Secure Operating System Design and Implementation
Events

Jon A. Solworth

Dept. of Computer Science
University of Illinois at Chicago

February 1, 2011

Part I

Events overview

Overview

- Events are associated with actions which have non-trivial latency
- For example, an event may be to wait for:
  - A Timer to expire
  - An RPC to complete
  - A packet to arrive
- Events enable syscalls to complete quickly, returning an eventId which will be later used to retrieve the data from the completed syscall
- A single process can have multiple events outstanding, and wait to act on the events as they arrive
- Note that Ethos Events (described here) are distinct from the similarly named XenEvents which is a signalling mechanism.

Part II

Event trees

Event trees

- A major cost is an OS is a context switch, which changes the process which is executing.
- Allowing a process to create many events and only be awoken when the relevant condition is satisfied reduces context switches
- Ethos thus allows a process to wait (via block) on a tree of events, and to specify which combination of events need to complete before the process is awoken (i.e., context switched into).
- For example, a process may do several disk reads and wait on all of them to complete or the arrival of a new request.

Event trees

- Event tree is an n-ary trees
- Each leaf is an event
- Each internal node specifies the number of children which must complete before the internal node completes.
- The tree can thus represent *and*, *or* and *thresholds*

Part III

Event completion

Event completion

- After an event completes, the event can be retired.
- Retiring an event returns the values as specified by the system call which created the event.
- For example, retiring a read returns a status and the object read
- While retiring a write returns just a status

Part IV

Internal event processing

When are events processed?

- If events were processed at any time, it could cause race conditions with other code in the kernel.
- Thus interrupts are designed to be mostly deferred until system call has completed
- Right before the return to user space.
- At that point it is safe to execute the events
- And hence the synchronization is only needed between the producer and consumer of the Ethernet queue to ensure the RPC packets are passed properly.

Internal event processing

- Consider an event created for an RPC
- The eventId is passed to the shadowdaemon as a parameter
- The same eventId is returned to Ethos when the shadowdaemon completes the RPC
- The event is looked up (using the EventId handle)
- The values from the RPC can be stored in the event or effect some other data structure such as FileInformation.
- If the processing is complete on the event, then complete is called.
- Otherwise the next step in the event occurs.
- In addition, there may be other events which are queued up behind the current event, and these are processed as well.

Memory

- Once the init process starts, the kernel is always executed in the context of some process.
- At the time of the system call, the context is the process invoking the system call
- But later, when events are triggered, the context is a different process
- Therefore it is important the copies to and from user space only occur in syscall.c