Problem And Goals

- Allow user mode applications to create, open and send diagnostic data to trace_event / dyn_event
  - Minimal overhead, especially when nothing is listening to the trace_event
  - Works across multiple languages/binary types and many cgroups without entering each namespace
  - Works with standard capture and analysis techniques (ftrace, perf, eBPF)
Problem Scenario

**Have:** Many processes running within many cgroups using different languages (Python, Go, Rust, C/C++, C#, Java)
- Single monitoring agent in root namespace, entering namespaces as required to find correct paths, PIDs, etc
- Multiple mechanisms to collect, have to merge/decode to get to a unified view

**Want:** All user and kernel events into a single eBPF program or trace buffer/file without entering cgroup namespaces
- Need consistent aux data when event is emitted (PMU data, Stack data, etc)
- Want to avoid having to mix collection mechanisms and merge/decode afterwards
- Want to avoid a daemon/agent with each cgroup/namespace
Proposed ABI

- **Creation / Open**
  - \texttt{event\_fd = open(“/sys/kernel/tracing/user\_events\_data”);}
  - \texttt{event\_id = ioctl(REG, “MyUserEvent”)}
  - MyUserEvent is now available to be used in code and also via tracefs, perf, eBPF, etc as a trace\_event / dyn\_event.

- **Writing / Emitting Data**
  - \texttt{write(event\_fd, “MyData”); /* Only works after REG IOCTL */}

- **Status**
  - \texttt{events\_page = mmap(“/sys/kernel/tracing/user\_events\_mmap”);}
  - \texttt{if (events\_page[event\_id]) { /* write, etc. event\_id from ioctl(REG) */ } }
  - Bits 0-6 describe system listening (ftrace, perf, etc). Bit 7 reserved for “Others”
  - All Bits clear if nothing is listening
Proposed Flow Diagram

trace_event_class

perfProbe

probe

trace_event_call

tracepoint

probe register callbacks

pass data to probes

lookup or create

events_page

data

create

mmap

write

ioctl

user process