A Robust Command, Communications and Data Acquisition System For Autonomous Sensor Platforms Using The Data Transport Network

Todd Valentic
 Center for Geospace Studies
 166 Encinal
 Menlo Park CA 94025
 todd.valentic@sri.com

Need
A reliable, low-power, computer system for collecting and transmitting data from multiple sensors at remote sites that are operated autonomously for long periods of time supporting the National Science Foundation’s Arctic Research Support and Logistics Services program.

Approach
A low power, temperature rated single board computer (SBC) is selected, upon which we run a version of the Linux operating system tuned for embedded systems. The application layer uses the Data Transport Network as a framework to organize the collection of programs used to collect data from scientific instruments. The entire system runs out of the on board flash memory, so there is no need for a hard drive. The entire system runs automatically without user interference. The Data Transport framework is used to organize a set of programs in the system that are responsible for management and data transmission over the Iridium satellite network.

Data Transport Network
The Data Transport Network is a system for designing robust field instrumentation that integrates the Data-Transport scientific work system health monitoring, data processing and distribution of real-time results over unstable and bandwidth limited networks. The system is built around a set of message servers that provide a store and forward approach for data transport. The Data Transport framework is used to organize a set of programs in the system that are responsible for management and data transmission over the Iridium satellite network. A number of fail safes are implemented to allow the system to recover from unexpected situations.

Tincan Linux
We roll our own embedded Linux distribution. At the core is the 2.6.34 kernel, patched to support the Telos B SBC. Most of the common Unix commands are provided by BusyBox and the lightweight uClibc C library. A typical system, including the Python programming language, has a footprint of 400kB.

The entire build process is controlled by Buildroot:
- Makefile based
- Builds the cross-compiler tool chain
- Package selection
- Automatic packaging
- Creates root file system

The entire system runs out of the on board flash memory, which uses HFS2 to prevent data loss during power outages and provides wear-leveling.

Hardware Stack
We use the TS-7200 single board computer (SBC) from Technologic Systems. It is temperature rated to -40C and is capable of running a standard Linux kernel.
- 200MHz ARM9 CPU
- 128MB SDRAM
- 2 USB ports
- 30 DI lines
- 2 12-bit A/D
- 2 Watchdog timers
- 200MB ARM9 CPU
- 128KB NAND Flash
- Ethernet
- 3 serial ports

The total power consumption is 1W, which can be reduced to 0.25W when idle. A custom power control board is used to turn instruments on to conserve power and includes signal conditioning to monitor power usage.

Scheduling
Each monitor has a set of schedules that determine when it runs. A simple schedule has a sample window within which data are periodically collected. Multiple schedules are supported, ordered by a priority value, allowing for seasonal or campaign overrides.

More sophisticated schedules are supported, such as sampling around solar local noon. Instruments, such as the data transport window, make use of this feature. The solar angle is computed from the GPS position and dynamically fed into the schedules.

Resource Management
The resource manager service is the key to optimizing the low power performance of the system. It maintains a scoreboard of the global system state, tracking which devices are powered on. When a new program needs a resource, a request is placed to the resource manager. The resource (an IO pin, the Ethernet port, etc.) will be enabled when the program is finished, it release the resource. If another program is still using the resource, it will remain allocated until no other programs are using it. The benefit of this approach is that each client program remains independent of each other, yet the resource usage is fully integrated.

Failsafes
Recovering from unexpected problems is necessary for long-term unattended operations. We use a layered approach:
- Kernel parameters set to reboot on panics and oopses
- Hardware watchdog routed to alert
- Software watchdog (via xon) reboots if data transfers stop
- Periodically power on modem for listen only

A flag can be sent to the system instructing it to keep the Iridium modem powered on, at which point you can manually log in to the system.

Fall AGU 2012 - C13E-0669