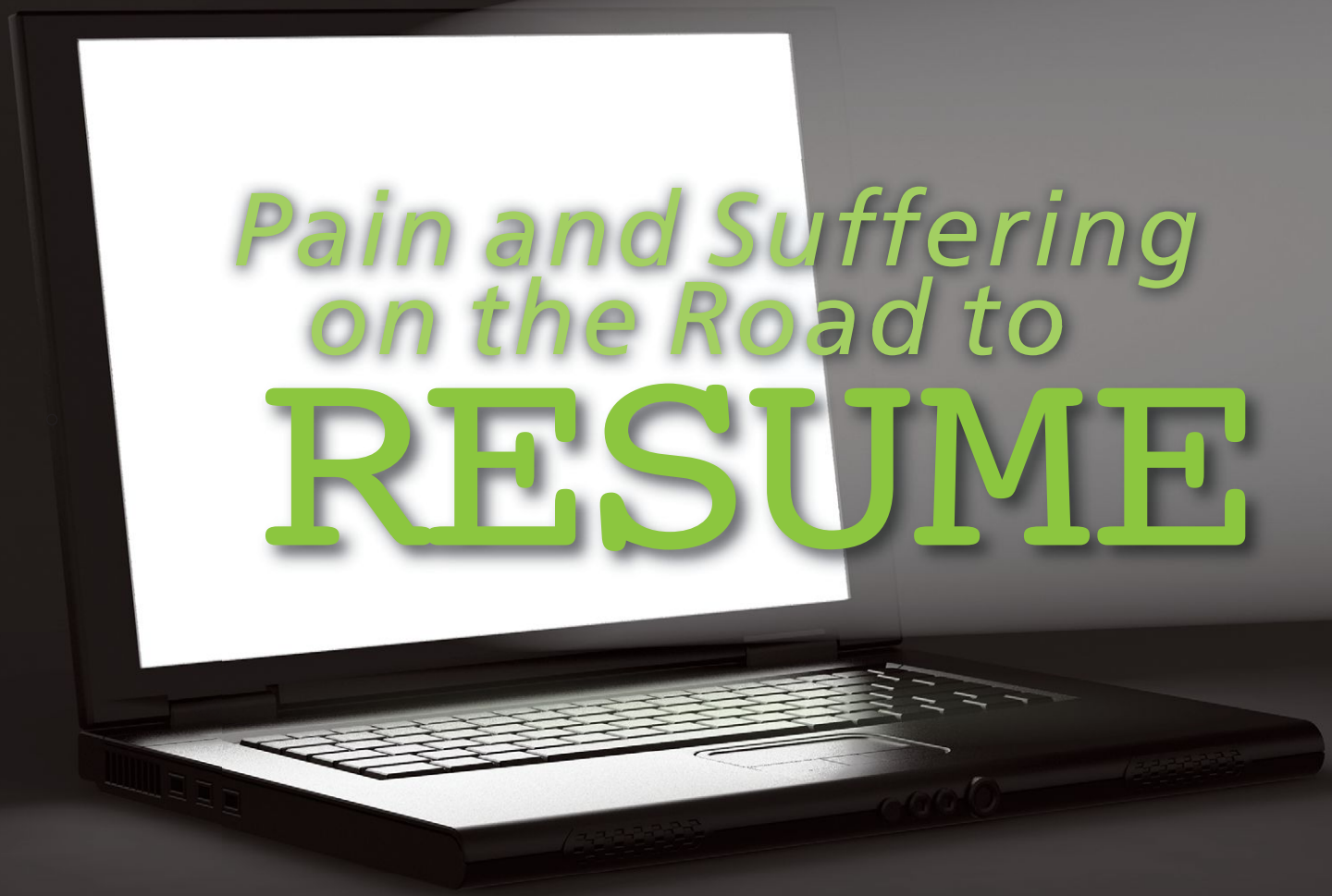
SEE TEXT ONLY

BY JOHN BALDWIN



Pain and Suffering on the Road to RESUME

Most consumer computing devices today are mobile devices. These devices are not tethered to a power outlet, but, instead, are able to run on battery power alone. One of the tasks of the operating systems running on mobile devices is to maximize the runtime while on battery. A primary way of achieving this goal is to power off components of mobile devices that are not in use (for example, powering off the screen on a smart phone).

Stock FreeBSD does not currently run on several classes of mobile devices such as phones. However, FreeBSD has run on another class of mobile device for many years: laptops. Traditionally, laptops have not offered very fine-grained power management, but they do permit the entire system to be placed in a low-power state when it is not being actively used. Transitioning a laptop into a low-power state is called "suspend" and returning the system to the fully-operational state is called "resume."

FreeBSD only supports suspend and resume on x86-based systems. In addition, while suspend and resume is supported on both desktop and laptop x86-based systems, development effort in FreeBSD has only focused on laptops. Even then, FreeBSD only successfully suspends and resumes on a subset of x86-based laptops.

Suspend and Resume on x86

Support for suspend and resume in x86-based systems has evolved over time.

The first standard for system-wide power management on x86 is called Advanced Power Management (APM). It supports two different low-power states: standby and suspend. The standby state was able to return to the fully-operational state more quickly than the suspend state. However, the suspend state used less battery power as it turned more internal devices off. While it is possible for the OS to control the power usage of individual devices, this is not required and the BIOS is generally responsible for saving device state when suspending and restoring the saved state when resuming. FreeBSD's APM support relies on the BIOS to manage the power of individual devices during suspend and resume.

APM was replaced by the Advanced Configuration and Power Interface (ACPI).

ACPI is a superset of APM as it includes several other components for managing devices than just power management. ACPI also adopted an expanded set of system sleep states (see Table below).

SLEEP STATE	DESCRIPTION
S0	System is fully operational
S1	No components are powered off
S2	Only CPUs are powered off
S3	All devices other than RAM are powered off
S4	All devices are powered off, state saved
S5	All devices are powered off, state lost

The S1 state in ACPI is similar to APM's standby state, and the S3 state is similar to APM's suspend state. ACPI also includes an S4 state (known as "hibernate") similar to S3 except that the contents of RAM are saved on a hard drive (or similar device). Resuming from S4 requires loading this saved copy back into RAM. Finally, ACPI adds an S5 state which allows the operating system to power off a device. In conjunction with this change, ACPI notifies the operating system when the power button is pressed by the user to give the operating system time to perform an orderly shutdown before the system is powered off.

In contrast to APM, ACPI requires the host operating system to actively manage the power state of many devices in a system during suspend and resume. For example, the OS is required to save and restore PCI config space register values for all PCI devices in the system for sleep states such as S3 that power off devices.

To aid the adoption of hibernation, early systems supporting ACPI's S4 sleep state included an option for the BIOS to assist with saving and restoring the contents of RAM. This option is called S4BIOS. When S4BIOS is supported, the BIOS saves the contents of RAM (usually to a dedicated hard drive partition owned by the BIOS) and restores the contents of RAM during resume from S4. When S4BIOS is not used, the OS is responsible for saving the contents of RAM to some type of OS-managed nonvolatile storage during suspend. During resume, the system powers up and follows the normal bootstrap process. The OS bootstrap is required to recognize a boot as being a resume from S4, locate the saved copy of RAM, and load the saved copy into RAM rather than performing a normal bootstrap.

FreeBSD's ACPI Support

While FreeBSD does include limited support for APM, the majority of modern x86 systems only support ACPI. FreeBSD first supported suspend and resume via ACPI for i386 in FreeBSD 5.0. Support for amd64 (x86_64) was first available in FreeBSD 8.0. The S1 and S3 sleep states are supported, but many laptops only support S3 as S1 offers little power savings. FreeBSD does not support native S4. FreeBSD is able to hibernate on systems supporting S4BIOS, but modern laptops do not support S4BIOS.

System Control Nodes

FreeBSD creates several system control (sysctl(8)) nodes related to suspend and resume under the `hw.acpi` node. Some nodes provide information while other nodes are used to control suspend and resume behavior.

The definition list format is below.

`hw.acpi.supported_sleep_state`
List of ACPI sleep states supported by the host.

`hw.acpi.s4bios`
Indicates if the host supports S4BIOS.

`hw.acpi.sleep_delay`
Number of seconds to pause during a suspend operation after all devices have been suspended, but before the ACPI driver asks the firmware to enter the requested sleep state. Defaults to 1 second.

`hw.acpi.reset_video`
Can be set to 1 to request the kernel to use a legacy BIOS interface to reset the graphics adapter during resume. This generally does not work on modern laptops but did fix issues on some older systems. Defaults to 0 (off).

`hw.acpi.standby_state`
The ACPI sleep state to enter when a userland application requests a transition to the APM "standby" state via the legacy APM interface. Defaults to S1 if the host supports S1.

`hw.acpi.suspend_state`
The ACPI sleep state to enter when a userland application requests a transition to the APM "suspend" state via the legacy APM interface. Defaults to S3 if the host supports S3.

`hw.acpi.lid_switch_state`
The ACPI sleep state to enter when the user closes the lid on a laptop. Defaults to NONE.

`hw.acpi.sleep_button_state`
The ACPI sleep state to enter when the user presses the suspend hotkey on a laptop keyboard. Defaults to the lowest suspend sleep state (S1 - S4) supported by the host.

`hw.acpi.power_button_state`
The ACPI sleep state to enter when the user presses the power button on the host. Defaults to S5.

On a Lenovo ThinkPad X220 the initial values of these nodes after boot are:

```
hw.acpi.supported_sleep_state: S3 S4 S5
hw.acpi.s4bios: 0
hw.acpi.sleep_delay: 1
hw.acpi.reset_video: 0
hw.acpi.standby_state: NONE
hw.acpi.suspend_state: S3
hw.acpi.lid_switch_state: NONE
hw.acpi.sleep_button_state: S3
hw.acpi.power_button_state: S5
```

This indicates that this system supports S3 (suspend), S4 (native hibernate), and S5 (soft-off). Pressing the sleep button suspends via S3. The power button triggers a graceful shutdown and power off. Closing the lid doesn't result in any action. The laptop can be configured to suspend via S3 when the lid is closed by setting the `hw.acpi.lid_switch_state` node to S3:

```
# sysctl hw.acpi.lid_switch_state=S3
hw.acpi.lid_switch_state: NONE -> S3
```

Userland Utilities

FreeBSD provides interfaces for userland utilities to request system suspension or shutdown. Third party applications such as window manager widgets can use these interfaces to permit user-initiated suspensions.

Several base system utilities can also request sleep state transitions. The shutdown(8) and halt(8) utilities accept a `-p` flag to request that the system be powered off via S5 after a clean shutdown. The poweroff(8) utility is an alias for `halt -p`. In addition, the acpiconf(8) utility requests a sleep state from S1 to S4 via the `-s` flag.

Device Driver Support

Device drivers are also responsible for saving and restoring state during suspend and resume. Bus drivers are responsible for saving bus-defined state for each child device before the system is suspended and restoring that state upon resume. Leaf device drivers are responsible for saving and restoring device-specific state. Device drivers should also quiesce active devices when preparing for suspend and restart any paused activity when resuming. Two bus drivers that require explicit suspend and resume support are the ACPI and PCI bus drivers.

The ACPI bus driver primarily manages power states of child devices and power resources. During a suspend request, the ACPI bus driver uses information from ACPI's device tree to place any ACPI devices supporting low power states into a firmware-indicated power state while preparing to suspend. These devices are restored to full power upon resume. In addition, the ACPI device tree describes the relationship between power producers and devices consuming power. If all of the devices that draw from a given power provider are turned off while preparing for suspend, then the ACPI bus driver will turn off the power producer. During resume, the ACPI bus driver restores power to the power producer before any of the associated devices are restored to full power.

Similar to the ACPI bus driver, the PCI bus driver is responsible for placing PCI devices into low power states (using firmware hints to choose specific states when available) while preparing to suspend and then restoring devices to full power on resume. Unlike ACPI, the PCI bus does not define power producer and consumer relationships. If PCI devices depend on discrete power producers, that relationship must be described in ACPI's device tree and managed by the ACPI bus layer. As a result, the ACPI and PCI drivers do share some joint responsibility for power management of PCI devices.

Unlike the ACPI bus driver, the PCI bus driver is also responsible for saving and restoring standardized configuration registers of devices. While preparing to suspend the system, the PCI bus driver takes a snapshot of all of the standard PCI configuration registers including registers to manage resource allocation, interrupt routing, and device control. The PCI bus driver saves the existing values of these registers for each PCI device before the device is placed into a low power state. (Once a device is placed into a low power state it no longer responds to requests to read or write to most configuration registers.) During resume, the PCI bus driver restores the value of these registers after the device has been restored to a fully-powered state.

All bus drivers including both ACPI and PCI invoke two methods in each device's driver to give leaf device drivers a chance to save and restore device-specific state. The `device_suspend` method is invoked by a bus driver while preparing to suspend the system. The `device_resume` method is invoked by a bus driver during resume.

The `device_suspend` method is invoked by the bus driver on each device before placing the device in a low-power state. Device drivers use this method to save a copy of device-specific registers as well as to pause any current activity and disable any active interrupts. For example, a driver for a network interface card will disable the card's receiver and disable any interrupts. In addition, if a device is able to wake the system from suspend, the suspend method should enable this functional-

ity if configured. Some network interface cards permit a system to be awoken via special network packets using a facility known as Wake On LAN (WOL). If the administrator has requested WOL for a specific network card via ifconfig(8), the driver's suspend method should enable WOL.

The `device_resume` method is invoked by the bus driver on each device after the device has been restored to a fully-powered state. Device drivers use this method to restore any device-specific registers and resume any previously-paused activity. Devices supporting wake functionality may also need to disable that feature. For a network interface card driver, a resume method will typically restart transmission of any pending packets and enable the card's receiver.

Debugging Suspend & Resume

Despite all of the work performed on suspend and resume in FreeBSD, suspend and resume only work on a limited set of systems to date. Unfortunately, an issue with a single device driver can cause the suspend and resume of an entire system to fail.

Diagnosing the cause of a suspend or resume failure is rather difficult. Typically, when suspend or resume fails, the laptop just hangs with a powered-off screen. Since the screen is powered off, one cannot use messages displayed on the console to narrow down the cause for the failure. If a laptop includes a non-USB serial port, then one can log messages over the serial port (by using a serial console for example). However, modern laptops generally do not include serial ports. Some laptops do provide a virtual serial port (such as via Intel's AMT), but those serial ports do not reliably resume to working operation during a failed resume attempt in the author's experience.

In some cases, a laptop will mostly resume but leave the screen powered off. The keyboard will still respond to input, however, and if one has suspended in single user mode, one can type blindly to run commands to determine if the laptop is in this state. For example, typing 'poweroff' followed by Enter when in this state should result in a spike in hard drive activity followed by the machine turning off a few seconds later.

This particular failure mode is in fact the most common type of resume failure in the author's experience. If the laptop's graphics adapter is supported by a GPU driver such as i915kms.ko, then loading this driver before suspending will usually resolve this particular failure case.

Additional suggestions for debugging some other suspend and resume failures can be found at https://wiki.freebsd.org/SuspendResume.

Future Directions

There is certainly room for improvement and future work on suspend and resume. Two main areas of work include support for ACPI's S4 state (hibernate) and ACPI's new low-power idle states.

FreeBSD does not yet support native S4 or hibernate. Unlike the S3 suspend mode, resume from S4 follows a normal power-on process prior to resuming the operating system. This leaves the hardware in a state similar to initial boot and depends on the firmware's bootstrap to initialize devices to a known state. This should make resuming from S4 less susceptible to device-specific issues such as failing to restore power to the screen. Hibernate also offers greater power savings compared to S3. While suspended in S3, a laptop still draws power. If the battery is exhausted while in S3, the saved state in memory is lost. With S4, the system does not draw power while suspended and can remain suspended indefinitely. In addition to supporting native S4, FreeBSD should also provide better support for suspend and resume policy management, such as auto-suspending when a laptop's battery is low.

Beyond support for hibernation, the next major challenge for FreeBSD will be support of ACPI's new low-power idle states. ACPI 6.0 introduces a new model for system power management. Rather than using monolithic, system-wide sleep states, low-power idle states encourage more fine-grained power control of both devices and processors. In this model, the operating system seeks to minimize power usage by leaving both devices and processors in low-power states whenever possible. In addition, when responding to wake events such as a WOL packet, the operating system should only restore power to those components of the system needed to handle a specific event leaving other components in a low-power state. While current systems still support ACPI's traditional system-wide sleep states, the expectation is that low-power idle states will eventually supplant system sleep states on systems such as laptops and desktops. This will require a more pervasive understanding of power usage and power producer/consumer relationships through all of FreeBSD's device driver infrastructure as well as alternate approaches to thread scheduling at a minimum. ●

JOHN BALDWIN joined the FreeBSD Project as a committer in 1999. He has worked in several areas of the system, including SMP infrastructure, the network stack, virtual memory, and device driver support. John has served on the Core and Release Engineering teams and organized several FreeBSD developer summits.