Real-Time Mach Timers

Exporting Time to the User

Stefan Savage
Hideyuki Tokuda
CMU School of Computer Science
savage@cs.cmu.edu
Contents

- Motivation
- Clocks
- Timers
- Timing Faults
- Current Status
Time Services

- Measurement
  - Precise timestamps

- Synchronization
  - Block until time x
  - Do something at time x

- Scheduling
  - Stop thread and do something else at time x
What Mach Provides...

- Measurement
  - host_get_time()
- Synchronization
  - mach_msg(...., timeout)
- Scheduling
  - Go Fish
Problems With What Mach Provides

- **Resolution**
  - Only represents time to system tick resolution (~10ms on present Mach systems)
  - Only *possible* to represent time to 1 ms granularity

- **Accuracy**
  - `host_adjust_time()` algorithm, inherited from BSD, forced on user
  - `mach_msg(...,timeout)` is a relative value, hence it is impossible to guarantee accuracy

- **Flexibility**
  - Only provides blocking facility for executing thread
  - One hardwired time source
What We Provide Instead

- **Clock Devices**
  - Mach devices which measure the passage of time to highest allowable resolution

- **Timers**
  - Synchronization objects which perform “certain actions” at certain times
  - Among “certain actions” are sending a message, signaling an event counter, or manipulating thread execution state
  - Rely on clock devices to provide information about when to “expire”
What’s A Clock Device?

- Regular Mach 3.0 device which measures time
  - Uses `device_open`, `device_close`, `device_getstatus`, and `device_setstatus` interfaces
  - Additionally informs the system it can provide “clock operations”

- Clock operations
  - Get/Set Time
  - Get/Set Resolution
  - Map Time
  - Kernel Only: Enqueue and Dequeue Timer Objects
How Does It Work?

- Hardware provides time service via periodic interrupt or readable counter (or some combination thereof)
- Timer objects are bound to clock devices which can interrupt
- Clocks are responsible for alerting the system when any of their bound timer objects have “expired”
Clock Interrupt

Timer Object Queue

Hi, I’ve expired!

Check Timer Object Queue

Timer expires at 3:50pm
Timer expires at 3:40pm
Timer expires at 4:10pm
Timer expires at 4:12pm

Current Time 3:41pm

Clock Device

Real-Time Mach
Maintaining 10ms Services

Real-Time Mach

clock_interrupt()
{
  do stuff;
}
Multiple Clocks

Clock A Timer Object Queue

Check Timer Object Queue

Clock Device A
resolution = 10ms

INTERRUPT!

Clock Device B
resolution = 100us

INTERRUPT!

Clock B Timer Object Queue

Real-Time Mach
What’s A Timer Object?

- Kernel synchronization object represented by a port
- Specifies a relative or absolute time at which to perform some action
- Can be periodic, in which case it automatically reloads at expiration time
- Bound to a clock device when created
- Accuracy is bounded by resolution of clock device
How Does It Work?

- Timers become active by being enqueued on their clock device’s queue.
- When the clock device notices that one of it’s timers has expired it sets an AST.
- At AST time the timers are dequeued and they execute their expiration action (most likely `thread_resume` or `mach_msg_send_from_kernel`).
Clock Interrupt

Timer Object Queue

- Check Timer Object Queue
- If timer has expired set AST
- AST HANDLER: Remove expired timers from queue and execute expiration action

Interrupt!

Clock Device

Real-Time Mach

Send Message (Un)block Thread
Signal Event
Timer Interface

- `timer_create()`
  - Creates a timer and binds it to a clock device
- `timer_terminate()`
  - Destroys a timer
- `timer_sleep()`
  - Blocks current thread until timer expires
- `timer_arm()`
  - Send a message to a port when timer expires
- `timer_cancel()`
  - Deactivate a timer; current instantiation or all
- `timer_info()`
  - Get the current timer state
Real-Time programs have timing requirements which must be satisfied for program and system correctness (e.g., period, deadline, etc...).

When timing requirements are not met this is a *timing fault*, and program execution must be halted to avoid cascading overload.

How to respond to a timing fault is application dependent and should be under user control.
How Does It Work?

- `timer_arm()` has a flag which indicates that the calling thread should be suspended when the timer expires.
- The thread which receives the expiration message can manipulate the offending thread as appropriate.
- This functionality is “built-in” RT-Mach’s `rt-thread` package using the same mechanisms.
Real-Time Mach

Deadline Handler

Deadline Handler Thread:
- Handler Blocks Waiting For Message
- Timer Message Received
- Take Corrective Action

Application Thread:
- Application Thread Starts
- Missed Deadline!
- Thread Resumes (Soft-RT)

Time

Real-Time Mach
## Performance

<table>
<thead>
<tr>
<th>operation</th>
<th>time in usecs</th>
</tr>
</thead>
<tbody>
<tr>
<td>timer_arm() [one-shot]</td>
<td>33</td>
</tr>
<tr>
<td>timer_arm() [periodic]</td>
<td>37</td>
</tr>
<tr>
<td>timer_cancel()</td>
<td>19</td>
</tr>
<tr>
<td>timer_create()</td>
<td>220</td>
</tr>
<tr>
<td>timer_terminate()</td>
<td>119</td>
</tr>
<tr>
<td>mach_thread_self()</td>
<td>20</td>
</tr>
<tr>
<td>null trap</td>
<td>7</td>
</tr>
</tbody>
</table>

**Table 1: Latency of timer operations**

*Results from a 66Mhz 486DX/2 with 16MB of 70ns RAM and 64K of 25ns cache*
# Performance

<table>
<thead>
<tr>
<th>expiration action</th>
<th>time in usecs</th>
</tr>
</thead>
<tbody>
<tr>
<td>unblock thread</td>
<td>106</td>
</tr>
<tr>
<td>send message</td>
<td>162</td>
</tr>
</tbody>
</table>

Table 2: *Latency of thread execution from clock interrupt time*

<table>
<thead>
<tr>
<th>operation</th>
<th>time in usecs</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestamp (using device_get_status)</td>
<td>85</td>
</tr>
<tr>
<td>timestamp (using mapped page/io port)</td>
<td>&lt;2</td>
</tr>
</tbody>
</table>

Table 3: *Latency of clock device timestamp operations*
Related Work

- POSIX
  - nanosleep, timer_settime, etc...
- NT Executive
  - Timer Objects
  - WaitForSingleObject, WaitForMultipleObjects
  - APC’s and DPC’s
- OSF RI
  - Clocks and Alarms
  - clock_sleep, clock_alarm, clock_get_time...
  - Alarms aren’t first class objects so difficult or expensive to implement periodic alarms, timing faults, or alarm cancelation
Current Status

- Runs on 386 AT platform, DECstation, and Sun3
- Support for a variety of clock hardware
  - I8254, MC16146818 clock chips, and STAT! timer board for the 386 AT platform
  - MC16146818 clock chip for the DECstation
- Timers used to implement kernel real-time threads in RT-Mach MK78
- Coming soon to a mainline Mach 3.0 kernel
Current Work

- Timers used to support user level real-time threads based on scheduler activations
- Timers used extensively in CPU reservation prototype