Outline for Today

• Objectives:
  – Time

• Administrative details:
  – Questions?

Highway 110 with Locks and Condition Variables

OneVehicle(int direc) //direc is either 0 or 1; giving the direction in which the car is to cross
{
    ArriveBridge(direc);
    CrossBridge(direc);
    ExitBridge(direc);
}

ArriveBridge(dir)
{
    bridgeLock->Acquire();
    if (dir == direction & passedby[!dir] > LIMIT)
        stop = TRUE;
    while (stop || (num_on_bridge != 0 & dir != direction) ||
           num_on_bridge > 2)
    {
        waiting[dir]++;
        OKtogo->Wait();
        waiting[!dir]--
    }
    if (waiting[!dir]) passedby[!dir]++;
    if (dir != direction) {
        direction = dir;
        passedby[!dir] = 0
    }
    num_on_bridge++;
    OKtogo->Broadcast();
    bridgeLock->Release();
}

ExitBridge(dir)
{
    bridgeLock->Acquire();
    num_on_bridge--;
    if (num_on_bridge == 0)
    {
        direction = !dir;
        stop = FALSE;
        passedby[!dir] = 0
    }
    OKtogo->Broadcast();
    bridgeLock->Release();
}
Uses of Time

- Coordinating events
  - Synchronized clocks
- Measurements – durations of activities
  - Stability – ability to maintain constant frequency
    - Environmental factors (temperature) or age
    - Synchronization protocols that adjust clock
- Driving periodic events
  - Granularity (frequency)
- Scheduling dynamic events at a particular time in the future.
  - Accuracy
  - Relative or absolute time?

Time Basics (Linux)

- Real time clock (RTC) keeps track of time even when system is off – boot-time initialization
- System timer – provide periodic interrupts
  - Programmable interrupt timer running at tick rate of HZ frequency
    - Time update (jiffies, wall clock time), do accounting (resource usage), dispatch events that are due (dynamic timers), rescheduling
  - Jiffies – number of ticks since reboot
  - Time of day
    - xtime structure – contains seconds since Jan 1 1970; wall clock time based on that.
- Delaying execution by looping `udelay(us)` or `sleeping schedule_timeout(s*HZ)`
**Dynamic Timers**

- Created and destroyed dynamically
- Handler is run when tick count is \( \geq \) expiration time.
- `init_timer(&mytimer);`
  
  `mytimer.expires = jiffies + delay;`
  
  `mytimer.data = 0; // arg passed to handler`
  
  `mytimer.function = myhandler;`
- `add_timer(&mytimer);`
- Can change `mod_timer` or remove `del_timer_sync`
- Timers are stored in buckets depending on how far into the future they should expire.
- Run asynchronously with respect to other code – protect shared data appropriately.

**Soft Timers**

_Aron & Druschel_

- Goal: to provide usec granularity events with low overhead.
  - Do not want timer interrupts at that granularity
- Approach: To leverage trigger points when execution has already been interrupted – amortize context switch and cache pollution already incurred by other causes.
  - End of syscall processing, end of exception handler, end of executing interrupt handler, during CPU idle loop
  - Bounded overrun if a trigger point doesn’t happen – backup hardware interrupt set
Accuracy

Example of minimum Event Time (just larger than $T=1$)

Example of maximum Event Time (just smaller than $T+X=4$)

Fig. 1. Lower and upper bounds for event scheduling.

Overhead

Table I. Per-Event Timer Costs with Null Event Handler

<table>
<thead>
<tr>
<th></th>
<th>Alpha-500</th>
<th>8253/P11-300</th>
<th>8253/P11-500</th>
<th>APIC/P1I-500</th>
<th>Soft Timers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overhead (μsec)</td>
<td>8.64</td>
<td>4.45</td>
<td>4.36</td>
<td>0.8</td>
<td>~0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>APIC/P1I-500</th>
<th>Soft Timers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overhead (μsec)</td>
<td>5.1</td>
<td>3.5</td>
</tr>
<tr>
<td>lcache-misses ($x10^6$)</td>
<td>153.2</td>
<td>149.7</td>
</tr>
<tr>
<td>Dcache-misses ($x10^6$)</td>
<td>551.4</td>
<td>377.9</td>
</tr>
<tr>
<td>ITLB-misses ($x10^6$)</td>
<td>18.25</td>
<td>17.00</td>
</tr>
</tbody>
</table>

Timer costs with synthetic event handler scheduled every 10usec
Trigger Occurrence

Fig. 2. Trigger state interval (CDF), 300MHz PIL.

Trigger Sources

<table>
<thead>
<tr>
<th>Source</th>
<th>Fraction of samples (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>syscall</td>
<td>47.7</td>
</tr>
<tr>
<td>ip-output</td>
<td>28</td>
</tr>
<tr>
<td>ip-intr</td>
<td>16.4</td>
</tr>
<tr>
<td>tcpip-others</td>
<td>5.4</td>
</tr>
<tr>
<td>traps</td>
<td>2.5</td>
</tr>
</tbody>
</table>
Reference Broadcast Time Synchronization
Elston, Girod, Estrin

- Goal: precise (1 us) relative time scale among wireless sensor nodes.
- Exploits (requires) broadcast medium
- Synchronizes among common receivers of the same beacon
  - I got it at time $t_1$ on my clock; you got it at time $t_2$ on your clock -> phase offset between clock values for this individual pulse is
    $\text{Offset}[i,j] = t_j - t_i$
- Must also deal with clock skew – clocks drift apart by ticking at slightly different rates

Naive Clock Synchronization

```
request

localclock = timestamp + rtt/2

timestamp = localclock

reply(timestamp)
```
Critical Path Nondeterminism

- RBS removes many causes of nondeterminism from critical path

Multiple Broadcasts

Why

Figure 2: A histogram showing the distribution of inter-receiver phase offsets recorded for 1,478 broadcast packets, grouped into 1μsec buckets. The curve is a plot of the best-fit Gaussian parameters. (μ = 0, σ = 111μsec, confidence=99.8%)

Pairwise error: 2 receiver (bottom line); 20 receiver (top line)
Fixing Clock Skew

- Perform least-squares linear regression – best fit line over time; recover frequency and phase of local clock wrt remote one from slope and intercept
- Also an effective post-facto synchronization scheme – extrapolate backward
- Evaluation comparing RBS to NTP: 8 times better wrt average error (light load)

Multi-Hop

- Intersecting receivers can relate clocks from both A and B
- $E_i(R_j)$ means time of event $i$ according to $j$’s clock.

1. Receivers R1 and R7 observe events at times $E_i(R_1)$ and $E_i(R_7)$, respectively.
2. R4 uses A’s reference broadcasts to establish the best-fit line needed for converting clock values from R1 to R4. This line is used to convert $E_i(R_1)$ to $E_i(R_4)$.
3. R4 similarly uses B’s broadcasts to convert $E_i(R_4)$ to $E_i(R_7)$.
4. The time elapsed between the events is computed as $E_i(R_7) - E_i(R_7)$