Using Threads in Interactive Systems: A Case Study

Phil Gibbons

15-712 F15

Lecture 4

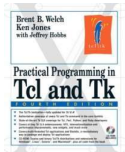
Today's Reminders

- I posted to Piazza at 10:45 am this morning – anyone not able to see it?
- Waitlist (33 currently enrolled)
- Office Hours: Wednesdays 4:30-5:30 pm

Using Threads in Interactive Systems: A Case Study

Carl Hauser, Christian Jacobi, Marvin Theimer, Brent Welch, Mark Weiser SOSP'93

- Carl Hauser (Washington State)
- Christian Jacobi (White Hawk Software)
- Marvin Theimer (Amazon Web Services)
- Brent Welch (Google; was CTO of CMU start-up Panasus)
- Mark Weiser (d.1999; Father of ubiquitous computing)







"To understand systems it is not enough to describe how things should be; one also needs to know how they are."

A Case Study

- **Examined 2 large research & commercial systems for the ways that they use threads (Cedar & GVX)**
  - Largest & Longest-used thread-based interactive systems in everyday use (2.5M lines of code, 10 year period)
  - GVX is purely interactive; Cedar also has compiles, makes, etc
- **Three methods:**
  - Analysis of macroscopic thread statistics
  - Analysis of the microsecond spacing between thread events
  - Static analysis of the implementation code ***
- **Identified 10 different paradigms of thread usage**
  - Defer work, general pumps, slack processes, sleepers, one-shots, deadlock avoidance, rejuvenation, serializers, encapsulated fork, exploiting parallelism

Thread Model (Mesa)

- **Multiple, lightweight pre-emptively scheduled threads that share an address space**
- **Primitives: Fork, Join, Detach**
- **Monitors – set of procedures sharing a mutex lock**
- **Condition variables – queue of threads waiting for a condition to become true (Wait, Notify, Broadcast)**
  - WHILE NOT(condition) DO WAIT cv END
- **7 scheduler priorities w/round-robin among equal priority**
  50 millisec scheduling quantum
  50 microsec thread switch time
  - Yield: causes the scheduler to run immediately
  - Linux: priority -20 to 99; default 100ms, 1-3 microsecs

Dynamic Thread Behavior

- **Dynamic data benchmark suite**
  - Compilation, formatting a document, previewing pages, user interface tasks (keyboarding, mousing, scrolling)
  - Sparcstation-2 running SunOS-4.1.3
- **3 Thread classes**
  - Eternal threads – repeatedly waited on CV, then ran briefly before waiting again
  - Worker threads – forked to perform an activity
  - Transient threads – forked by a long-lived thread, run for short while, then exit
- **Time between thread switches**
  - Bimodal: 75% < 5 ms, second peak ~45 ms

Dynamic Thread Behavior

Table 1: Forking and thread-switching rates

Cedar	Forks/sec	Thread Switches/sec
Idle Cedar	0.9	132
Keyboard input	5.0	269
Mouse movement	1.0	191
Window scrolling	0.7	172
Document formatting	3.6	171
Document previewing	1.6	222
Make program	0.3	170
Compile	0.3	135
GVX		
Idle GVX	0	33
Keyboard input	0	60
Mouse movement	0	34
Window scrolling	0	43

Observations

- **Most execution intervals are short, but longer execution intervals account for 20-50% (30-80%) of total execution time**

"None of our benchmarks exhibited forking generations greater than 2."

Dynamic Thread Behavior

Table 2: Wait-CV and monitor entry rates

Cedar	Waits/sec	% timeouts	ML-enters per sec
Idle Cedar	121	82%	414
Keyboard input	185	48%	2557
Mouse movement	163	58%	1025
Window scrolling	115	69%	2032
Doc formatting	130	72%	2739
Doc previewing	157	56%	1335
Make program	158	61%	2218
Compile	119	82%	1365
GVX			
Idle GVX	32	99%	366
Keyboard input	38	42%	1436
Mouse movement	33	96%	410
Window scrolling	25	61%	691

Table 3: Number of different CVs and monitor locks used

Cedar	# CVs	# MLs
Idle Cedar	22	554
Keyboard input	32	918
Mouse movement	26	734
Window scrolling	30	797
Document formatting	46	1060
Document previewing	32	938
Make program	24	1296
Compile	36	2900
GVX		
Idle GVX	5	48
Keyboard input	7	204
Mouse movement	5	52
Window scrolling	6	209

Cedar: Contention occurred on 0.01% - 0.1% of all entries to monitors;
GVX: Up to 0.4% when scrolling a window

Thread Paradigms

- **[Birrell91]**
  - Exploit concurrency on a multiprocessor
  - Make progress on several tasks at once
  - Provide network service to multiple clients simultaneously
  - Defer work to a less busy time

Defer Work (31% in Cedar)

- **Same as Birrell91**
- **Most common use on both Cedar and GVX**
- **Reduce client-observed latency**
- **E.g., "Interrupt handler" style critical threads**
  - E.g., keyboard-and-mouse watching thread

"Our programmers have become very adept at using"

Sleepers & Oneshots (26%)

- **Sleepers – repeatedly wait for a triggering event & then execute**
  - Triggering event is often a timeout
  - Do very little work when awoken
- **Oneshots – sleep, run, go away**
  - E.g., Used to detect double-clicking
- **Easy to use**
  - But sleepers are encapsulated in PeriodicalProcess module that use closures to maintain state instead of 100KB stack frames

Pumps (16%)

- **Related to Birrell91**
  - Creating pipelines to exploit parallelism on a multiprocessor
- **Components of a pipeline**
  - E.g., bounded buffers, external devices
  - Programming convenience, e.g., in Unix shell pipelines
- **Slack process**
  - Add latency to a pipeline to enable input & output coalescing
- **Easy thread use, as long as no critical timing constraints**
  - Challenging when timing constraints
  - User aware of bad performance in echoing & mouse motion
  - Buffer thread & X server thread problem – YieldButNotToMe

Deadlock Avoiders (10%)

- **Avoid violating lock order constraints**
  - Repainting after adjusting the boundary between two windows
  - Fork thread & release locks; forked thread grabs locks in order
  - Forking the callbacks from a service module to a client module
- **Very simple to use, but in context of very complicated locking scheme**

Task Rejuvenation (3%)

- **This thread is in trouble, let's make two of them**
- **Adds significantly to system robustness**
- **"controversial paradigm"**

Serializers (1%)

- **Serializer – a queue and a thread that processes the work on the queue**
  - Queue acts as point of serialization

Concurrency Exploiters (1%)

- **Same as Birrell91**
- **Threads created specifically to make use of multiple processors**
- **Hard to use in interactive systems**

Frequencies

Table 4. Static Counts

	Cedar		GVX	
Defer work	108	31%	77	33%
Pumps				
General pumps	48	14%	33	14%
Slack processes	7	2%	2	1%
Sleepers	67	19%	15	6%
Oneshots	25	7%	11	5%
Deadlock avoid	35	10%	6	3%
Task rejuvenate	11	3%	0	0%
Serializers	5	1%	7	3%
Encapsulated fork	14	4%	5	2%
Concurrency exploiters	3	1%	0	0%
Unknown or other[2]	25	7%	78	33%
TOTAL	348	100%	234	100%

Scheduling Priorities Hacks

Problem addressed: High priority thread waits on lock held by low priority thread that is prevented from running by a middle-priority CPU hog

- **Donated cycles: For per-monitor metalock (locks each monitor's queue of waiting threads), cycles are donated from a blocked thread to a thread that is blocking it**
- **Anyone can win the lottery: High-priority sleeper thread (SystemDaemon) wakes up and donates, using a directed Yield, a small timeslice to another thread chosen at random**
  - Ensures all ready threads get some CPU resources, regardless of their priorities

Common Mistakes

- **IF NOT(condition) THEN WAIT cv**
- **Timeouts introduced to compensate for missing NOTIFYs**
  - System can become timeout driven
  - Debugging is harder

When a Fork Fails due to lack of resources

- **Catch the error, but no good recovery schemes**
- **Wait for more resource, but unexplained delays for users**

Timeouts, Weak Memory Ordering, Serializing Threads

"We found many instances of timeouts and pauses with ridiculous values."

- Chosen with some particular now-obsolete processor speed or network architecture in mind

"We saw several places where the correctness of threaded code depended on strong memory ordering."

- **Introduce an additional thread to help manage concurrency and interactions with external I/O events**

Issues in Thread Implementation

- **Spurious lock conflicts**
  - Between a thread notifying a CV and the threads it awakens
  - Arises even on uniprocessor when waiting thread has higher priority than notifying thread
  - Solution: Defer processor rescheduling, but not the notification, until after monitor exit
- **Priorities**
  - Currently: Priority scheduler with hacks
  - "We do not regard this as a satisfactory state of affairs."
- **Sensitivity to Time-slice Quantum = 50 ms**
  - Sending of X requests from the buffer thread performs well
  - "Choice of scheduler quantum is not to be taken lightly."

Friday's Paper

Time, Clocks, and the Ordering of Events in a Distributed System

Leslie Lamport 1978