Embeddable Mac OS X CoreFoundation Notifier

EuroTcl 09

5 June 2009

Daniel A. Steffen

das@users.sf.net

Outline

- Tcl Notifier Recap
- CoreFoundation RunLoops
- Embeddable CoreFoundation Notifier
- Embedded Notifier Setup
- Caveats
- Demo

Tcl Notifier Recap

- Core of Tcl event loop
  - Wait for OS events: file readable, socket activity, timer...
  - Via `Tcl_DoOneEvent()` → `Tcl_WaitForEvent()`
- Unthreaded UNIX notifier: `select()`
- Threaded UNIX notifier: `pthread_cond_timedwait()`
  - Common notifier thread waiting in `select()`
- Timeouts passed to wait API ensure timers are serviced

Tcl Notifier on Mac OS X

- UNIX notifier ok if only POSIX APIs are used
- Many OS X facilities rely on platform-specific event loop API
  - e.g. interact with WindowServer, Bonjour, Mach Ports, etc
- Need notifier integrated with CoreFoundation RunLoop
  - CF notifier available since 8.4.10/8.5a3 (May 2005)
  - only for event loop driven via `Tcl_DoOneEvent()`
  - i.e. TkAqua, Bonjour extension, but not embeddable

CoreFoundation RunLoops

- Common event loop mechanism all higher APIs build on
  - based on Mach port IPC internally
- One CFRunLoop per thread
- API to run (until given timeout) and stop current runloop
  - runloops can run recursively, can run in custom *modes*
- Runloop monitors *sources*, *timers* and *observers*
  - when triggered, these execute specified callback

CoreFoundation RunLoops

- Runloop *sources* monitor event sources
  - e.g. Mach ports, sockets, custom events
  - sources can be signaled (and runloops woken up) from other threads
- Runloop *timers* trigger at specified interval (once/repeatedly)
- Runloop *observers* trigger each time specific stages of runloop execution are reached
  - e.g. before sources, before waiting, after waiting, etc

CoreFoundation Notifier

- Pre-8.5.7 (non-embeddable):
  - always uses notifier thread to run `select()`
    - uses `pthread` API directly to work in unthreaded Tcl
    - Tcl-specific runloop source in each thread
    - Notifier thread signals source to wake up waiting thread
    - `Tcl_WaitForEvent()` blocks in `CFRunLoopRun()` until a source is triggered or timeout reached
    - Tcl events are enqueued from `Tcl_WaitForEvent()` once `CFRunLoopRun()` returns

Emeddable CoreFoundation Notifier

- Tcl events and timers need to be enqueued & serviced when event loop is not being run via `Tcl_DoOneEvent()`
  - Tcl runloop observer and runloop timer in each thread
  - Timer wakes up runloop so tcl timers can trigger
  - Observer services tcl events before runloop wait state
  - Tcl events are enqueued from tcl runloop source callback
  - `Tcl_WaitForEvent()` essentially reduced to a call to `CFRunLoopRun()`

Emeddable CoreFoundation Notifier

- Finer grained locking: per-thread lock on thread-specific data accessed from both thread and notifier thread
  - replaces global lock for all notifier structures
  - uses OS X spinlock API for reduced overhead over `pthread` mutexes
- Runloop observer places/removes thread from global waiting list as runloop is entered/exited
  - wakes up notifier thread via trigger pipe, causing `select()` masks to be recomputed

Emeddable CoreFoundation Notifier

- Custom runloop mode for nested invocations of tcl event loop (avoid loosing wakeups of non-tcl sources)
- Notifier thread created lazily on first `Tcl_WaitForEvent`
- `Tcl_Sleep()` based on `CFRunLoopRun()` to allow non-tcl events to be processed during sleep
- `TclUnixWaitForFile()` currently still `select()` based
  - can block embedder event processing
- New internal stubs API to add runloop mode to the set of modes where tcl notifier processes events

Emedded Notifier Setup

```
Tcl_SetServiceMode(TCL_SERVICE_ALL);
```

- Call during Tcl initialization
  - sets up runloop timer and enables tcl event servicing from runloop observer
- Use standard high-level-API facilities to run CFRunLoop
  - e.g. `-[NSApplication run]`
- Without this, CF notifier behaves as pre-8.5.7
  - e.g. tclsh continues to use that mode of operation

Caveats

- Watch for nested tcl event loops (e.g. `[vwait]`)
  - block event processing in embedder
  - avoid long `[vwait]` if possible (e.g. use coroutines)
    - if necessary, handle by adding a runloop observer that processes embedder events from tcl event loop
      - difficult to get right, e.g. Cocoa may not expect this
      - need to ensure Tcl is not re-entered from embedder
- Nested embedder event loops work fine
  - may need to add tracking runloop modes to tcl notifier

Caveats

- `fork()` not immediately followed by `execve()`
  - e.g. via TclX or Expect
- Not supported by threaded UNIX notifier
  - CF notifier always uses notifier thread
  - `atfork()` handlers in place to reset state after `fork()`
    - lazy notifier thread re-creation in child
- BUT in recent Mac OS X releases, CoreFoundation actively guards against its use after `fork()` and calls `abort()`
  - use unthreaded, non-corefoundation notifier...

Demo

Thanks

http://categorifiedcoder.info/