Tiny Kernel

By Martin Kelly

Tiny Kernel
6/22/2015

Tiny Kernel – design goals

- Good test coverage.
- Simplistic design (in the good sense) so that debugging is easy. Reject overengineering and overdesign.
- Make the API as simple as possible so the user doesn't have to worry about OS internal details. Keep complications inside the OS internals and don't let them leak into the overall design or API.
- Reasonably fast, although speed was not a main design goal.

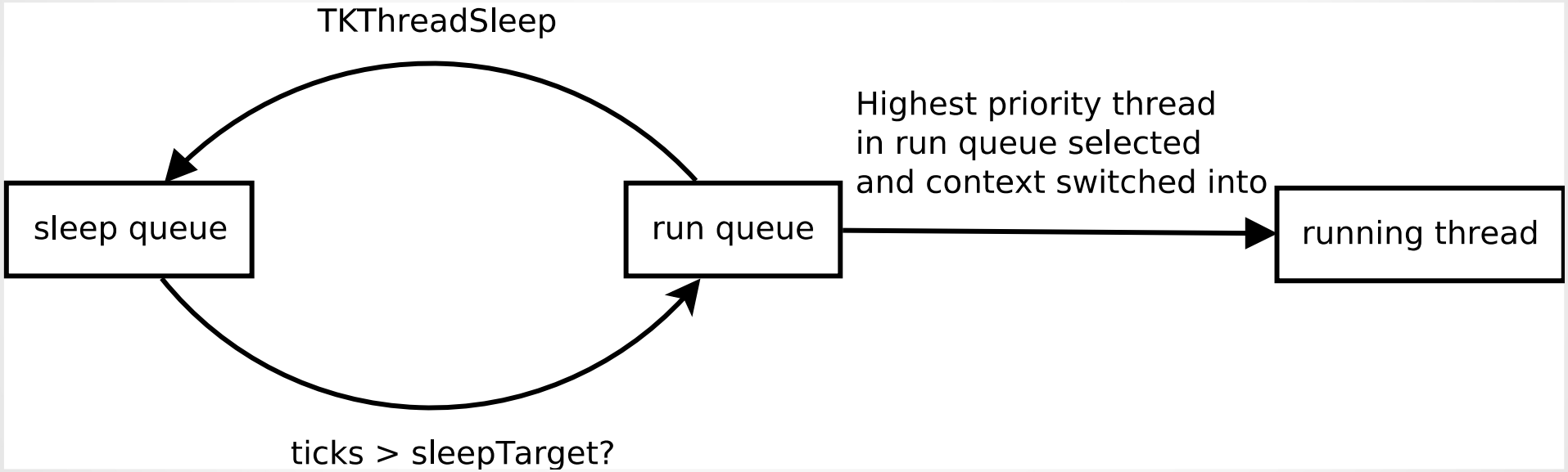
Tiny Kernel – tests

- Keep OS code in the *tk/* directory and hardware-dependent code in the *lpc/* directory.
- Make sure TK doesn't accidentally take an unneeded dependency on the LPC hardware.
- Make porting to other boards easier.
- Device drivers are an exception, as they live in *tk/* but are necessarily hardware-dependent.

Tiny Kernel – scheduler

- Thread states are implicit in which queue a thread lives. A thread is always in either the sleep queue or the run queue.
- This keeps the scheduler relatively fast and simple.
- See diagram (next slide).

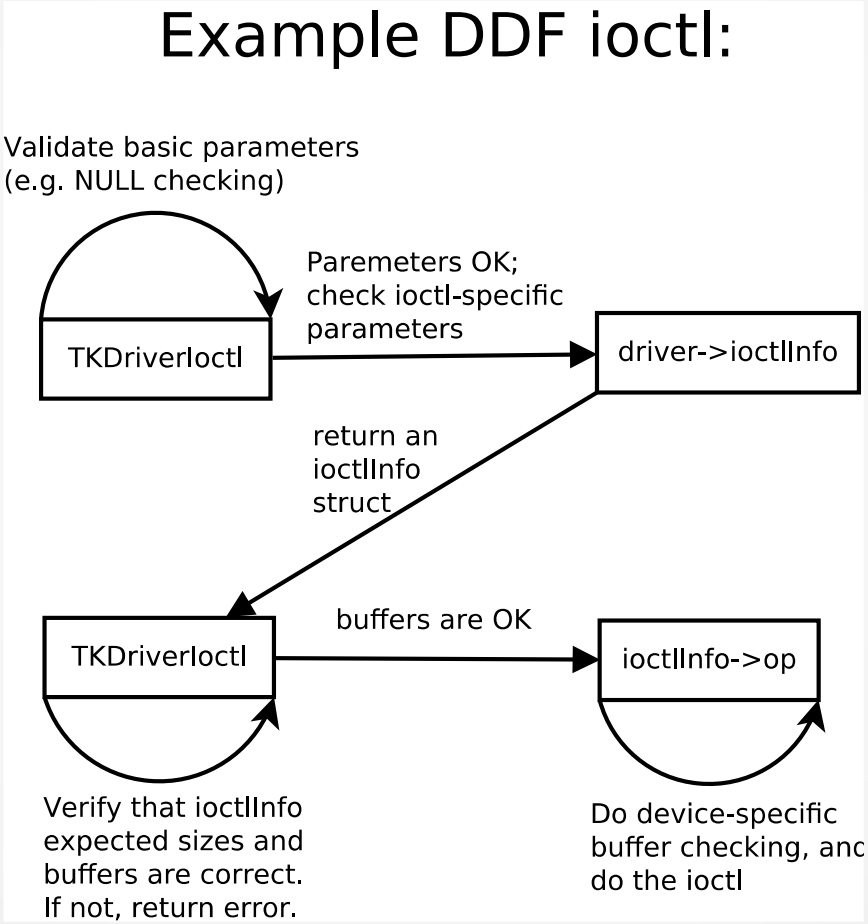
Tiny Kernel – scheduler



Tiny Kernel – device driver framework

- The device driver framework (DDF) aims to do as much as it possibly can in order to reuse code for input validation and minimize the amount of code required for writing a new device driver.
- Examples:
  - ioctls are structured so the device driver just declares how its input buffers should look and DDF does validation.
  - Power states are tracked by the DDF so that nonsense transitions (e.g. powerdown, powerdown again) can be rejected without the device driver having to worry about it.
- This means the driver can focus on the happy path.

Tiny Kernel – ioctl



Tiny Kernel – instrumentation

- Created generic instrumentation functions:

  void TKStartInstrumenting(struct TKInstrumentData *data);

  void TKStopInstrumenting(struct TKInstrumentData *data);

  These can instrument arbitrary sections of code in a nicely encapsulated way.
- Used these to instrument the scheduler.

Tiny Kernel – instrumentation data

- Scheduler found to take 8 us avg, 0.43% overhead with -O0.

  Improves to 3 us avg, 0.18% overhead with -O3.
- Given a 500 Hz timer interrupt, the theoretical expected overheads are:

  8 us * 500 / 1000000 = 0.40% overhead expected with -O0

  3 us * 500 / 1000000 = 0.15% overhead expected with -O3
- Given that:
  - We're rounding to the neareste tick in the overall percentage calculation
  - Instrumentation itself has some overhead

  These measurements are close enough to theory that I believe them.

Tiny Kernel – funny bugs

- Deadlock when timer Hz was 2000, but not when 500 or 5000. Came down to this code section in TKThreadSleep:
  - Enter a critical section
  - Remove this thread from the run queue
  - Add this thread to the sleep queue
  - Leave the critical section
  - Yield into the scheduler
- If the thread was scheduled away after removing itself from the run queue and before adding itself to the sleep queue, it would hold the critical section but be unschedulable. Deadlock!
- Fixed by disabling interrupts until the thread was added to the sleep queue.

Tiny Kernel – funny bugs

- Tried to validate the instrumentation code via the debugger.
- Found funny timings of several seconds to call TKSchedule.
- Assumed my instrumentation math had overflow or similar.
- Turned out, the time spent broken into the debugger was counting against me!
- This is why general-purpose OS'es often have two timings, one that corresponds to physical time and one that stops when the debugger stops (e.g. the tick)

Tiny Kernel – questions?

Any questions?

Thanks!

Thanks!

This was a really fun course

I finally fulfilled a childhood dream of writing my own OS