Power Management Implementation Plan

Bob Herlien
May 19, 2003

Phases

The implementation of Power Management for MOOS/CIMT is roughly planned for two phases.

Phase 1 includes two capabilities: the ability to put the CPU into sleep mode to save power, and the ability to prevent power overload (make sure we don't exceed the 12 amp capacity of the system). To implement the latter capability, we will estimate the current load and deny any instrument requests for power if the load exceeds the system capacity. However, we will make no attempt to manage the battery energy usage. This corresponds to the "Static Power Management" capability stated in the Functional Requirements doc.

Phase 2 will implement the "Power Prediction and Adaptive Power Management" section of the Functional Requirements doc. That is, it will attempt to measure and/or estimate the current battery state and allocate and prioritize power requests.

This implementation plan covers only Phase 1.

Power Policy Management

We will attempt to reuse, as much as possible the mechanisms already in place in SIAM and those designed by Tim Meese. The following assumes familiarity with those mechanisms.

Implicit Activity Detection

The implicit activity detection using the **Sentinel** thread is probably not effective, and will not be reused. Any well designed system (i.e. JVM), unless it's seriously overloaded, will allocate **some** execution time for even the lowest priority thread. Thus, the Sentinel thread will most likely never indicate activity.

Explicit Activity Detection

The main facility we'll use for detecting activity in CIMT will be the SleepRollCallListener facility. We will add SleepRollCallListeners for:

- The InstrumentService base class, which will be inherited by all Instrument drivers
- SensorLog
- RMI (see below)
- LeaseManager

We'll need to modify the InstrumentService class to:

- Add a SleepRollCallListener, deny SleepRollCallCallback if busy
- Either add a field and methods to indicate the estimated current used, or use the existing Current Limit as an estimate
- Request power from PowerManager before turning on instrument power

Issues:

- Should use Current Limit as estimate or add field?
- Need a separate estimate for power for EIA drivers?
- We'll use estimates for now; but in future, we'll determine how much power is actually being used by system and add the estimate for the requested instrument. Need interface to get actual power used by instrument.
- How do we determine how much power is used by CPU and other non-instrument parts of system?
- What to do if PowerManager denies power request? New exception?

RMI

In addition to determining activity due to servicing instruments, we need to determine if the system is busy servicing RMI methods.

We'll create a PowerManagedRMISocketFactory that produces PowerManagedSocket's and PowerManagedServerSocket's. The socket classes will have read() and write() methods that note the time of the last read or write. If no activity has taken place within some timeout (1 minute by default?) , then any subsequent SleepRollCallCallback will succeed. We'll have a method for setting that timeout.

Issue: RMI reuses sockets, so just having a socket open doesn't necessarily indicate activity.

System Activity

By convention, we'll declare that Linux system facilities (ssh, etc) are not used remotely for normal operations. All normal operations will take place through application-level facilities, presumably via RMI. Thus the RMI facility above will detect those operations. We'll have to add methods to NodeService to support any required user operations (e.g., connectToInstrument).

But – for debugging, special needs, and as a stopgap until all facilities are implemented via NodeService, we'll continue to use LeaseManager. If anyone wants to use Linux services(ssh, etc), they must first obtain a lease through LeaseManager.

Issues:

- httpd???
- PPP – does it keep SideArm alive by frequently asserting the UART interrupt?
  Long term, the SideArm communications controller should manage the PPP session. It will get the PPP keepalive traffic and only wake up (assert an interrupt to) the SideArm when there's actual IP traffic. What happens in near term?
- Add SleepRollCallListener to LeaseManager

Power Management Mechanism

We've demonstrated most of the capability we need to implement sleep/wakeup of the CPU. To implement, the PowerManager will call the SleepRollCall list, and if all say it's OK, it will go into sleep mode.

Issues:

- Need to know how long to sleep. Interface to scheduler?
- Need to share the serial port that goes to the MSP430 with the Environmental service (where is this in the sources?) How? Either a superclass from which each class (PowerManager, Environmental) can lease the port, or PowerManager asserts suspend/resume on Environmental
- The MSP430 can wake us up due to environmental conditions in addition to the sleep time. Need to check and do something? First cut – only enable timed wakeup on MSP
- We can also wake up from UARTs and other hardware. Need to check and do something?
- Ethernet doesn't have power management support, so it's broken when we wake back up. Workaround: ifdown/ifup, or don't use Ethernet.
- Need to manage power to comm devices (FreeWave, GlobalStar)? If so, how?