Discussion Questions for BEAM Scheduling

B. LaChance and N. Ramsey, for COMP 250RTS

November 20, 2017

Required reading

In the BEAM book:

- Introduction
  - Read all of the *Scheduling* section.
- Scheduling chapter
  - Reductions (the first one)
    - Read the paragraphs through the end of "How Many Reductions Will You Get?"
  - The Process State
    - The states we need to understand are *running*, *runnable*, and *waiting*.
  - Process Queues
    - The Ready Queue
      - Read all of this
    - Waiting, Timeouts and the Timing Wheel
      - Read up until the paragraph starting with "Timers are handled in the VM by a *timing wheel*." It's neat but not fundamental.
  - The Scheduler Loop
    - Not important, but quickly skimmed.
  - Load Balancing
    - There are two techniques: *task stealing* and *migration*. Read both. In the migration section, beware of verbiage.

Enrichment reading

Pekka Hedqvist, *A Parallel and Multithreaded Erlang Implementation*, master's thesis, 1998.

At 20 years old, the implementation specifics likely aren't still in use. But parts of chapter 5 section 2, which describes the implementation, were fun to skim. The Task abstraction (numbered page 18), which is used to represent processes and ports, could be fun to compare to its modern-day incarnation. I also liked that chapter 2 gave a nice introduction to Erlang at the source level.

Jianrong Zhang, *Characterizing the Scalability of Erlang VM on Many-Core Processors*, master's thesis, 2011.

For scheduling, I really liked the overview in Chapter 3 Section 3.1. If you want to see deep details about how schedulers and migration paths, the rest of Chapter 3 Section 3 should serve you well.

Questions

Warmup

(1) If a scheduler's ready queue is empty, what should the scheduler do?

BEAM scheduling in context

(2) What do we know about where code can and cannot be interrupted

   (a) In a system built using Cormack's micro-kernel?
   (b) In BEAM?
   (c) In Lua?

   Is this order (Cormack, then BEAM, then Lua) the right order in which to consider these systems. Or at least *a* right order? Why or why not?

(3) Suppose we want to implement Erlang's model of preemptive multithreading on a Lua virtual machine:

   - We'll keep the idea of a Lua thread, but we'll eliminate `coroutine.suspend` and `coroutine.resume`.
   - We'll create a thread by passing a function to `thread.create`, and the thread will start running right away. When it can't affect any future computation, the thread will be garbage-collected.
   - Threads could communicate through locks and shared memory or through message passing—for our purposes, this detail won't matter too much.[1]

   If we want to implement Erlang's thread model on a Lua virtual machine, how much of a stretch will it be?

   (a) Easy peasy
   (b) We'll have to blow up the VM and start over
   (c) Somewhere in between

   Try to get a feel for the level of difficulty—but don't get too bogged down in details.

[1] To justify this claim, consider what happens in Cormack when a process P runs `v` on a semaphore on which process Q is waiting, and compare that with what happens in BEAM when a process P delivers a message to the mailbox of a process Q which is blocked in `receive`.

Load balancing

(4) How does BEAM "task stealing" compare with Cilk's "work stealing"?

(5) Cilk wants to keep all processors busy and to minimize overhead. Is that enough for BEAM, or does BEAM want more? Justify your answer.

(6) When it gets control, a BEAM scheduler seems to have only three choices, which it repeats indefinitely:

- Run a process
- Migrate a process in from another scheduler
- Migrate a process out to another scheduler

Each choice is governed by a complicated system of algorithms and data structures, involving "priorities," "task stealing," and "migration."

(a) What ends do you conjecture that priorities are intended to serve?
(b) What ends do you conjecture that migration is intended to serve?
(c) How would you compare BEAM and Cilk?

A warning about priorities

This note about the priority mechanism is from *Erlang Programming* by Francesco Cesarini and Simon Thompson:

> *Endless flame wars and arguments regarding process and priorities have been fought on the* `Erlang-questions` *mailing list, deserving a whole chapter on the subject. We will limit ourselves to saying that under no circumstances should you use process priorities. A proper design of your concurrency model will ensure that your system is well balanced and deterministic, with no process starvation, deadlocks, or race conditions. You have been warned!*

Bonus questions

Here are some questions on preemptive, cooperative multitasking, which we won't have time for.

(7) Why does it matter how many "reductions" are charged to a foreign function?

(8) When looking at Lua coroutines, we observed that a foreign (C) function can yield in the middle if the programmer is willing to CPS-convert the function and pass an explicit continuation to the Lua runtime. Does anything similar happen in Erlang? Why or why not?

(9) The BEAM book says, "The way the C code is structured, it is the emulator (`process_main` in `beam_emu.c`) that drives the execution and it calls the scheduler as a subroutine to find the next process to execute." How would you relate this design to Lua?

Rejected questions

Perhaps we've had our fill of locking.

(10) Does the ready queue need to be protected by a lock? How about the waiting queue?

(11) In task stealing, how does the locking work?