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**
  - event_fd = open(“/sys/kernel/tracing/user_events_data”);
  - 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**
  - write(event_fd, “MyData”); /* *Only works after REG IOCTL* */
- **Status**
  - events_page = mmap(“/sys/kernel/tracing/user_events_mmap”);
  - 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