WIND RIVER

Power State Management in Automotive Linux

Table of Contents

Executive Summary 1
Vehicle Power States and Device Energy Management 1
Key Linux/Open Source Software Technologies for Automotive 2
- CAN and MOST Drivers and Protocols 2
- D-Bus 3
- Energy Management 3
- Initng: Next-Generation Init System 3
- System Health Monitoring – Monit 4

Conclusion 4

Executive Summary

The automotive environment presents device original equipment manufacturers (OEMs) with a unique set of design challenges centered on power and energy use and management. Self-powered and with substantial battery capacity, today's vehicles occupy a midpoint between stationary devices powered by constant AC power and fully mobile devices such as 3G phones and media players. A viable automotive Linux platform must accommodate this hybrid position and support vehicle-centric and device-specific requirements for both power state and energy management subsystems.

This document examines the challenges presented by supporting intelligent power state management for in-vehicle systems, including designs for automotive infotainment, telephony, navigation and maintenance, and available technologies to meet those challenges.

Vehicle Power States and Device Energy Management

Standalone portable consumer electronics (CE) devices such as mobile phones, media players, and digital cameras require rich and robust energy management to draw the optimum mix of performance and longevity from on-board batteries.

Terminology

As automotive, consumer electronics, and embedded software and hardware engineering intersect, technical traditions and vocabulary for in-car systems design begin to overlap. The following terms are defined for use in this document.

Power state management: Automakers and their supply chains use *power management* to describe the state of vehicles and in-vehicle systems relative to their policies for and use of electric power flowing from a vehicle's alternator and battery. To minimize confusion between familiar automotive terms and current embedded and mobile terms, in this document *power state management* is used to describe the software infrastructure to support vehicle power states and transitions.

Energy management: Silicon suppliers, device manufacturers, and Linux kernel developers use a range of terms to describe optimization of power and energy utilization at the chipset and device levels. These technologies and software techniques variously involve scaling CPU clock speeds, adjusting device voltage levels, and invoking, maintaining, and exiting from CPU sleep modes, all to conserve power consumption and energy usage over time. To avoid confusion with terminology used in the automotive industry, in this document *energy management* describes device-level power and energy conservation efforts.

These types of devices can perform in a varied set of operational states that impact energy availability and management but typically only need observe a two-way distinction between connected/charging and disconnected/discharging power states. CE device designers typically make no allowance for energy availability and management of charging power sources, which can include AC and converted DC, USB, automotive and airplane DC, and even solar charging.

Vehicle Power State	Description	Device Power Source	Device Energy Management
Ignition Off	Vehicle powered off by key switch; vehicle battery may be available but use should be restricted to conserve energy	Device battery	Device-centric policies to conserve battery (e.g., sleep)
Accessory Mode	Accessories (radio, navigation system, lighter, etc.) powered on by key switch; policy and user sensibility determines energy use	Vehicle battery	Policies to conserve car battery (dim screen, sleep, etc.)
Ignition On	Engine on, alternator supplying DC and charging vehicle battery	Vehicle generation	Policies for best performance; vehicle system powers device and charges device battery
Diagnostic Mode	Vehicle in service bay, device (optionally) used to display diagnostic information	Vehicle battery or external source	OEM/tier supplier specified

Primary power management states and energy management implications

By contrast, automotive Linux must account for the power state of the system that supplies power to in-car devices, as illustrated in the above table.

These primary power states form the basis for designing state machines with the following event types driving state transitions:

- Key switch transitions among Off, Accessory, On, and Start positions
- Vehicle doors opening and closing and door locks engaging
- Seat sensor and seatbelt closure switches showing occupants present
- Events from wireless key fobs
- Remote input from satellite systems and services
- Vehicle security enabling and disabling systems in response to user input and system challenges
- Special input to initiate diagnostic modes ("magic" key chords and under-the-hood switches and connections)
- Other user input from dashboard and device-based switches and touch screens

To support this kind of power state management paradigm, an automotive Linux platform must integrate a mix of standard and emerging Linux capabilities for CE and embedded applications, functionality borrowed from desktop and enterprise Linux, and facilities unique to automotive systems design not yet mainstream in Linux.

Key Linux/Open Source Software Technologies for Automotive

Fortunately, as an instance of an end-to-end OS, the use of which spans from deeply embedded to desktop to server, automotive Linux can leverage a wide range of existing code and technologies.

CAN and MOST Drivers and Protocols

Automotive connectivity to dashboard-embedded devices is complex, requiring support for determinism and interference shielding. To serve the core requirements of automakers and automotive device OEMs, automotive Linux must support industry-standard interconnects and data transports that exist in real-world vehicle systems. Such interfaces apply directly to power state management as the transport for many of the events that drive power state transitions. The two most relevant are CAN (Controller Area Network) and MOST (Media Oriented Systems Transport).

Several implementations of CAN exist as open source software. The most prevalent and successful is SocketCAN, contributed by Volkswagen and accepted into the Linux kernel mainline version 2.6.25. For additional details on SocketCAN, visit the project site at http://developer.berlios.de/projects/socketcan/.

To date, there have been no open source projects targeting the relatively newer MOST interface. As such, automotive Linux looks to proprietary implementations from commercial software vendors. For example, in its automotive Linux open source middleware layer, Wind River chose to integrate the MOST stack provided by SMSC, a supplier of semiconductor solutions for the automobile. Learn more at http://www.smsc-ais.com/AIS/.

D-Bus

Linux offers developers a rich toolbox of interprocess communications (IPC) mechanisms. Given the need to inform applications about power state transitions and to let applications affect power state policy and drive transitions, the most logical choice for automotive Linux is the Linux Desktop Bus (D-Bus).

D-Bus is an open source software project that originated with the FreeDesktop, an open source project working on interoperability and shared technology for X Window System desktops. Today D-Bus is the native IPC for the K-Desktop Environment (KDE) and is increasingly prevalent on the GNOME desktop as well (replacing Benobo). D-Bus is also the preferred IPC for a range of mobile/embedded Linux platforms including LiMO, Maemo/Hildon, Moblin, and OLPC.

D-Bus lets applications register themselves and publish the services they offer. D-Bus then gives subscribing applications the ability to discover which services are available and to respond to both application-created and system-level events. In automotive Linux, D-Bus serves to deliver power state transition events to relevant applications, including events emanating from CAN and MOST bus messages. D-Bus events can also be used to drive Linux run-level directly (see the "Initng" section below).

Energy Management

In addition to uses ranging from deeply embedded to desktop to server, Linux runs on CPU architectures as diverse as ARM, Intel Architecture, MIPS, and PowerPC. This diversity has led to the evolution of a range of CPU and device-based energy management schemes. Some but not necessarily all of the power management frameworks and energy management technologies of the desktop and server paradigm have been adopted in automotive Linux.

Developers and integrators of an automotive Linux platform can use any of the following energy management paradigms to meet the particular capabilities presented by the choice of embedded CPU and the rest of the bill of materials:

- Advanced Configuration and Power Interface (ACPI): To take advantage of notebooks, desktop computers, and blades with BIOS support for energy management, Linux kernel developers enabled support for the ACPI specification developed by Hewlett-Packard, Intel, Microsoft, Phoenix, and Toshiba. ACPI is primarily available with PC motherboards, single board computers, and blades built with Intel and AMD x86 architectures. ACPI support is standard in the Linux kernel. See http://www.lesswatts.org/projects/acpi/ for more information.
- Advanced Power Management (APM): The original Linux energy management scheme, this targets notebooks and desktop machines. APM has been retargeted to support a range of CPUs and is employed on many embedded applications. APM is also standard in the Linux kernel.
- Dynamic Power Management (DPM): To meet the needs of global consumer electronics manufacturers, the IBM Austin Research Lab, the Consumer Electronics Linux Forum (CELF), and CELF members specified and developed implementations of DPM. DPM features extremely rapid transitions (1–10 milliseconds) among power usage states and includes a scaling CPU clock, voltage, and other attributes. DPM originally targeted PowerPC and ARM-based devices but is otherwise architecture-independent. Learn more at http://dynamicpower.sourceforge.net/.
- Intelligent Energy Management (IEM): ARM Ltd., the licensors of architecture and intellectual property underlying ARM processors, offers its own IP cores for energy management. The software designed to drive that IP is IEM. Not all ARM-based silicon suppliers take IEM from ARM Ltd.; in many cases licensees use their own CPU-specific schemes instead, driving them with DPM or APM. Learn more about IEM at http://www.arm.com/products/esd/iem_home.html.

Initng: Next-Generation Init System

UNIX-type operating systems, including Linux, present developers and users with a series of "run levels" that accommodate varying modes of use. By default, Linux executes in one of five levels: Single User (1), Networked Single User (2), Text-Based Multi-User (3), X-Windowed Multi-User (5), and Shutdown and Reboot (6). Desktop Linux users will be familiar with running at level 5, and embedded systems developers can and do deploy systems running at levels 1, 2, 3, and 5. This level scheme, implemented by sysvinit (originally from SVR4 UNIX), while suited to desktop and server applications, does a poor job of representing the power state management needed by automotive Linux and of processing the needed agile transitions among states.

Initng has emerged as a strong alternative to sysvinit, not just in embedded applications but in many enterprise applications as well, and is being included in many desktop and server Linux distributions. Initng offers developers a range of useful capabilities:

- Parallelized service startup and concurrent execution (based on dependencies)
- Reuse of sysvinit scripts
- Plug-ins to manage services (add commands, options), react to signals, add states, etc.

- Service configuration with respawn options, dependencies, nice, delays, environment, chdir, chroot, etc.; available graphical application for configuring and controlling run levels and services
- Automatic respawning of prematurely defunct daemons
- Available patches to respond to D-Bus events to change run levels

To learn more about initng, visit http://www.initng.org.

System Health Monitoring – Monit

The scope of applications addressed by automotive Linux today covers in-car communications and infotainment. These are highly differentiating and key to a positive user experience but not mission- or life-critical functions such as engine control or antilock braking. Nonetheless, automotive OEMs benefit from making these systems as reliable as possible, in particular, with the monit project.

Monit is a utility for managing and monitoring processes and resources on UNIX/Linux systems. As with initng, monit was originally intended for data center applications but fits nicely into automotive Linux. Applications can use monit to do the following:

- Start or restart a process that terminates or fails to launch (as part of power state transitions)
- Monitor files, directories, and file systems for changes in timestamp, checksum, or file size
- Check TCP/IP port connections and monitor remote hosts

In practice, monit lets automotive OEMs and integrators do the following:

- Maintain free RAM and persistent storage (disk, flash, etc.) within predetermined ranges
- Instigate action to warn users and free memory as user-managed content (e.g., music, videos) occupies growing percentages of storage
- Implement policies for terminating processes in the face of RAM and other resource shortages (instead of arbitrary "reaping")
- Run garbage collection in Java or custom run-time platforms
- Log engine and system faults into infotainment system storage and manage those logs
- Drive power state transitions based on available resources

You can learn more about monit at http://www.tildeslash.com/monit/.

Conclusion

Power state management in automotive Linux is not a monolithic capability. Supporting real-world requirements of automakers and automotive OEMs entails leveraging and carefully integrating a mix of open source and commercial software components. As part of a comprehensive automotive platform, power state management needs to transport and respond to power state events from both vehicle and application-internal sources, representing both normal and exceptional operating states. With its synthesis of enterprise and embedded features and flexible system architecture, Linux provides an ideal platform to meet these diverse requirements and enable a range of advanced next-generation in-car systems.