15441 Paper Summaries, November 11, 2001

SUMAN KUMAR NATH(sknath@cs.cmu.edu)

Using Threads in Interactive Systems: A Case Study

- *One-line Summary* : Examines thread usage in two systems at Xerox Parc - Cedar Programming Environment and Global View for X Windows (GVX) - in an attempt to detect common paradigms and common mistakes of programming with threads.
- *Main Points*
  - Two types of analysis - dynamic and static. Dynamic examines online thread usage and timing data, static examines code base.
    * Examining *dynamic* thread behavior revealed three classes of threads: (1)*Eternal*: Repeatedly waiting on Condition Variables, (2) *Worker*: Performing some activity, (3) *Transient:* Forked by long-lived threads - execute quickly and die (most numerous).
    * *Static analysis* found nine paradigms for working with threads: (1)*Defer Work* : (worker threads) - most common, used for interrupt handlers and critical threads, (2)*PUMPS*: pipeline components - can be used to exploit parallelism, but most often used for programming convenience, (3)*Slack* : Processes - type of PUMP that adds latency to pipeline in hopes of reducing total work done, (4)*Sleepers & One Shots* : (eternal threads) - long wait for triggering event and then brief execution (one shots exit after executing), (5)*Deadlock Avoiders* : (transient) - fork to avoid violating lock order constraints or some other form of deadlock, (6)*Task Rejuvenators* : cleanup and recovery from a crashed thread, (7)*Serializers* : Queue + thread that does work on queue, (8)*Concurrency Exploiters* : threads that take advantage of multiprocessors and explicit parallelism, (9)*Encapsulated Forks* : modules that encapsulate paradigm (delayed fork/MBQueue)
  - *Issues in Thread Use* : (1) Overhead Cost vs. Simplification & Concurrency (2) Easy Thread Use - Sleeper, One-Shot, Pump / work defferers (3) Hard Thread Use - Concurrency Exploiters, Pumps with timing constraints, slack processes, (4) Mistakes - Monitors (notify - at least one waiter wakens vs. exactly one waiter wakens)
  - Issues in thread implementation : (1) Spurious lock conflicts - waiting thread has higher priority than the notifying thread,(2) Priorities - system violates strict priorities, (3) Common Mistakes - i. WAIT code didn't recheck the predicate ii. People are probably used to Hoare's original monitors, so timeouts used to compensate for missing NOTIFYs
- *Flaws* :
  - The paper focusses on extremely large systems developed in the same place. As such its results are probably not very representative of the general scenario.
  - A third of GVX's threads weren't covered by any of their paradigms. (This suggests their set of paradigms is not complete.)
- *Comments* : The paper is an extremely valuable source of paradigms, pitfalls and important issues related to multi-threading. It would be very useful guide for anyone trying to develop large concurrent systems using thread.