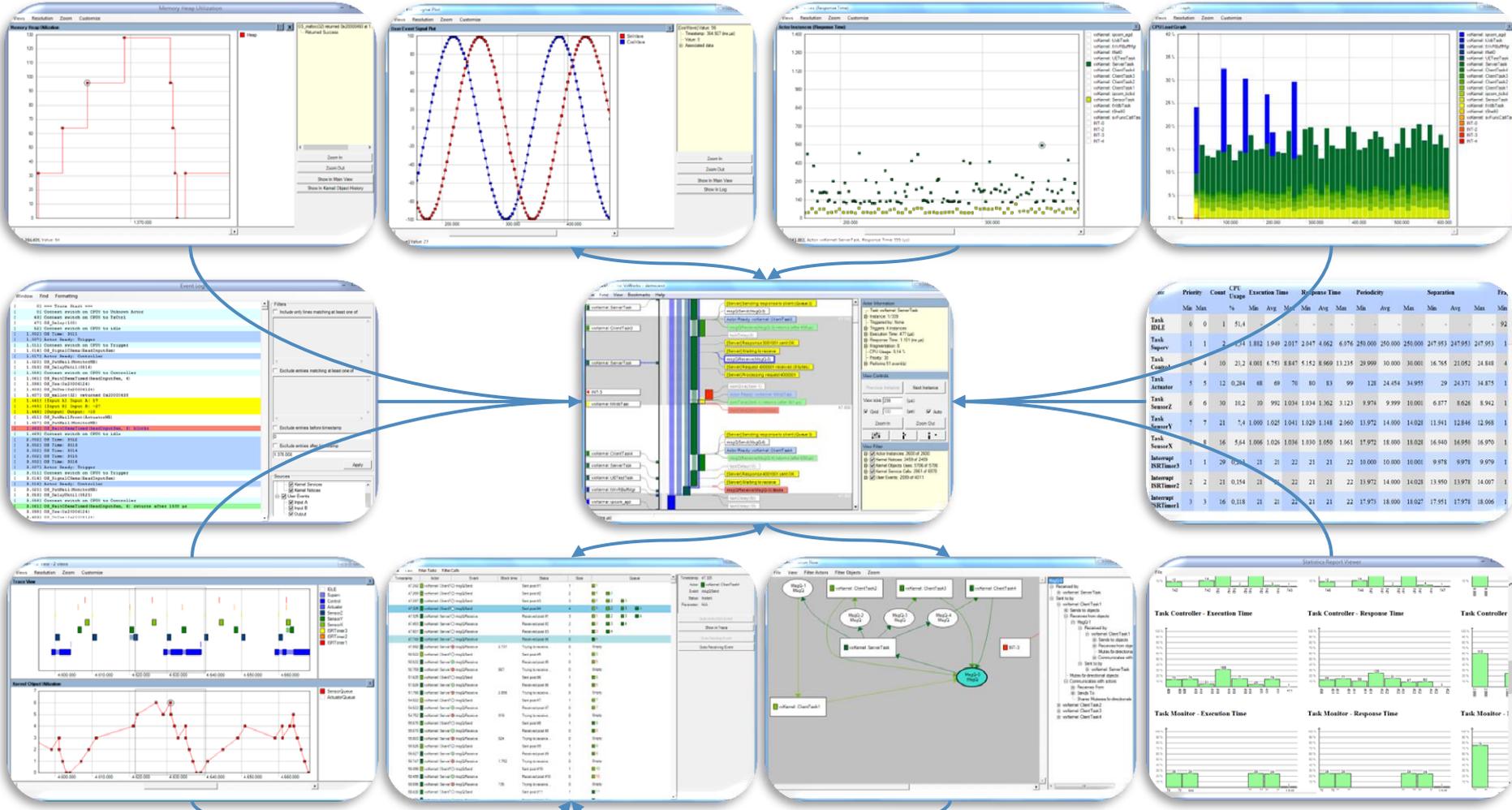
# Tracealyzer is about *Visualization*



Task	Priority	Count	CPU Usage	Execution Time	Response Time	Periodicity	Separation	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	
Task IDLE	0	0	1	51.4												92	
Task Super	1	1	2	1.34	1.812	1.948	2.017	2.047	4.062	6.076	249.000	250.000	247.953	247.953	247.953	1	
Task Control	4	4	10	21.2	1.001	6.751	8.847	5.152	8.960	13.235	29.900	10.000	30.000	16.765	21.052	24.818	4
Task Actuator	5	5	12	0.284	48	69	70	80	83	99	128	24.454	34.955	29	24.371	34.875	1
Task SensorZ	6	6	30	16.2	10	995	1.034	1.034	1.362	3.123	9.974	9.999	10.001	6.577	8.626	8.942	1
Task SensorX	7	7	21	5.4	1.000	1.020	1.041	1.029	1.148	2.360	13.972	14.000	14.020	11.941	12.846	12.968	1
Task SensorY	8	8	16	5.64	1.006	1.026	1.034	1.030	1.050	1.961	13.972	14.000	14.020	16.940	16.956	16.970	1
Interrupt ISRTime3	1	1	29	0.233	21	21	22	21	21	22	10.000	10.000	10.000	9.978	9.978	9.979	1
Interrupt ISRTime2	2	2	21	0.154	21	21	22	21	21	22	13.972	14.000	14.020	13.950	13.974	14.007	1
Interrupt ISRTime1	3	3	16	0.118	21	21	22	21	21	22	17.951	18.000	18.020	17.951	17.976	18.006	1



# Cleverly Interconnected



# Increasing Development Efficiency and Software Performance

## Debugging

Detailed real-time behavior

## Validation

Code running as intended?

## Profiling

RAM and CPU usage?

## Training

Learn the software platform

## Documentation

Visualize designs or issues

*"In less than 5 days from running the tool, we improved the performance of our graphic rendering engine by **3x!**"*  
**Terry West, CEO, Serious Integrated Inc.**

*"Tracealyzer has **doubled our development speed.** Problems that otherwise would take days to solve are obvious with this tool and just a quick fix. We use it all the time."*  
**Alex Paboutisids, Lead Firmware Engineer, Flyability.**

**SERIOUS™**



# For Leading Companies

Percepio's tracing tool allowed me to **quickly understand and solve serious multi-threading issues**, that otherwise would have taken least two weeks to analyze. I got started and solved the first issue in a single day. I strongly recommend Percepio's tracing tools."

Chaabane Malki, Embedded Systems Engineer, CGX Aero



ABB Robotics is using the first generation Tracealyzer in **all of the IRC5 robot controllers shipped since 2005**. The tool has proven its value many times in all corners of the world."  
Roger Kulläng, Global System Architect, ABB Robotics.



"In today's tough competition with time-to-market pressure constantly increasing, visualization support is natural for software developers in order to produce software of **higher quality, in shorter time and at a lower cost**. We choose Tracealyzer from Percepio."

Jörgen Appelgren, R&D Manager, Atlas Copco Rock Drills

Atlas Copco

"The many system views of the Tracealyzer from Percepio **makes it easy to quickly find solutions** that we have not seen using (Wind River) System Viewer. The visualization has several advantages over the system viewer and makes it easier to understand system behavior. This tool would be of great use for us."

Johan Fredriksson, Software Architect, SAAB AB.



# For Leading Software Platforms



Tracealyzer for Linux



Tracealyzer for VxWorks



FreeRTOS+Trace

**Micrium**  $\mu$ C/Trace

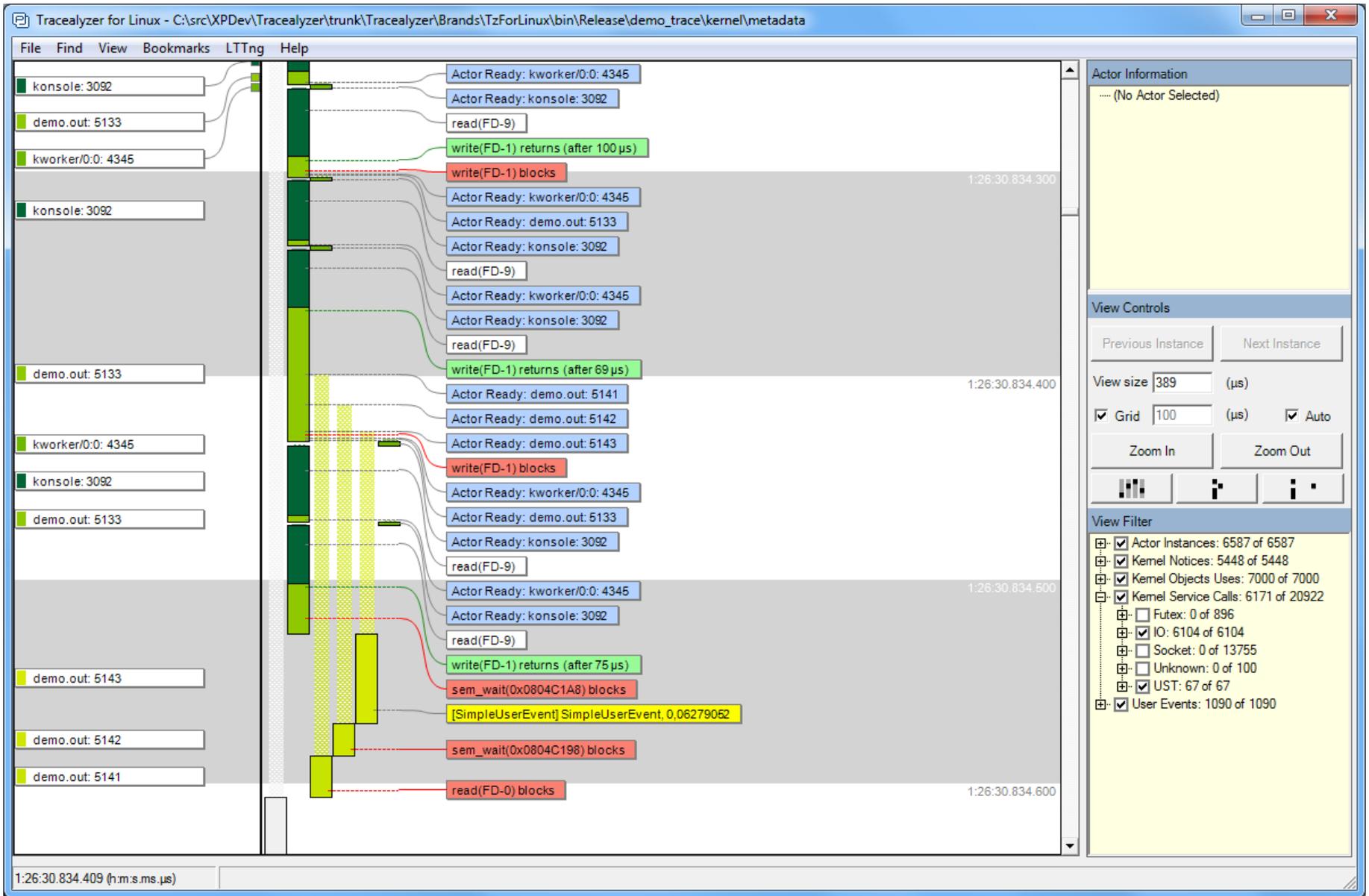


SafeRTOS+Trace

WITTENSTEIN



embOS-Trace



**Main Trace View:** Vertical time-line showing task scheduling, interrupts, kernel service calls and custom "User Events". Filter the display using the "View Filter" in the bottom right.

FD-3 (FileDesc)

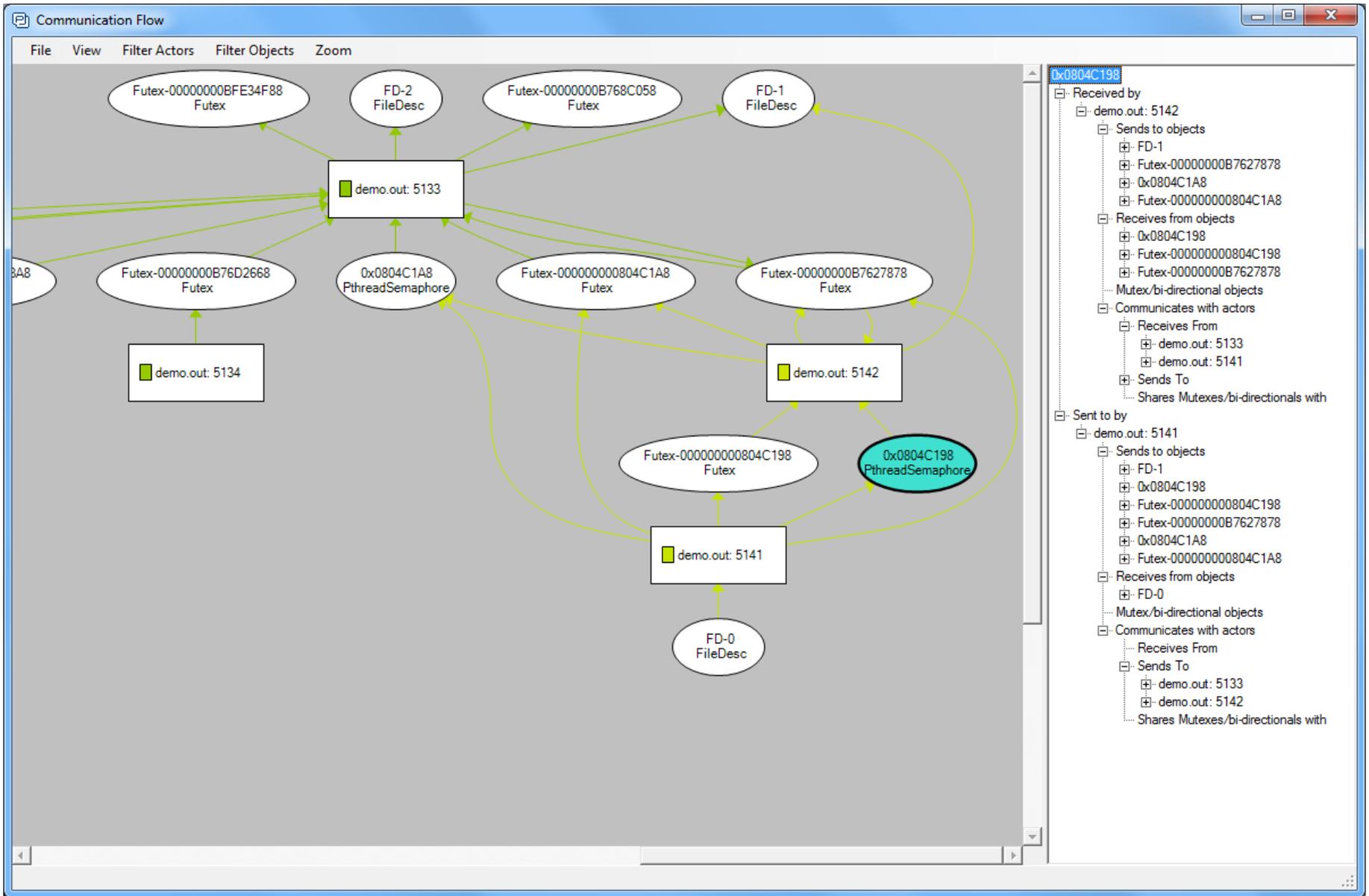
Timestamp	Actor	Event	Block time
23:30.135.491	kinfocenter	read	
23:30.135.741	kinfocenter	read	
23:30.214.679	bash	read	1.880
23:30.215.439	bash (2)	read	469
23:30.215.908	bash (2)	read	
23:30.215.918	bash (2)	read	299
23:30.216.216	bash (2)	read	
23:30.216.559	bash	read	
23:30.216.572	bash	read	277
23:30.216.849	bash	read	
23:30.227.963	konsole	write	
23:30.230.241	konsole	read	
23:30.230.743	konsole	read	
23:30.239.447	kinfocenter	write	
23:30.239.460	kinfocenter	write	
23:30.239.914	kinfocenter	write	
23:30.241.922	kinfocenter	write	
23:30.241.928	kinfocenter	write	
23:30.242.142	kinfocenter	write	
23:30.242.199	kinfocenter	read	
23:30.242.245	kinfocenter	write	
23:30.242.317	kinfocenter	write	
23:30.242.752	kinfocenter	write	

Timestamp: 23:30.215.439  
 Actor: bash (2)  
 Event: read  
 Status: Blocked for 469 (µs)  
 Parameter: N/A

Goto Entry/Exit Event

Show in Trace

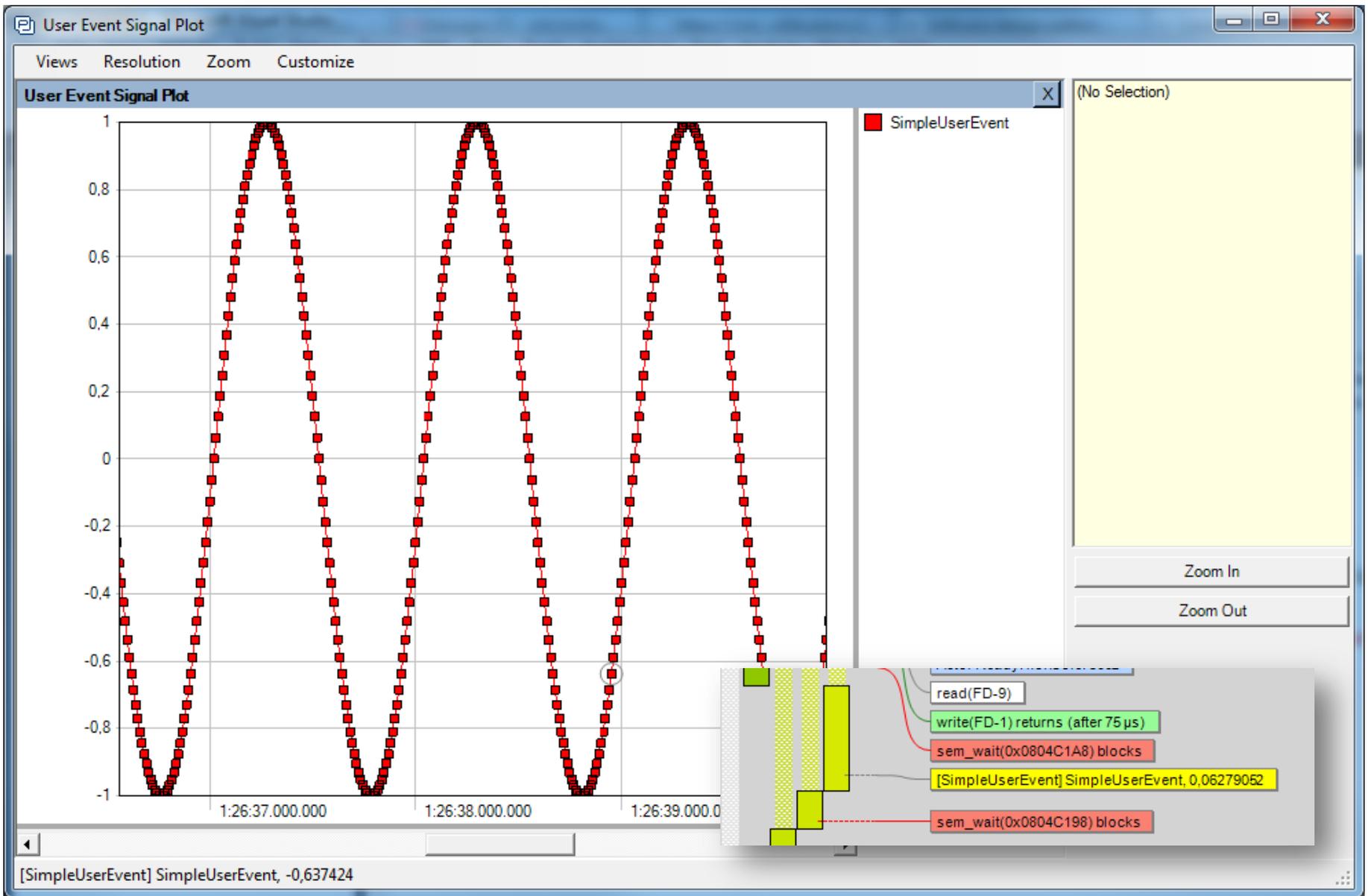
**Kernel Object History:** Double-clicking on a kernel call in the main trace view opens this view, showing the list of events for the same kernel object (e.g., File Descriptor or Semaphore ).



**Communication Flow:** Auto-generated summary of dependencies between threads and kernel objects. Can be generated for a selected interval, or for the whole trace.



**CPU Load Graph:** Displays the relative usage of CPU time, per thread and time interval on each processor core. Double-click on intervals to see the details in the main trace view. The interval displayed in the main trace view is indicated using a grey outlined rectangle.



**User Event Signal Plot:** Displays a plot of data arguments in "User Events", can be used for continuous input signals, state variables, or any data of interest.

The screenshot shows the 'Event Log' window with a list of events. The events are color-coded and include timestamps and descriptions. The right-hand side of the window contains a 'Filters' panel and a 'Sources' panel.

**Event Log List:**

- [1:26:33.657.318] Context switch on CPU0 to Unknown: swapper/0
- [1:26:33.661.661] Actor Ready: X11-NOTIFY: 911
- [1:26:33.661.668] Actor Ready: demo.out: 5143
- [1:26:33.661.689] Context switch on CPU0 to X11-NOTIFY: 911
- [1:26:33.661.705] recv(???)
- [1:26:33.661.720] Context switch on CPU0 to demo.out: 5143
- [1:26:33.661.750] [SimpleUserEvent] SimpleUserEvent, -1
- [1:26:33.661.764] Context switch on CPU0 to Unknown: swapper/0
- [1:26:33.671.239] Actor Ready: kwin: 1100
- [1:26:33.671.281] Context switch on CPU0 to kwin: 1100
- [1:26:33.671.372] recv(???)
- [1:26:33.671.468] Context switch on CPU0 to Unknown: swapper/0
- [1:26:33.671.833] Actor Ready: demo.out: 5143
- [1:26:33.671.845] Context switch on CPU0 to demo.out: 5143
- [1:26:33.671.868] [SimpleUserEvent] SimpleUserEvent, -0,9980267
- [1:26:33.671.876] Context switch on CPU0 to Unknown: swapper/0
- [1:26:33.674.049] Actor Ready: X11-NOTIFY: VBoxClient: 906
- [1:26:33.674.066] Context switch on CPU0 to X11-NOTIFY: VBoxClient: 906
- [1:26:33.674.070] FUTEX\_WAIT(Futex-0000000008CB6AA4) returns after 50092 us
- [1:26:33.674.075] FUTEX\_WAKE(Futex-0000000008CB6AD0)
- [1:26:33.674.083] FUTEX\_WAIT(Futex-0000000008CB6AA4) blocks
- [1:26:33.674.090] Context switch on CPU0 to Unknown: swapper/0
- [1:26:33.681.972] Actor Ready: demo.out: 5143
- [1:26:33.681.992] Context switch on CPU0 to demo.out: 5143
- [1:26:33.682.030] [SimpleUserEvent] SimpleUserEvent, -0,9921147
- [1:26:33.682.042] Context switch on CPU0 to Unknown: swapper/0
- [1:26:33.686.793] Actor Ready: X11-NOTIFY: 911
- [1:26:33.686.812] Context switch on CPU0 to X11-NOTIFY: 911
- [1:26:33.686.824] recv(???)
- [1:26:33.686.839] Context switch on CPU0 to Unknown: swapper/0
- [1:26:33.692.111] Actor Ready: demo.out: 5143
- [1:26:33.692.125] Context switch on CPU0 to demo.out: 5143
- [1:26:33.692.146] [SimpleUserEvent] SimpleUserEvent, -0,9822872
- [1:26:33.692.153] Context switch on CPU0 to Unknown: swapper/0
- [1:26:33.702.236] Actor Ready: demo.out: 5143
- [1:26:33.702.261] Context switch on CPU0 to demo.out: 5143
- [1:26:33.702.309] [SimpleUserEvent] SimpleUserEvent, -0,9685832
- [1:26:33.702.323] Context switch on CPU0 to Unknown: swapper/0
- [1:26:33.712.818] Actor Ready: X11-NOTIFY: 911

**Filters:**

- Include only lines matching at least one of
- Exclude entries matching at least one of
- Exclude entries before timestamp: 1:26:27.746.605
- Exclude entries after timestamp: 1:26:45.914.203
- Apply

**Sources:**

- Trace Events
  - Context Switches
  - Kernel Services
  - Kernel Notices
- User Events
  - SimpleUserEvent

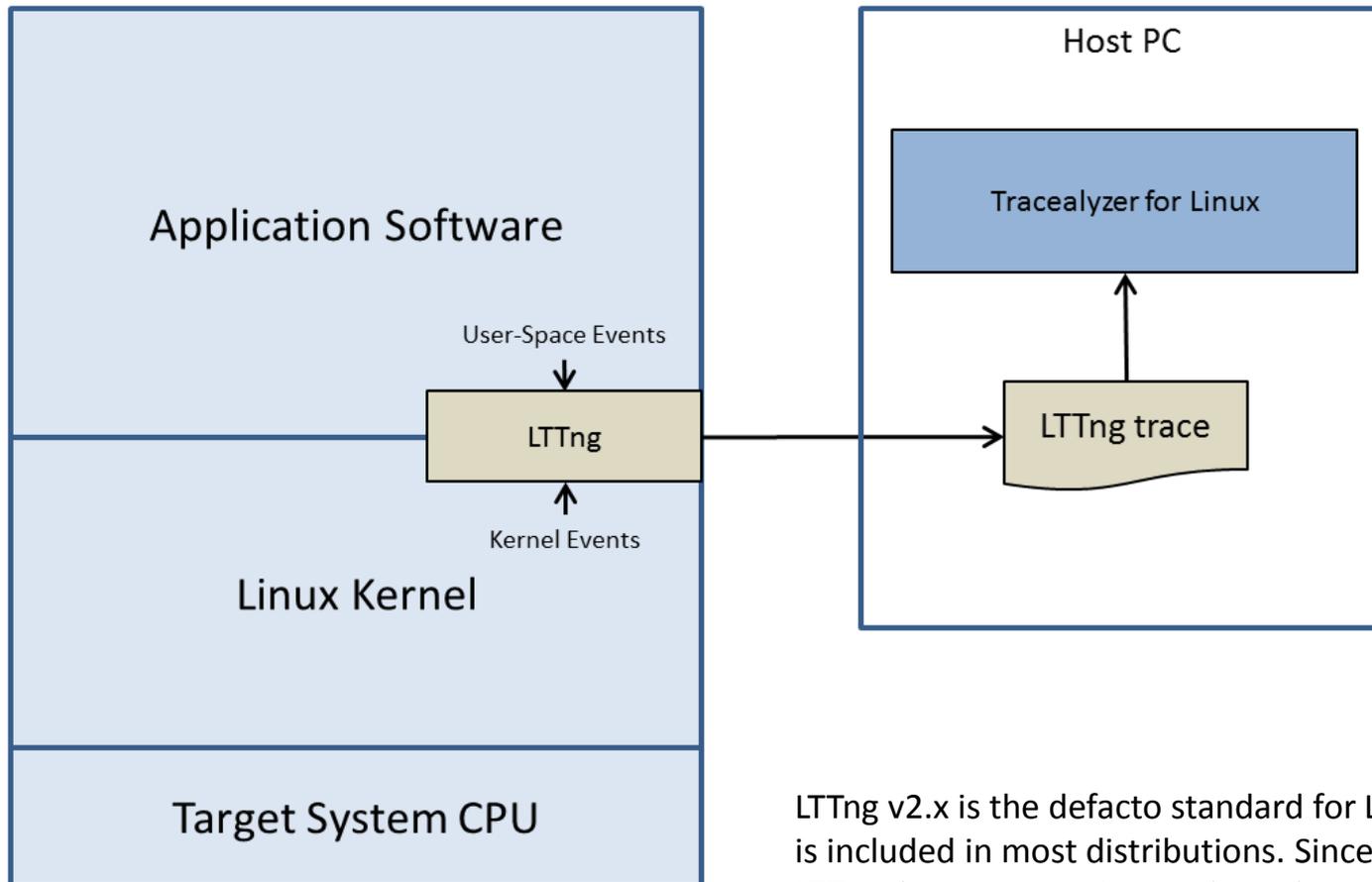
**Event Log:** Displays a textual list of the event timeline of the main trace view, with filtering and color coding. Exports traces in text format for comparisons and custom analyses.



# Other Views Provided

- Kernel Object Utilization
  - Watch queue sizes over time
- Kernel Call Intensity
  - Check the rate of kernel calls
- Kernel Blocking Times
  - See blocking on kernel calls
- Scheduling Intensity
  - The rate of context-switches
- Statistics Report
  - Timing and CPU usage
  - Descriptive statistics
  - Timing distributions per job
- Heap Memory Allocation
  - Malloc/free over time
- Actor Instance Graphs
  - Plot the timing of each job

# Tracealyzer for Linux relies on LTTng



LTTng v2.x is the defacto standard for Linux tracing and is included in most distributions. Since kernel v.2.6.38 LTTng does not require any kernel patching. But it works also on older kernels from v2.6.32.

# Tracing with LTTng

- Install LTTng
  - See <https://lttng.org/docs/#doc-installing-lttng>
- Create a trace session
  - `lttng create my-session`
- Select what kernel events to enable
  - `lttng list --kernel // shown the available kernel events`
  - `lttng enable-event --kernel sched_switch[, ...] [--all]`
  - “`sched_switch`” should always be included
- Select any userspace events to enable
  - `lttng list --userspace // shown the available userspace events`
  - `lttng enable-event --userspace [process:tracepoint] [--all]`
- Start tracing
  - `sudo lttng start // default output is ~/lttng-traces`
- Stop tracing
  - `sudo lttng stop`
  - `sudo lttng destroy // closes the trace session`
- Learn more about LTTng at <https://lttng.org/>

# Running on Linux

- Running Tracealyzer on Linux requires **Mono**, the Open Source .NET environment.
  - Check if already installed (mono --version)
  - Otherwise get it from <http://www.mono-project.com/>
- Download TracealyzerForLinux-2.7.5.tgz or later
  - From <http://percepio.com/tz/downloads/>
- Extract the .tgz archive to any suitable location
- Open a console and run mono TzForLinux.exe
  - You get 30 days free evaluation on registration
- To open an LTTng trace (a directory of files), select *File* -> *Open* and then locate the "metadata" file.

# Creating "User Events" from your code

## Old detailed method: LTTng tracepoint

```
#include <lttng/tracepoint.h>

TRACEPOINT_EVENT(percepiodemo, simpleuserevent,
    TP_ARGS(char*, channel, float, val),
    TP_FIELDS( ctf_string(channel, channel) ctf_float(float, val, val) ) )
...
char* channel = "SimpleUserEvent";
tracepoint(percepiodemo, simpleuserevent, channel, val);
```

## New easy method: tracef, a printf-style interface for LTTng tracepoint

```
#include <lttng/tracef.h>
...
tracef("Hello world: %d", myValue);
```

## New even easier method: Just write to "/proc/lttng-logger"

```
echo -n 'Hello world!' > /proc/lttng-logger
```



# Configuring "User Events" in Tracealyzer

Tracealyzer only displays **recognized** LTTng events. The "User Events" must therefore be specified in a "Platform Extension" file, added in *File -> Settings*. To see ALL LTTng events, use the "Raw Trace View".

Example: PercepioDemo.xml

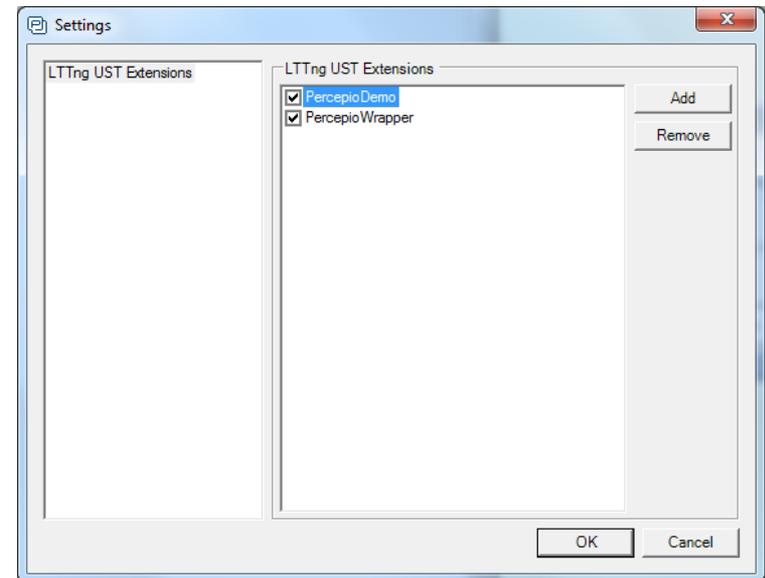
```
<?xml version="1.0" encoding="utf-8"?>
<PlatformExtension>
  <EventMap>
    <Event name="percepiodemo:simpleuserevent" type="UserEvent">
      <Parameter name="channel" var="channel"/>
      <Parameter name="val" var="val"/>
    </Event>
  </EventMap>
</PlatformExtension>
```

**name:** the Tracealyzer name for this parameter

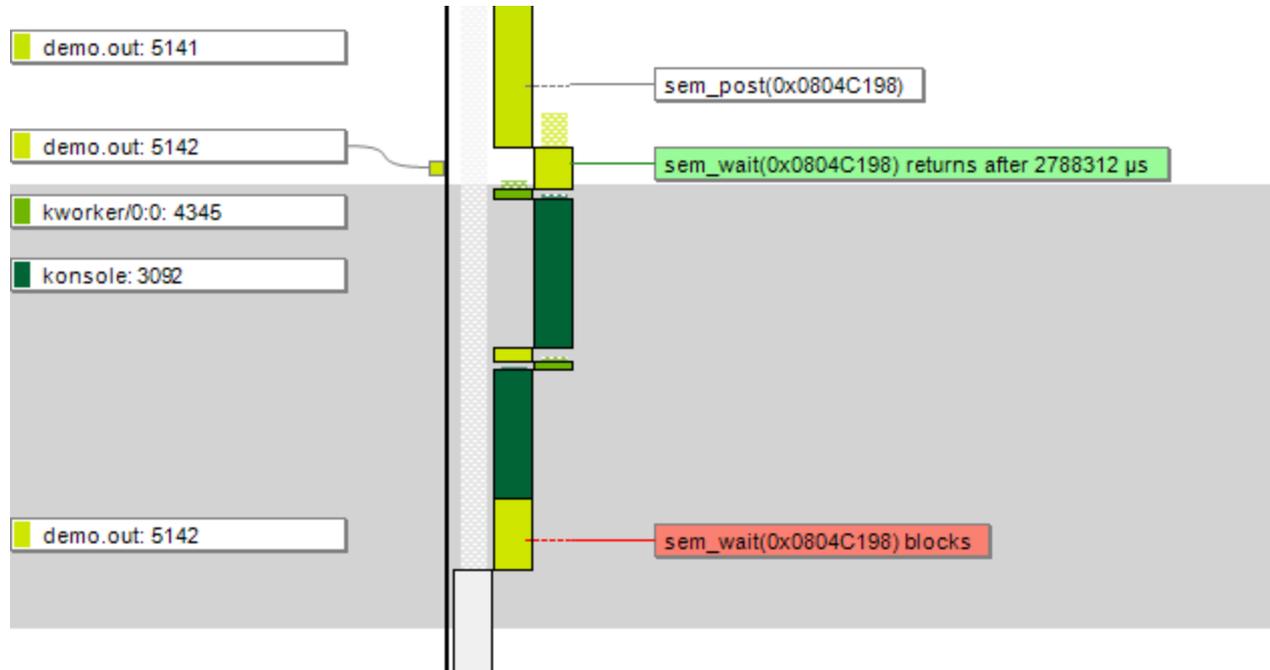
**var:** the name of the LTTng tracepoint

Parameter name "channel" has special meaning, specifies the *User Event Channel* name.

**This example is available in Help menu -> "LTTng UST Example"**



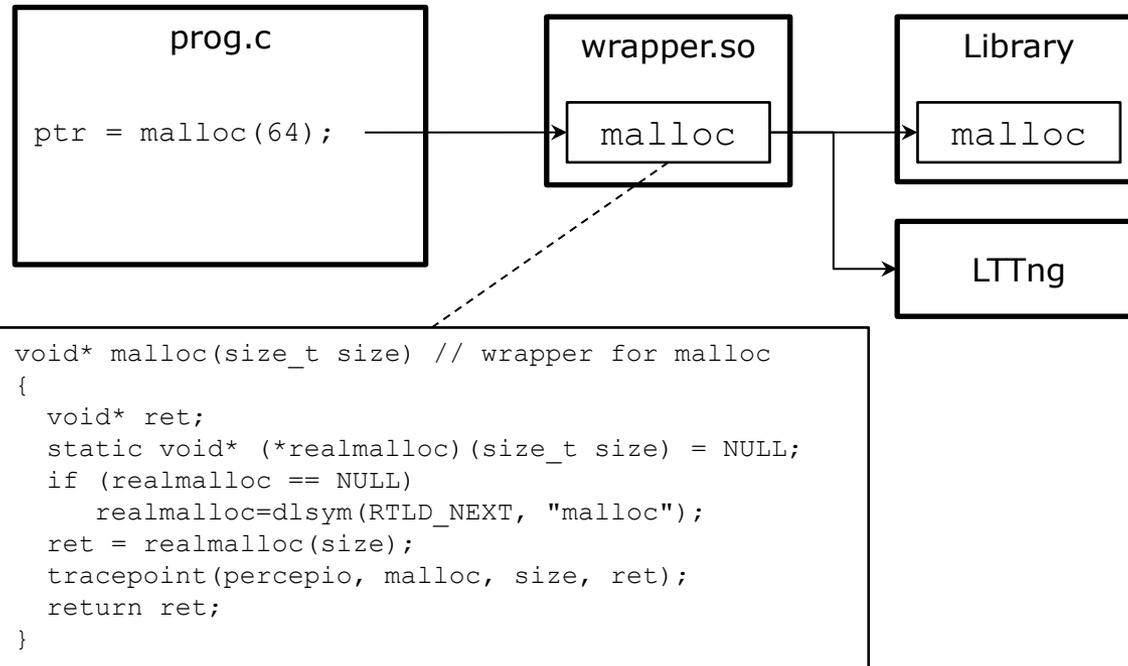
# Advanced: Tracing your own "Kernel Services"



Tracealyzer can be configured to display LTTng events from any function call as "Kernel Services" which enables many functions like Kernel Object History and matching of related events. This requires three steps...

# Step 1. Record the function calls with LTTng

```
$ LD_PRELOAD=./wrapper.so ./prog
```



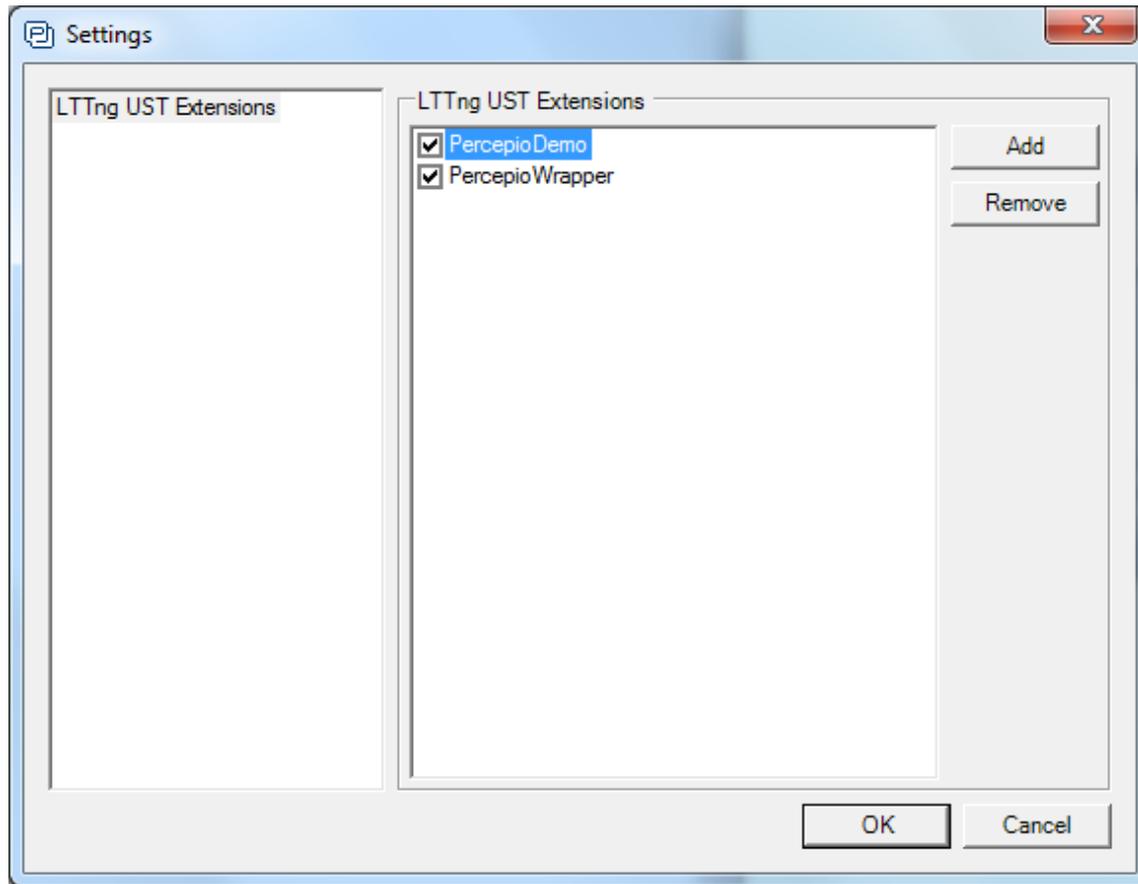
With tracepoints in wrapper functions and using LD\_PRELOAD, the code don't need to be changed or even recompiled. The dynamic linking will route the calls to the new wrapper function, which calls LTTng plus the original function.

## Step 2. Create a Platform Extension XML file for the new LTTng events

```
<PlatformExtension>
  <TargetPlatform>
    <KernelServiceGroups>
      <KernelServiceGroup name="UST"> - the group name showed in the View Filter
        <KernelService name="sem_wait" operation="IncreaseSemaphore"> - The name and meaning of event
          <Parameter name="sem"/>
        </KernelService>
        ...
      </KernelServiceGroup>
    </KernelServiceGroups>
    <ObjectClasses>
      <ObjectClass name="PthreadSemaphore" type="Semaphore"/> - Type and display name of object class
    </ObjectClasses>
  </TargetPlatform>
  <ObjectMap>
    <Object class="PthreadSemaphore" format="0x{0:X8}"/> - How to format identifiers for such objects
  </ObjectMap>
  <EventMap> - Mapping between LTTng events and kernel services calls
    <Event name="percepio:sem_wait_entry" service="UST/sem_wait" type="Entry" status="Detect">
      <Parameter name="sem" var="sem" class="PthreadSemaphore"/>
    </Event>
    <Event name="percepio:sem_wait_exit" service="UST/sem_wait" type="Exit" status="Success">
      <Condition cvar="ret" cval="0" cop="&lt;" target="status" value="Timeout"/> - if (ret < 0) status = "Timeout";
    </Event>
    ...
  </EventMap>
</PlatformExtension>
```

This example is available in Help menu -> "LTTng UST Example"

# Step 3. Import the Platform Extension



# Questions?

[support@percepio.com](mailto:support@percepio.com)