CELF_Specification_V_1_0_R2

CE Linux Forum

1 Introduction

The purpose of this document is to define and specify features and technologies intended to make Linux a better base for developing consumer electronics products. This document is the result of the work of the CE Linux Forum's technical working groups, addressing technical requirements in six specific areas of Linux where barriers to use in consumer electronics products had been identified. The specification represents the consensus of the forum members in those working groups.

Some of the specifications in this document describe features for which implementations already exist (and, indeed, are already integrated into the 2.6 version of Linux). The purpose of including these specifications is to provide rationale for the desirability of these features in the consumer electronics space.

Other features specified in this document are forward-looking, describing features that have not yet been implemented, or for which the implementations are not yet finished or mature. The purpose of these specifications is to provide developers with guidance in creating or augmenting these features.

The rationale for each specified feature is presented, to communicate and clarify the requirements in the consumer electronics space that affect the design decisions for the (existing or future) implementation of that feature.

This document is targeted at multiple audiences. First, it is primarily directed at system developers who wish to create or extend individual technologies in Linux. The specification will help enable collaboration among them and between them and CE Linux Forum members on development of these technologies. Many of these developers are employees of the member companies of the CE Linux Forum itself. However, this document is also open for public review and comment. Therefore, a second audience for the document is members of the open source community who are interested in embedded use of Linux, or in the specific technologies mentioned herein. For this group, this document should provide rationale for individual desired features, provide input on specific requirements and interfaces, and foster dialog on their design and use.

The scope of this document, in this first release, is limited (with one exception - see Section 5.7) to the Linux kernel. Note that this specification is meant as a guideline for Linux development, not as a formal standard to which an implementation should be claimed to conform.

The CE Linux Forum welcomes any and all feedback and inputs in response to this document. Please direct your comments to the forum's public mailing list, at: celinux-dev@tree.celinuxforum.org.

2 Global Terminology

In general, each technology working group defines terminology for their section of the specification. However, the following definitions apply to the entire specification document.

2.1 Normative Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3 Bootup Time Specification

This is the specification of bootup time technologies and features, of the Bootup Time Working Group of the CE Linux Forum.

3.1 Introduction

The specifications of the Bootup Time Working Group deals with reducing the time required to boot a Linux kernel in a consumer electronics products. The purpose of this specification is to define requested or required features of Linux which improve the bootup time of the system for such products. Also, this specification mentions features which, over the long term, will assist developers in measuring and enhancing the bootup time for their systems.

While suspend, resume, and shutdown times are also within the scope of the Bootup Time Working Group, this version 1.0 of specifications does not include any technology in support of reductions in these areas. Reductions in those areas will be covered in future versions of the specification.

3.2 Rationale

Users expect to be able to use their CE products very soon after turning them on. Linux, as configured and used for desktop and server systems, exhibits long bootup times - on the order of 30 seconds to a few minutes. The technologies mentioned here (while small in number, in this first release), represent a few mechanisms which can be used to reduce bootup time.

3.3 Terminology

The following table presents terms used in this specification related to bootup time technology and features.

Definition of Terms for Bootup Times working group

Boot up:
The time from power on to user start (the start of user init).

Busy wait:
Using a timed loop to create a delay. This is often used with probing. Because the device is executing instructions in the loop, no other work can be done until the loop ends.

Cold start:
The time from power on to first available use.

Deferred:
An operation, which would occur at a specific point in the unmodified Linux boot up processing, is changed so that it can occur later in the processing.

De-serialized:
A set of operations, which would be serialized in an unmodified Linux environment, is changed so that the operations can be done concurrently.

Disk spinup:
The time required for a hard disk to reach operational speed after application of power.

Firmware:
The first set of machine instructions to run on the hardware after the application of power. The firmware may consist of several stages that bootstrap from one stage to the next. On an x86 machine, for example, the firmware is the BIOS. It is the responsibility of the firmware to do initial preparation of the hardware, and to load and execute the OS. This typically includes probing the hardware to discover attached devices and setting up configuration information that the OS will read during boot up.

First available use:
The first opportunity for actual use of the product, after a boot up, resume, or unhibernate operation.

First user experience:
The first visual or audio indication of activity to the user of the product, after a boot up, resume or unhibernate operation. (same as "splash")

Flash:
Flash memory

Hibernate:
The action of transitioning the machine into a no power state, in a way such that the current operating state can be restored in the future without a full initialization sequence.

Kernel decompression:
The action of decompressing a compressed kernel image. This stage usually includes simultaneously transferring the kernel image from persistent storage to RAM.

Kernel init:
: The kernel initialization sequence, consisting of the period of time from kernel start to start of user init.

Kernel start:
: The point in the boot up when the product executes the first instruction of start_kernel() rouine in the kernel.

Power lost:
: The moment that the product can no longer function because there is no more power applied to it.

Power off:
: The moment that the machine transitions to a "losing power" state (this is not the same as the time when no more power is being applied to the CPU).

Power on:
: The moment of application of power to the device.

Probing:
: The act of discovering the configuration or attributes of the machine or its operating. A probe typically consists of sending a hardware signal and waiting, for a given time, for a response from an attached bus or device.

Resume:
: The action of transitioning the machine into a usable state from a low power state (suspend).

ROM:
: Read-only-memory

Shutdown:
: The action of turning the product off. The time from user power off to power lost.

Suspend:
: The action of transitioning the product into a low power state, wherein the device cannot be used, but can be resumed quickly.

Serialized:
: A set of operations is serialized if the operations are performed in order, without overlap or concurrency.

Splash:
: The first visual or audio indication of activity to the user of the product, after a boot up, resume or unhibernate operation. (same as "first user experience")

Time-to-splash:
: The time from power on to splash (or first user experience).

Unhibernate:
: The action of transitioning the machine from hibernation into a usable state. Specifically, the usable state is the same or as close as possible to the state when hibernation was entered (eg., the same applications are running, the same files are open, etc.)

User init:

The time from the first instruction of the first user space program (usually /sbin/init) until the product is ready for first available use.

User power off:

The action or moment that the user initiates terminating the power to the device. Some devices do not use "user power off" as the primary means of "turning off" the device. Instead, a press of the "off" button may result in a suspend operation.

Warm start:

The time from start of resume to first available use.

XIP:

Execute-in-place - a method of executing code directly from ROM or Flash, without first loading it into RAM. This affects startup time, RAM and ROM footprint (size), and execution performance.

The terms used to describe boot up can refer to **events** (things that occur or start at a specific moment in time) or **time periods** (the time interval from one event to another).

Here is a list showing the sequence of main **events** during cold start: *Note: * means this event is optional.*

```
1.    power on
2.    firmware starts
2.1 * firmware splash
3.  * kernel decompression starts
4.    kernel start
4.1 * kernel splash
5.    user start
5.1 * user splash
6.  * RC script start
7.    application start
8.    first available use
```

Note that there may be multiple "splash" events in the system, but that the terminology here refers only to the first splash. This is also referred to in the terminology definitions as "first user experience". Splash events are highly important to diminish the *apparent* time to boot of the product. Note that the first splash event might occur at different locations in this sequence, depending on how the system is configured and what software performs the first user output. Also, even though user splash presentation is listed as event 5.1, it could occur anywhere between events 5 and 8, depending on the user-space software which performs the splash.

Here are bootup **time periods** (intervals), defined according to the above sequence of events:

- cold start: time from 1 to 8
- bootup: time from 1 to 5
- kernel init: time from 4 to 5
- user init: time from 5 to 8
- time-to-splash: time from 1 to either 2.1, 3.1, or 5.1 (depending on which is used)

3.4 Calibrate Delay Avoidance

3.4.1 Introduction

This page specifies a facility to avoid the delay associated with the Linux kernel function calibrate_delay(). The calibrate_delay() routine is intended to calculate at boot time an appropriate value for loops_per_jiffy, on an arbitrary platform. The value of loops_per_jiffy is used to execute "busywait" (non-yielding) delays inside the Linux kernel.

The duration of calibrate_delay() is independent of the CPU architecture or processor speed. Rather, the delay is dependent on the length of a "jiffy" and the number of iterations of the internal loops required to perform the calibration. A jiffy is the time period defined by the HZ variable in the Linux kernel. For the 2.4 Linux kernel, a jiffy is 10 milliseconds. Each iteration of the calibration algorithm takes about 1 jiffy, and there are, on average, about 20-25 iterations. Thus, the time to execute calibrate_delay() takes between 200 to 250 milliseconds on most platforms.

3.4.2 Rationale

The calibrate_delay() function in the Linux kernel can be one of the single most time-consuming events during system startup. For a 2.4 version Linux kernel, the calibrate_delay() operation accounts for about 200 to 250 milliseconds of delay during kernel startup.

The value of loops_per_jiffy is primarily dependent on processor speed, and its initial value at boot time is expected to be constant for each boot of Linux on the same hardware.

Because of this attribute, it is possible in most embedded situations to calculate the initial value in advance for a particular platform. This value can then be specified at compile time in the kernel, or provided to the kernel at boot time from an external source. This avoids the delay associated with dynamically calculating the value, by the kernel, on every system boot.

3.4.3 Specifications

1. A configuration option for the Linux kernel SHALL be provided which controls whether the value of loops_per_jiffy is statically compiled into the kernel or not. This option MUST be named CONFIG_USE_PRESET_LPJ.
2. With the kernel compiled with the CONFIG_USE_PRESET_LPJ option set, the system MUST avoid the delay associated with calculating loops_per_jiffy at runtime (with the exception of a "validation" error, mentioned below).
3. If the option CONFIG_USE_PRESET_LPJ is set, but the value of loops_per_jiffy is not specified at compile time, then the kernel compile SHOULD fail.
4. A kernel MAY provide support for accepting the value of loops_per_jiffy at boot time.

    4.1 If the kernel supports accepting the loops_per_jiffy value at boot time, then when loops_per_jiffy is so specified the kernel SHOULD avoid the delay associated with calculating loops_per_jiffy.

    4.2 If the kernel supports accepting the loops_per_jiffy value on the kernel command line, the syntax for this option MUST be: "lpj=<value>". The value for the option MUST be interpreted in loops_per_jiffy units.

    4.3 If the kernel supports accepting the loops_per_jiffy value at boot time, then the value provided at boot time should be used instead of any compiled-in (statically defined) value.

5. When the kernel does not dynamically calculate the value of loops_per_jiffy itself, it MAY validate the value of loops_per_jiffy at bootup time. If validation is performed, and the value of loops_per_jiffy is found to be

incorrect (to within a predetermined margin of error not specified here), the kernel SHOULD perform a normal delay calibration operation.

3.4.4 Notes (informational and non-normative)

1. Validating loops_per_jiffy means measuring that the value is accurate (within a certain tolerance) for creating a delay of 1 jiffy on the target platform.
2. It is expected that the cost to validate a preset value for loops_per_jiffy is about 1 jiffy.
3. Note that this specification avoids indicating how the value of loops_per_jiffy is specified at compile time. It could be done via another configuration option (intentionally not named here) or as a constant in the kernel source code.
4. An example of accepting the value of loops_per_jiffy at boot time would be a implementation where the firmware passed the value to the kernel on the kernel command line.
5. It is preferred that the feature to support acceptance of loops_per_jiffy at boot time not be a compile-time option. That is, if the feature is present it should always be available, independent of the setting of CONFIG_USE_PRESET_LPJ.
6. It is expected that power management schemes that change the CPU frequency dynamically will be enabled after the calibrate_delay() function completes. This means that the CPU frequency used for the initial calibration operation (or for which a preset value for loops_per_jiffy is provided at compilation or boot time) is the default bootup CPU frequency for the platform.

   Power management systems which subsequently change CPU frequency may need to adjust the value of loops_per_jiffy with changes in the CPU frequency to keep internal kernel timing delays accurate.
7. Accepting the lpj= argument at the kernel command line is one possible way to comply with requirement 4 of the specification. Other ways are to use persistent memory populated by firmware or having the firmware pass the value to the kernel using a tagged sequence.

3.4.5 References

A patch is available for the CELF source tree which provides the features described by this specification. It is available here.

3.4.6 Remaining Issues

- need to specify the manifest constant for ATAG_PRESET_LPJ?

3.5 IDE NoProbe

3.5.1 Introduction

This is the specification for a change in behavior of Linux support for the IDE "noprobe" option.

Historically, the IDE driver supports command line options to avoid probing certain devices and interfaces.

> *Note - The command line options for the IDE driver are documented in the file* `drivers/ide/ide.c`*, in a comment right before the routine ide_setup() (at about line 3200 in the file).*

The "ide<x>=noprobe" option was found not to work as expected in the in the 2.4.20 code.

The basic problem is that the data structures that describe possible IDE interfaces are initialized to indicate "noprobe" when the option is set at the kernel command line. However, later some drivers reset the value of "noprobe" in the data structures when certain interfaces are detected.

The end result is that (at least in Linux version 2.4.20) the kernel command line processing for "ide<x>=noprobe" has no practical effect.

The desired behavior is that the "noprobe" option will be observed, even if subsequent driver initialization finds out that an interface is actually present.

3.5.2 Rationale

Todd Poynor of MontaVista measured the effect of specifying "hdf=none ide3=noprobe" (avoid probing ide2 slave and both ide3 devices) on a 200MHz IBM 405GP "Walnut" evaluation board with a 33MHz PCI bus. A Seagate Barracuda ATA IV 60GB disk drive was cabled to one of the two IDE interfaces on a Promise Ultra66 PCI-IDE bridge card (PDC20262 chipset). All of the drivers for PCI, IDE, PCI-IDE disk, and EXT2 file system were built into the kernel.

The time to initialize IDE was 1.3 seconds when the missing devices were probed, and about 230 milliseconds when "hdf=none ide3=noprobe" was specified. Thus the resulting bootup time savings, with "noprobe" properly observed by the Linux kernel, were about 1.1 seconds, in this test.

This specification is needed because the kernel sometimes fails to observe the "noprobe" command line option.

3.5.3 Specifications

1. When the "ide<x>=noprobe" option is specified on the kernel command line, the IDE driver in the Linux kernel SHALL avoid probing the specified IDE interface.
2. When the "hd<x>=noprobe" option is specified on the kernel command line, the IDE driver in the Linux kernel SHALL avoid probing the specified IDE device.

3.5.4 Notes (informational and non-normative)

1. Processing for the option "hd<x>=none" (which is related to the "hd<x>=noprobe" option) is specific to a single processor architecture (i386). No aspect of the behavior of this option is specified here.

3.5.5 References

See the page IDENoProbe for more information about this problem, and for a sample implementation.

3.5.6 Remaining Issues

- see if "noprobe" behavior has changed in 2.6

3.6 Kernel Execute-In-Place (XIP)

3.6.1 Introduction

This specification is for Kernel Execute-In-Place (XIP). XIP refers to the capability for a Linux kernel to be executed directly from a persistent, read-only memory or media type (usually ROM or FLASH). When the kernel is executed in this fashion, the bootloader can avoid loading the kernel from persistent storage, decompressing it, and saving it into system RAM.

3.6.2 Rationale

The process of loading, decompressing, and writing a kernel to RAM is dependent on the size of the kernel and the speed of the processor and the associated memory devices. An observed time in practice is about 500 milliseconds. (For examples of observed time, please see the page referred to in the References section below.) This time can be saved by avoiding these steps and instead executing the kernel directly from its location in persistent storage.

3.6.3 Specifications

1. The Linux kernel SHALL support an option for executing the kernel directly from a persistent memory-mapped storage location.

    1.1. The configuration option for kernel XIP MUST be called CONFIG_XIP_KERNEL

2. When the option CONFIG_XIP_KERNEL is enabled, compilation of the kernel SHALL produce a kernel image that can be directly executed from ROM or FLASH.

3.6.4 Notes (informational and non-normative)

1. It is possible that FLASH memory being used for kernel XIP may not be available for other uses by the kernel which is being executed from it. Whether this is the case or not depends on the implementation details for the code which supports this specification. This specification does not currently require that the kernel be able to use the flash from which it executes for other uses (like filesystems.)

3.6.5 References

See the page KernelXIP for information about this feature, and for examples of time reductions that have been demonstrated with its use.

3.6.6 Remaining Issues

- decide if we should address performance side effect of kernel XIP
- Figure out difference in usage or intent (if any) between use of CONFIG_XIP_ROM and CONFIG_XIP_KERNEL. Determine if the difference in feature set for execution from ROM vs. execution from FLASH (when flash is also used for filesystem operations) means that we should specify these separately.
- Decide if we should standardize use of CONFIG_XIP_PHYS_ADDR vs. CONFIG_XIP_KERNEL_TXTBASE

3.7 Work In Progress

The following item is currently being worked on, but is not ready for publication yet.

3.7.1 Timing API

3.7.1.1 Introduction

This specification is for a timing API for the Linux kernel which is available early during the bootup process.

3.7.1.2 Rationale

The purpose of this specification is to provide a platform-neutral API for timing instrumentation of the Linux kernel. Currently, there are a few different timing APIs in the kernel used for such instrumentation. For example, on the i386 architecture the Time Stamp Counter (TSC) register of the processor is used for getting fast timestamps for low overhead and high resolution time measurements. (This has historically been made available with the get_cycles()

function call.) Many timing measurement systems for the Linux kernel have been based on either get_cycles() or directly on a call to read the TSC register, such as rdtscll().

There are a few problems with this setup. First, these functions are not supported on all architectures and platforms of interest. Thus, measurement systems that utilize them are not portable to these other platforms. Also, other functions which ARE available on all platforms are either high overhead, or are not usable from the very beginning of kernel bootup, or both.

This specification, then, is intended to address the need for a common API available on all embedded platforms, for a low-overhead, high resolution clock source accessible very early (even before setup_arch()) in the boot sequence of the kernel.

3.7.1.3 Specifications

1. The Linux kernel SHALL support a configuration to provide the following described timing API.

    1.1 The configuration option to enable this feature MUST be called CONFIG_FAST_TIMESTAMPS

2. When CONFIG_FAST_TIMESTAMPS is enabled, the kernel SHALL provide the following 2 routines:

    2.1 void store_timestamp(timestamp_t *t)

    2.2 void timestamp_to_timeval(timestamp_t *t, struct timeval *tv)

3. The store_timestamp() API MUST be available and useful no later than upon the completion of the setup_arch() routine in the kernel.

    3.1 The store_timestamp() API SHOULD be available and useful as soon as possible after start_kernel()

4. The size and resolution of timestamp_t *t SHOULD be such that timestamps can record at least 30 seconds of timing information.

    4.1 The size of timestamp_t SHOULD be no greater than 64 bits.

5. store_timestamp() MUST produce values for timestamp that are monotonically increasing with each call *on the same processor* (with the exception of overruns or explicit clock value changes).
6. The resolution of timestamp, after conversion to timeval, MUST NOT be less than 1 jiffy.

    6.1 The resolution of timestamp, after conversion to timeval, SHOULD be at least 100 usec.

    6.2 A resolution of at least 1 usec for values of timestamp is preferred.

3.7.1.4 Notes (informational and non-normative)

- The clock needs to be accessible at least during the boot up period of the kernel.
- A free-running clock could be set up by firmware (before kernel start) and polled afterwards by the kernel in order to measure total system bootup time. Note that this implies that the timestamp value does not need to start at 0 with the beginning of execution of the kernel or the setup of the timer device.
- It is preferred that the timestamp value returned should not fluctuate with changes in CPU frequency, but this is

not mandatory.
    - Alternative, it should be possible to avoid changes in the CPU frequency during the timing period.
- The values returned need not be linearly related. That is, it is acceptable for the values to be non-linear, as long as the conversion to timeval (sec, nsec) is correct. Thus, as one example of value management, it is possible to store the hardware clock value in the low 32 bits, and the number of rollovers in the high 32 bits. This works even if the clock source itself is less than 32 bits wide (eg 12 bits, or 16 bits).
- It is not necessary for there to be an accurate correlation of timevals produced by timestamp_to_timeval() to wall clock times. That is, there is no requirement to implement a strict gettimeofday call.
- the purpose of separating the "store" function from the "to_timeval" function, is to make the overhead of actually acquiring the timestamp as little as possible. Some callers may immediately convert the timestamp to a timeval, but other time-sensitive code may defer the conversion to timeval units until some later time (or indefinitely).
- The size of timestamp_t should be as small as possible. It is expected that timestamp values will be stored in trace logs which may accumulate data points quickly. It is expected that on many embedded architectures, timestamp_t will be defined as an unsigned long (32 bits), and that it will NOT be defined to be any greater in size than an unsigned long long (64 bits).
    - Embedded processors have CPU clock frequencies ranging from a few MHZ to a few GHZ. A 1 GHZ clock will overrun 32 bits in about 4 seconds. This means that for fast-running hardware timestamp_t should probably be an unsigned long long.

3.7.1.5 References

There is general information about the requirements, other timing mechanisms, and proposals at InstrumentationAPI

3.7.1.6 Remaining Issues

- ⚠ *It would be good to try this out somewhere!!*

4 Power Management Specification

4.1 Introduction

The goal of power management technology is to allow Consumer Electronic (CE) device manufacturers to optimize the power use of their products, including a variety of mobile devices, home appliances, and other end-user devices.

4.2 Rationale

In CE products, battery-driven mobile devices have particularly stringent requirements for power management, and this affects the product's competitiveness significantly in specific product categories. Mobile phones and other handheld devices are driven by secondary batteries, not wired power supply. Efficient power usage is required to increase the utilization time of such products.

4.2.1 Power Dissipation Elements in CMOS Circuits

CMOS(Complementary Metal Oxide Semiconductor) circuit has three elements of power dissipation; leakage current, dynamic current, short-circuit current power dissipation. The leakage current power dissipation is caused by the leakage current. The leakage current power dissipation is quadratically dependent on operating voltage but not dependent on clock frequency. The dynamic current power dissipation is caused by the dynamic current which is required for signal transitions. The dynamic power dissipation is quadratically dependent on the voltage and linearly dependent on the frequency of signal transitions. The short-circuit current dissipation is generated by short circuit current when NMOS and PMOS transistors are activated simultaneously. Short-circuit current power dissipation is either linearly or quadratically dependent on the supply voltage, depending on the size of the length of channel between PMOS and NMOS transistors. Of those three elements, the dynamic current power dissipation is a dominant factor because the frequency of signal transitions is very high.

4.2.2 Power Reduction Methods

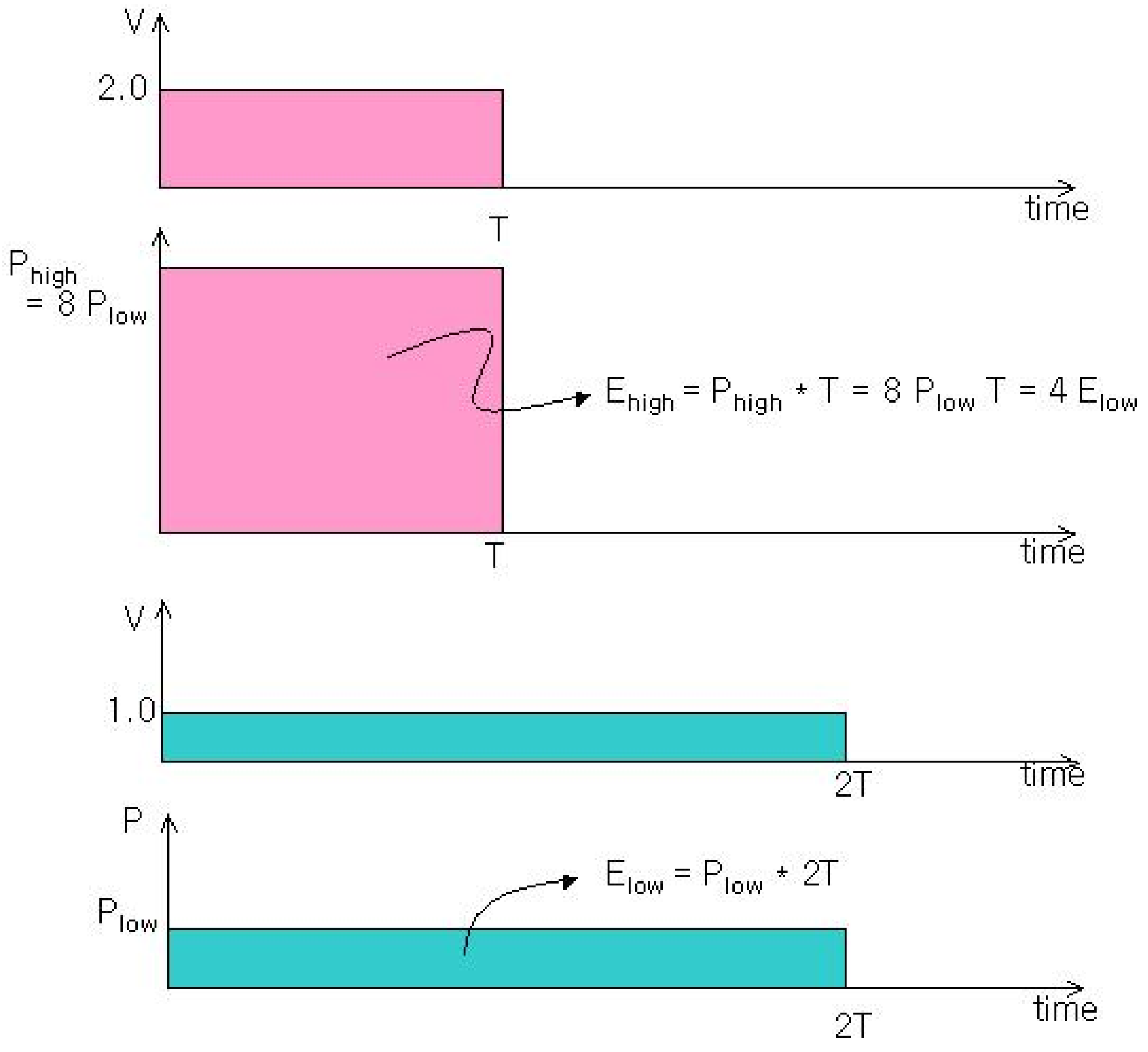
There are various methods for reducing power consumption: voltage/clock scaling, clock gating, power supply gating, input signal selection and so on.

- The voltage/clock scaling is the method that reduces both the dynamic power dissipation and the static power dissipation by decreasing voltage and clock dependently or independently.
- The clock gating reduces the dynamic power dissipation by disconnecting the clock from an unused circuit block.
- The power supply gating disconnects the supply power from unused circuit blocks, which are components of whole circuit. In clock/power supply gating methods, it is important to decide whether the block is used or not and to restore the blocks when they are used again.
- The input signal selection affects the reduction of the leakage power dissipation. The input signals of CPU pins should be adjusted to minimize the usage of power when the system is suspended.

Voltage scaling is the key to achieve power reduction and energy reduction. As clear from the analysis of the three power dissipation elements, operating voltage plays a major role in power dissipation. Let's take an example to see how voltage/clock scaling can reduce power and energy. If operating voltage is down-scaled from 2.0V to 1.0V, operating frequency is linearly dependent on the operating voltage in practice. So let us assume clock frequency at 1.0V is half of the frequency at 2.0V. In this case, nearly 87.5% of power can be reduced theoretically. In terms of energy, the total energy for finishing a job need to be compared. Please note that energy is the integration of power dissipation over a period of time. The time taken to finish a job at 1.0V is twice as that of 2.0V. So the total energy required to finish a job will be reduced by 75%.

$$P = k \times C_{out} \times V_{dd}^2 \times f$$

k	: activity factor
C_{out}	: total chip capacitance
V_{dd}	: supply voltage
f	: clock frequency



Clock gating and power supply gating can be used to reduce power usage by unused blocks. It is important to predict correctly the idle period of a hardware block because incorrect prediction may lead to performance degradation due to the latency of reconnecting clock or power to the hardware blocks.

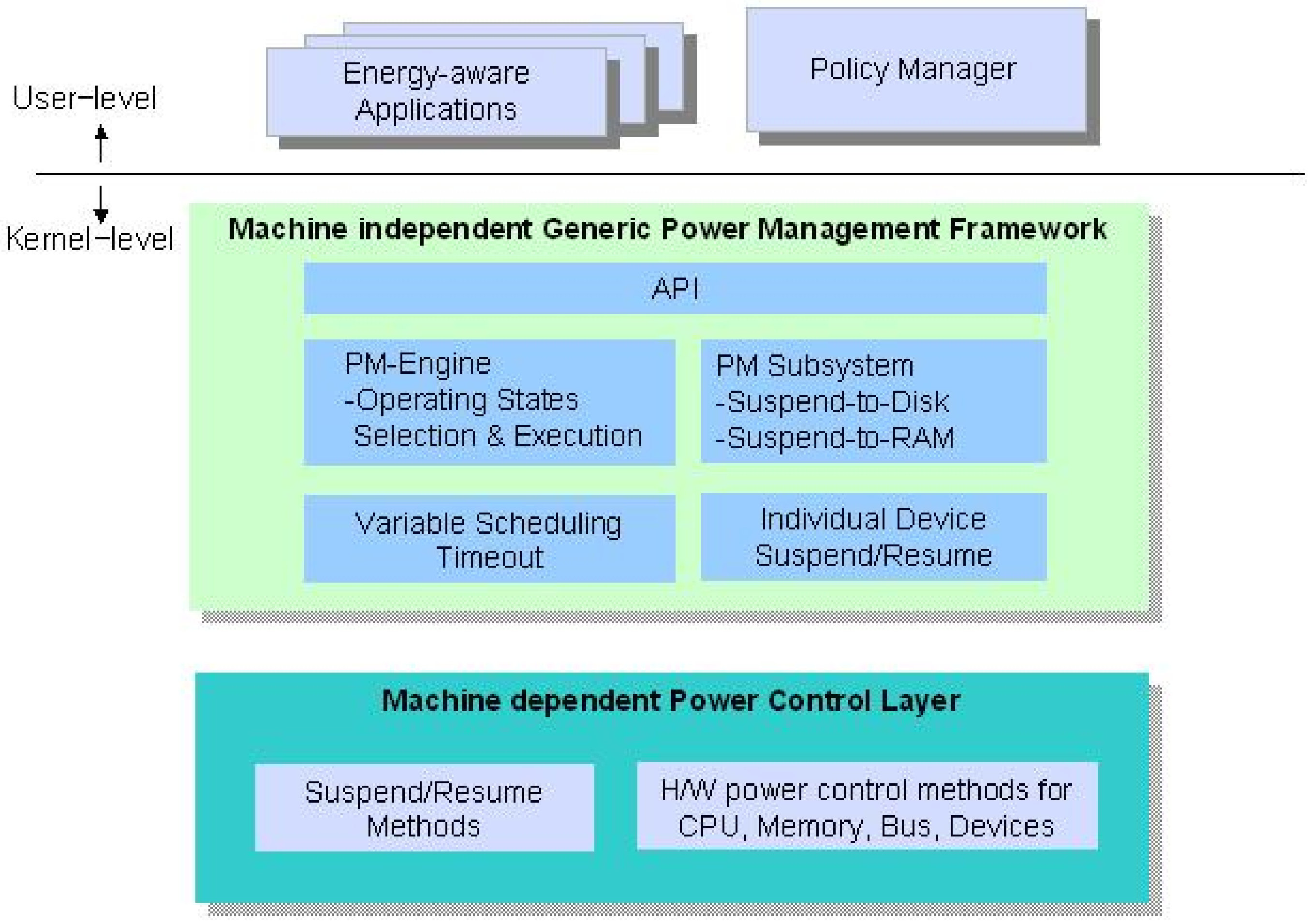
4.2.3 Power Management Architecture

In designing power management architecture for Linux, the followings are kept in mind.

- To design a system-level optimization of power usage involving interactions among application, OS, and hardware.
- To provide a generic power management framework for various hardware platforms

System-level power management is important for coordinating the performance and power dissipation of hardware components in a system. Recently, CPU chip vendors deploy dynamic clock/voltage scaling support in their processors to enable OS-level power management capability. The power reduction techniques offered by hardware components should be synthesized, managed, and adjusted to minimize the power dissipation of the whole system while the requirements of system should be satisfied as well. Existing applications need to be adapted to use the power management capability of operating system and such applications are said to be "Energy-Aware".

The power management architecture is designed to provide a generic framework for various hardware architectures. Hardware specific power control interfaces are abstracted by a power control layer. Power management can be implemented for various hardware platforms by adapting the machine-specific power control layer as necessary.


User-level
Kernel-level
Energy-aware Applications
Policy Manager
Machine independent Generic Power Management Framework
API
PM-Engine -Operating States Selection & Execution
PM Subsystem -Suspend-to-Disk -Suspend-to-RAM
Variable Scheduling Timeout
Individual Device Suspend/Resume
Machine dependent Power Control Layer
Suspend/Resume Methods
H/W power control methods for CPU, Memory, Bus, Devices

In this specification, power management technologies are classified into three categories; platform suspend/resume, device power management, and platform dynamic power management. Platform suspend/resume is aimed at major power state changes at infrequent intervals, while platform dynamic power management is aimed at subtle changes across a wider range of power states at frequent intervals. Platform suspend/resume technology is used for reducing most of power dissipation of a product while the product is idle for a long time. In contrast, dynamic power management is used for reducing power dissipation while a product is being used. Dynamic power management covers also reducing power for very short idle periods of time while a product is busy. Device power management is to suspend/resume devices in a platform, which is used by platform suspend/resume and dynamic power management technologies.

The rest of this specification is organized as follows. In Terminology section, general power management related terminologies are defined. Platform suspend/resume, device power management, and platform dynamic power management are described in separate sections. Work in Progress section describes important on-going projects which are not included in CELF 1.0 spec but need to be considered for next CELF specification.

4.3 Terminology

Definition of Terms for Power Management working group

Classes:
In the future, a Class will be a set of Operating Points. DPM will be architected to choose the appropriate Operating Point for the current Class, based on constraints. The current implementation of DPM restricts each Class to contain only one Operating Point. Continuing the above example of device constraints for a network adapter, if a Class contained two Operating Points, "slow_cpu_fast_io" which specified an IO bus frequency of 25 MHz and "slow_cpu_slow_io" which specified an IO bus frequency of 5 MHz, then DPM would not be able to set the Operating Point to "slow_cpu_slow_io" while the network adapter was active.

Device constraints:
Constraints describe the required relationships between the hardware operating parameters. For example, for a network adapter to function properly, it might be required that the frequency of the IO bus to which it is attached must be greater than 10 MHz.

DPM:
The MontaVista Dynamic Power Management subsystem

Operating Points:
An Operating Point is a specific set of values of power parameters. The system designer chooses an Operating Point to reflect a certain level of system performance together with an associated level of energy consumption.

Hibernate:
See Suspend-to-Disk

Operating States:
The Operating State reflects the current state of the system: executing a task, processing interrupts, or idle. Each process has an associated Operating State, as does the system idle loop and the interrupt handling code path. It is also an abstraction that software uses to request a specific level of system performance, since there are multiple operating states that can be associated with a task, and the various operating states may be mapped to differing Operating Points, thus causing differing levels of performance and power consumption for different tasks. The current system Operating State is set to the Operating State of the process currently executing (or the idle loop or interrupt handlers). The board specific implementation portion of DPM may also alter the system's current Operating State while kernel code is executing. For example, the kernel idle loop code may explicitly set the system's current Operating State to "idle" while there is no task that is ready to execute.

Policy Manager:
Software that dynamically sets the current policy from the set of defined policies. The Policy Manager selects the current Policy based on certain system conditions. For example, the Policy Manager can select the Current Policy in one of the following ways: 1. If the user presses the device's soft-power switch, the Policy Manager can choose an ultra_low_power Class; 2. If remaining battery power reaches a threshold, the Policy Manager may choose a lower_power_and_performance Class; 3. If the device is connected to main power, the Policy Manager may choose a high_performance Class.

Power Control Layer:

Power control layer is the software layer on top of which power management framework sits. This layer provides H/W specific power control methods to power management framework with abstracted power control interfaces.

Power Management Framework:

Power management framework in kernel maintains system power management policy and implements core functions for static power management and dynamic power management technologies. PM framework plays the role of coordinating energy resource among multiple tasks and adapting system operating state according to task specific requirements.

Power Parameters:

Power parameters are the hardware parameters that may be modified to affect the power usage of the platform, such as clock frequencies or dividers, voltage levels, and so forth.

Power Policy:

A Policy is a mapping of which Class is invoked for each Operating State. For example, a "power-off" policy may map all Operating States to an extremely low power Class; a "power-off-fast-wakeup" may map all Operating States to a low power and low wakeup latency Class; and a "active" policy may map an Operating State used by interactive processes to a high power Class, an Operating State used by background processes to a medium power Class, and an Operating State used by the kernel idle loop to a low power and low wake up latency Class. Whenever the Operating State changes (due to an explicit request or implicitly due to a change of which process is currently executing) DPM uses the current Policy to determine which Class is associated with the new Operating State. DPM selects the best Operating Point from the new Class, and then sets the hardware parameters defined by the Operating Point.

Suspend/Resume:

System suspend refers to placing the system in a low-power state with many components powered off, or perhaps entirely powering off the system after saving state to stable storage. This is a relatively lengthy procedure undertaken when an extended period of product inactivity is expected, as for example when the user presses a "standby" button on the product or the product has not been in use for a long period of time. This procedure can thus be distinguished from the power state changes appropriate for a running system driven by technologies such as Dynamic Power Management, which are executed more quickly during very brief idle periods. System resume refers to the process of restoring the state of the system to approximate pre-suspend conditions upon resume.

Suspend-to-Disk/Hibernate:

A technique by which systems preserve state on disk during suspend and restore system state from disk upon resume. This is typically done on platforms where all of RAM may be saved to a disk device and then the RAM (and perhaps the entire system) may be powered off during the suspend period. This is commonly referred to as "hibernating" for laptop/notebook computers

Suspend-to-RAM:

A technique by which systems preserve state in RAM during suspend and restore system state from RAM upon resume. This is necessary on platforms that remove power from some portions of the system, such as the CPU core or I/O core, but maintain power to SDRAM during the suspend period

Suspend modes / Standby modes / Sleep modes:

A platform may support more than one suspend mode, that is, system behavior during a suspend may differ depending on what suspend mode has been requested. For example, one mode may stop certain clocks during suspend but leave all components powered on, while another mode may power off various components (such as the CPU core or I/O modules) but leave others powered on (such as to leave SDRAM in self-refresh state), and so forth.

4.4 Platform Suspend/Resume

4.4.1 Background

Embedded platforms typically offer one or more low-power system states in which power consumption is lowered through such means as stopping clocks and/or powering off some or all system components. CE products may take advantage of these states to save power during periods of inactivity by entering these states. This is particularly important crucial to CE products powered by batteries.

Generic Linux contains some support for these topics. However, the community emphasis is currently on certain features, primarily hibernate-to-disk, for laptop/notebook computers based on desktop-compatible processors (and powered by batteries much larger than are used in many CE products, such that aggressive power savings may not be as critical). Basic platform suspend/resume support as defined here is not a priority at present, and neither is support for embedded platforms. One of the largest efforts is for hardware based on the ACPI standard, which is not implemented on most embedded platforms.

This specification addresses the potential lack of static power management features on a platform used for CE products by requiring that a CELF-conforming Linux for that platform support its basic system suspend/resume capabilities. This specification also requires a minimal set of functionality closely associated with static power management. The basic suspend/resume capabilities outlined here may be extended by future CELF specifications that cover additional features useful for CE products.

4.4.1.1 Scope

This specification generally covers the basic functionality needed in an embedded linux system to save power by suspending and resuming operation of the system, utilizing suspend states implemented by the platform. The ability to suspend and resume systems is often referred to as "static power management" in the CELF Power Management Working Group.

System suspend is usually a relatively lengthy procedure, undertaken when an extended period of product inactivity is expected. This procedure can thus be distinguished from the power state changes controlled by more dynamic mechanisms, such as the IBM-MontaVista Dynamic Power Management subsystem (DPM), which may be executed more frequently during very brief idle periods, and are covered in separate CELF power management specifications. The system power state modifications made by dynamic power management schemes during brief idle periods typically result in less power savings than do the system suspend states covered in this specification. Furthermore, the dynamic power management state modifications normally have lower entry and exit latencies than do system suspend states, and require few or no interactions with device drivers. A system suspend, on the other hand, may take a noticeable amount of time to suspend and resume, and in many cases requires interactions with device drivers to save state and/or to enter and exit device suspend states.

A related topic not explicitly covered in this specification is "hibernate-to-disk", where system state is saved to a stable storage device (such as a hard disk) during a system suspend and restored from storage at system resume. The state saved usually includes most or all of RAM. This type of system suspend/resume is commonly implemented for laptop/notebook computers. Because the more specialized nature of hibernate-to-disk (which is typically a software feature not tied to the specifics of the base platform or devices), as well as the fact that this functionality is only appropriate for certain hardware and product configurations with large and fast storage devices available for saving state, hibernate-to-disk is not covered in this specification. This topic may be covered in future CELF power management specifications.

Platforms vary considerably in the kinds of support provided for static power management, and any attempt to standardize these technologies for a wide variety of platforms will necessarily be subject to a number of grey areas. The intent of this specification is to ensure a developer that, by choosing a CELF-conforming kernel source base for a particular platform, a number of commonly useful mechanisms and interfaces are available to take advantage of basic static power management features of that platform. In many cases this basic support will need to be augmented to obtain the exact behaviors desired for a particular product.

4.4.1.2 Terminology

The following terms are used in this portion of the specification:

Active state:
A system or device that is not in a suspend state is said to be in an "active state".

SDRAM self-refresh mode:
A platform capability that typically saves additional power during times in which no SDRAM access may occur.

System suspend:
The process of placing the system into a system suspend state. This may be triggered, for example, in response to the consumer pressing a "standby" button on the product. This process generally also performs a device suspend sequence for most or all devices.

System suspend state:
A reduced-power state supported by the platform. Modern embedded platforms often offer a variety of modifiable parameters and execution modes related to power consumption; this specification primarily targets those system suspend states used to dramatically reduce power consumption for extended periods of time (further clarification appears in the discussion below).

System resume:
The process of restoring the state of the system to approximate pre-suspend conditions when resuming operation after a previous system suspend. System resume may be triggered, for example, when the consumer presses a "wakeup" button or when the product is automatically awoken by an alarm or external event (such as incoming call on a phone). This process generally also performs a device resume sequence for most or all devices.

Wakeup method:
A platform mechanism for exiting a system suspend state. Wakeup may be triggered by such means as pressing a button, I/O activity (such as sending characters to a serial console device), or asserting an interrupt.

4.4.1.3 System Suspend/Resume Discussion

System suspend may be triggered for such reasons as:

- the consumer presses a "standby" button on the product
- the product automatically suspends after the product has not been in use for a certain period of time

- an application or kernel driver suspends the system due to such conditions as a low battery strength indication or temperature threshold exceeded

The documentation for the particular platform may refer to its system suspend states using names such as "standby", "sleep", or "suspend" states or modes. The precise characteristics of system suspend will vary by platform and by the particular system suspend state. A system suspend state might entail some or all of the following actions:

- stopping or slowing various clocks
- removing power from some or all devices, or placing devices into a low-power state
- placing SDRAM into self-refresh mode
- lowering core operating voltage and other platform power parameters
- halting CPU execution

This specification places no requirements upon the exact behavior of system or device suspend states, or upon the wakeup methods that resume normal system operation. The intent is to ensure that a CELF-conforming Linux supports the basic system and device suspend/resume features offered by the platform to a minimal degree.

This specification does require that a system suspend first call drivers for devices in an active state, in order to prepare for the system suspend. Depending on the platform and system suspend state characteristics, such preparation may be useful to further lower system power consumption by placing all appropriate devices into a suspend state, and/or may be necessary to save device state such that device operation can be later resumed during system resume. If the platform will remove power from the device during the system suspend state then first placing the device into a low-power state is, of course, not necessary, but it is commonly the case that device state will need to be saved in order to reconfigure the device when later re-powered.

If multiple system suspend states are implemented by the platform and supported by Linux, then a CE application may choose among the suspend states based on product-specific criteria such as whether a particular suspend state is compatible with proper functioning of required I/O devices during the suspend, the latency involved when entering and exiting the state, and so forth.

4.4.2 Specifications

1. If the platform supports at least one system-suspend state and at least one method to wake up from the system-suspend state, a configuration option for the Linux kernel SHALL be provided that controls whether the kernel has the ability to enter and wake up from at least one system suspend state.
2. If the platform supports more than one system suspend state, the ability to enter and wake up from all platform-supported system-suspend states SHOULD be implemented.
3. Both a kernel programmatic interface and a userspace interface to initiate system suspend SHOULD be provided.
4. If the platform supports SDRAM self-refresh mode and supports at least one system-suspend state that leaves SDRAM powered, the Linux kernel system-suspend code SHOULD have the ability to place SDRAM in self-refresh state.
5. The kernel SHOULD provide interfaces for delivering power management events, including power button, smart battery, and temperature sensor events, to user space.
6. A configuration option for the kernel SHOULD be provided that controls whether the kernel supports a hibernation technique by which the system preserves state on disk or flash memory during suspend state and restores the state from disk or flash memory upon resume.

Specific interfaces to perform the above are not required by this specification; see Non-Normative Notes for discussion.

4.4.3 Non-normative notes

Among the choices for system suspend/resume interfaces are:

- The Linux 2.5/2.6 PM subsystem implements a kernel API for system suspend and resume, with limited support for choosing a specific system suspend state.
- The Linux 2.5/2.6 sysfs filesystem exports interfaces that applications may use to suspend the system; these interfaces call the PM subsystem kernel API.
- The /proc/sys/pm interface for system suspend/resume. Note that these interfaces are generally being replaced with the PM subsystem in future Linux versions.
- The Dynamic Power Management subsystem (DPM) offers an interface for system suspend that may be convenient to use in systems that employ DPM for other power management tasks.
- Hardware Platform Dependencies
    - The ability to support a special register used by boot-loader to detect whether the type of booting is cold boot or system resume.
    - Usually, platform is designed to support power supply gating which is to disconnect power from a part of PCB. The power supply gating can eliminate both leakage current and dynamic current while clock gating can eliminate dynamic current only. PCB has multiple power plane whose power can be gated independently.
    - Boot-loader need to detect whether booting is cold boot or system resume.
    - There are four types of power management events typically; button event, battery event, thermal event, and timer events. Those events are useful for predictable power control and prevent a system from being overheated.

A version of the Linux 2.5/2.6 technology described above has been backported to Linux 2.4 for use in CELF-conforming systems based on the 2.4 Linux kernel.

4.5 Device Power Management

4.5.1 Background

Various devices on embedded platforms upport low-power states that can be employed by CE products at times when full operation of the device is not required. The ability to manage device power usage may be crucial to many CE products, especially those powered by batteries.

Generic Linux contains some support for these topics. However, device suspend/resume support is not a priority at present, and neither is support for embedded platforms. This specification addresses the potential lack of device power management features on a platform used for CE products by requiring that a CELF-conforming Linux for that platform support its basic device power management capabilities. This specification also requires a minimal set of functionality closely associated with device power management. The basic capabilities outlined here may be extended by future CELF specifications that cover additional features useful for CE products.

4.5.1.1 Terminology

CE platform device:

A device that is closely associated with the platform, that is supported by a Linux driver under an open-source license, and that may reasonably be expected to appear in an actual consumer electronics product based on the platform. See further discussion below.

Device resume:
The process of restoring the state of a device that was previously suspended to normal operation.

Device suspend:
The process of placing a device into a device suspend state, and/or of preparing the device for a system suspend. This may occur for such reasons as explicit instructions to power down an unneeded device by an application, or as part of a system suspend. Depending on the platform and device, device suspend may include saving state to allow later restoration of state at device resume time.

Device suspend state:
A reduced-power state supported by a device. Many, but not all, devices support at least one device suspend state, which may prevent operation of the device until a device resume is performed. The various device suspend states may be activated by device-specific interfaces and/or automatic criteria, or may follow standards such as ACPI. This specification primarily targets reduced-power states that are activated by the device driver when needed (rather than automatic hardware mechanisms, such as inactivity timers).

4.5.1.2 Device Suspend/Resume Discussion

Device suspend may occur for reasons that include:

- an application explicitly manipulates device state, such as to power down a device no longer required by the application
- a system suspend occurs, which suspends all devices prior to suspending the system
- a power policy management subsystem, such as DPM, places the system in a state that is incompatible with operation of the device
- a hardware or software mechanism triggers a low-power state after a period of inactivity
- the driver powers down the device because applications no longer hold an open reference to the device

In many cases, CELF specifies support for evaluation or reference boards, based on which CE products may be derived using a custom hardware design that incorporates the processor and various devices. This specification targets only devices and drivers termed "CE platform devices" here, which meet these criteria:

- The device is closely associated with the platform. A CE platform device may be physically located on a single-board computer or in some other way be tightly coupled to the platform supported by CELF-conforming kernel source, such that its presence is likely in many products that may be based on the platform. Devices not included in this definition include arbitrary cards that plug into buses provided on the platform, such as PCI or PCMCIA, or that may be attached in custom hardware designs.
- The device is supported by a Linux driver under an open-source license.
- The device may reasonably be expected to appear in an actual consumer electronics product based on the platform. A CE platform device may be distinguished from devices that are present on evaluation or reference boards for development or debugging purposes, such as an ethernet interface that is unlikely to appear in an actual CE product.

This specification targets CE platform devices exclusively, in order to give product designers the necessary tools to save power in actual product configurations. This distinction is made in order to avoid mandating power management capabilities for:

- devices not present in the CELF-supported evaluation/reference boards
- devices for which no open source driver has been made available
- devices that serve only a development or diagnostic function

Device state may need to be be saved during the device suspend operation, such that device operation can later be restored to approximately the same condition at device resume. If so, device state is typically saved in SDRAM since SDRAM is usually powered (perhaps in self-refresh state) during the suspend interval -- a platform that does not preserve SDRAM during suspend generally must reboot at resume time, whereupon device state can be restored from stable storage if needed.

System suspend may remove power from some or all devices, depending on the platform and the particular system suspend state entered, leaving the devices unpowered during the suspend interval and restoring power at system resume. This may affect the manner in which device resume occurs during system resume, since recovering from a power cycle may require different procedures than are needed for individual device suspend/resume. This may also affect the actions to be taken to accomplish a device suspend during system suspend; for example, entering a low-power suspend state may not be useful if the platform is about to remove power from the device.

4.5.2 Specification

1. If a CE platform device that supports device suspend/resume actions, then both kernel programmatic interfaces and userspace interfaces MUST be provided to individually perform device suspend and device resume for only that device. These interfaces SHOULD suspend other devices that depend upon the selected device for correct operation. Other devices that do not depend upon the selected device for correct operation MUST NOT be suspended or resumed by these interfaces.
2. If a CE platform device supports a device suspend state, or if any actions are needed in order to correctly resume device operation after a system resume from at least one system suspend state supported by the Linux kernel, then the driver for the device SHOULD implement the support necessary for the device suspend and resume interfaces.
3. If a CE platform device supports one device suspend state, then the device suspend processing performed by the driver for the device MUST be capable of causing the device to enter the supported device suspend state. Where multiple device suspend states are available, the driver SHOULD be capable of entering each of these states.
4. The device resume processing performed by the driver for a CE platform device SHOULD restore device operation to approximate pre- suspend conditions.
5. A mechanism for requesting device suspend for all active CE platform devices at system suspend time MUST be provided. A mechanism for system resume to restore to an active state all CE platform devices that were in an active state prior to system suspend MUST be provided.
6. It is RECOMMENDED that drivers place devices into low-power states when not in use or after a period of inactivity.
7. It is RECOMMENDED that drivers and platform support code make use of hardware features to automatically place devices into lower-power states, such as to stop clocks (sometimes referred to as "automatic clock gating"), after a period of inactivity or when the hardware is in some manner able to detect that the device is not in use.

4.5.3 Non-normative notes

Among the choices for device suspend/resume interfaces are:

- The Linux 2.5/2.6 Linux Driver Model (LDM) implements a kernel API for driver suspend and resume functions. LDM also provides a kernel API for calling drivers to suspend all devices at system suspend, and to

restore normal operation of the devices at system resume.
- The Linux 2.5/2.6 sysfs filesystem exports interfaces that applications may use to suspend and resume devices (individually); these interfaces call the LDM driver suspend and resume kernel APIs.
- APM function calls (such as pm_register) for device suspend/resume. Note that these interfaces are generally being replaced with Linux Driver Model interfaces in future Linux versions.

A version of the Linux 2.5/2.6 technology described above has been backported to Linux 2.4 for use in CELF-conforming systems based on the 2.4 Linux kernel.

Comments on the 4.5.2 specification section are the following:

- 4.5.2.4 and 4.5.2.6 does not make specific requirements upon devices that may change state in significant ways between device suspend and device resume, for example, a controller for a hotpluggable bus such as PCMCIA or MMC, where the consumer may insert or eject cards during the suspend interval.
- 4.5.2.5: In 2.6 kernel,Linux Driver Model (LDM) has this mechanism. CE platform devices that are already suspended via a previous individual device suspend should be skipped, if it is not necessary to perform any further device suspend processing prior to a system suspend (such as to save state in RAM in preparation for power-off). Devices that were already individually suspended prior to system suspend should be left in a suspended state after system resume.
- 4.5.2.6: For example, when the last open file descriptor for a device is closed, the driver may suspend the device.

4.6 Platform Dynamic Power Management

4.6.1 Rationale

Many CE products are powered by batteries, not by a wired power supply. Making efficient use of the energy stored in the batteries is very important, both in terms of achieving an acceptable length of product usage per battery charge and in terms of product form factor; an energy-efficient product can be powered by a smaller sized battery than other would be required. This also has a bearing on the manufacturing cost of the product. Power savings can be achieved through the use of low-power system states, entered when the product is inactive or when selected explicitly by the user - for example, by use of the power switch to put the product into a standby state. This type of power management is referred to as static power management.

Dynamic power management refers to managing power while the product is in use, running user applications. Such power management can involve dynamic control of peripheral clocks and power supplies, varying the timer tick frequency during idle periods and CPU frequency/voltage scaling. This uses a combination of user-space and kernel-space software:

- a Policy Manager component running in user-space
- a Power Management Engine ("PM-Engine") component running in kernel-space

The software selects the system operating state based on a combination of the following:

- application requirements
- the parameters for battery lifetime or task deadlines
- CPU loading
- the current device constraints

- the current operating state
- user interaction

4.6.2 Specifications

If a platform has an interface to change its power parameters while platform is running, a CELF conforming kernel for the platform MUST satisfy the following:

1. A kernel SHOULD provide a userspace interface to add/remove/modify a power policy.
2. A kernel SHOULD provide interfaces to modify the platform's operating state and power parameters of the platform accordingly.
3. A kernel SHOULD have the capability to set the platform at the operating state of a task while the CPU is executing that task.
4. A kernel SHOULD have a userspace interface to initialize/remove/modify the operating state of a task.
5. A device driver framework MAY provide interface to register device constraints on device open and unregister the constraints on device close.
6. When a CPU is idle, a kernel SHOULD set the CPU at its low power mode if the CPU supports at least one low power mode.
7. System clocks MUST be correct regardless of power parameter changes.

4.6.3 Non-normative Notes

- Specific interface for an application to give hints on selecting an appropriate system operating state to the PM-Engine is not required by this specification.
- Recommendations for Hardware Platform
  - The latency of changing power modes of a CPU should be small enough to do it whenever a CPU becomes idle.
  - Memory address/data buses and peripheral buses can be put to low power modes when CPU becomes idle if the latency of putting those buses to low power modes are small enough.
  - CPU voltage/clock changing latency need to be quite small enough to do frequently while a task is running.

4.6.4 References

Power management framework - lower layer of PM-Engine:

1. IBM and MontaVista Software, "Dynamic Power Management for Embedded Systems", Nov., 2002, http://www.research.ibm.com/arl/projects/papers/DPM_V1.1.pdf
2. "Dynamic Power Management", http://sourceforge.net/projects/dynamicpower/

Hooks required in tracking task scheduling etc.:

3. "Kernel Hooks", http://www-124.ibm.com/developerworks/oss/linux/projects/kernelhooks/

4.7 Variable Scheduling Timeouts

4.7.1 Rationale

The Linux kernel relies on periodic timer interrupts to allow the kernel:

- to keep the current time and date up to date,
- to do accounting,

- to check if a different task should be dispatched, and
- to see if software timer service routines should be called.

Even when the processor is otherwise idle, these interrupts occur and cause the processor to be brought out of a power savings mode even though there may not be anything for it to do other than update the time and go back to sleep. The Variable Scheduling Timeouts (VST) feature modifies the kernel to avoid unnecessary timer interrupts, allowing the processor to stay in a power saving mode for longer periods.

4.7.2 Specification

A kernel SHOULD disable periodic timer interrupt while CPU is idle and there is no scheduled timer. The length of the time interval during which periodic timer interrupt is disabled SHOULD not be more than a value which SHOULD be configurable.

4.7.3 Notes

More information on VST is available at the community site http://sourceforge.net/projects/high-res-timers/

4.8 Work in Progress

4.8.1 Platform suspend/resume

4.8.1.1 Protocol between Energy-Aware Application and Policy Manager

1. Energy-aware(EA) application SHOULD register itself to policy manager.
2. Policy manager SHOULD send system suspend request to all applications registered as EA application.
3. Energy-aware application SHOULD respond to the system suspend request from policy manager appropriately.
    a. If EA application agree to system suspend, it invokes its suspend handler.
        i. The suspend handler may store important data including user data to a non-volatile storage.
        ii. The suspend handler SHOULD send acknowledgement to policy manager and SHOULD wait until it receives resume request from policy manager.
    b. If EA application disagree to system suspend, it SHOULD send negative acknowledgement to policy manager.
4. If system inactivity which is longer than system timeout threshold is detected, policy manager MUST send system suspend request to EA applications and wait for responses from EA applications.
    a. The policy manager SHOULD provide user interface for initializing the system timeout threshold.
    b. The policy manager SHOULD initialize system timeout threshold to a default value.
    c. The policy manager MAY check device inactivity by periodically reading last access time of device.
5. If a user has requested system suspend, policy manager SHOULD enforce every EA application to suspend itself and wait for responses from EA applications. An EA application which is enforced to suspend by policy manager SHOULD invoke its suspend handler unconditionally.

4.8.2 Device Power Management

4.8.2.1 Automatic Device Power Saving

1. The policy manager set the length of inactivity period for each device.
2. The period of device inactivity can be detected by PM-engine/device drivers and notified to policy manager via signal.

4.8.3 Dynamic Power Management

4.8.3.1 Energy-Aware(EA) Application Specification

This specification shall define requirements of energy-aware applications so that application can interact tightly with in-kernel PM-engine to control system power states dynamically adaptive to various application specific environments.

EA application may choose one appropriate operating state among system PM policy. The system policy may be given to the EA application by policy manager or hard-coded, which is dependent on design decisions by system administrator. If the system policy information should be given by policy manager, EA application need to set up a IPC mechanism with the policy manager while it starts up.

For real-time EA applications, the information regarding the real-time task scheduling such as the period of execution and their priorities should be given to in-kernel PM engine via some whatever existing API which we have to choose after extensive discussion on pros and cons of existing APIs. (Linux kernel need to support scheduling periodic real-time tasks.)

There are some cases that an EA application need to change system operating state according to the task's environment. This situation is highly dependent on system constraints which system administrator should know correctly. If a system administrator firmly believe that the operating state of an EA task need to be changed, the decision by system administrator should be respected and the in-kernel PM engine will proceed with the decision. It should be noted that system performance and power consumption efficiency might be greatly hampered if the decision of the system administrator is incorrect. So the system administrator should be very much considerate on changing application operating state.

4.8.3.2 Performance vs. Battery Lifetime Policy

1. A kernel SHOULD provide userspace interface for specifying requirements on the lower limit of battery lifetime. If the battery lifetime requirement exists, a kernel SHOULD set the operating state of the task to allocate the remaining energy resource amongst all tasks, based on the task priorities, to satisfy this requirement.

5 Audio/Video/Graphics Specification

5.1 Introduction

Audio, video, and graphics processing is at the core of many CE products. The AVG requirements for CE devices are different than those for PCs/Servers, notably with respect to footprint, input devices, interlacing, streaming, etc.. Multiple graphics planes and video planes may be combined using, e.g., alpha blending and animation.

5.2 Rationale

No single default/standard interfaces exist for AVG. Having a well defined, well supported interface for AVG devices will reduce fragmentation of solutions and encourage the CE community to develop solutions that apply to conforming interfaces, so that they can be deployed across a wider range of systems.

5.3 Terminology

5.3.1 Acronyms and terms

Term	Definition

ALSA	Advanced Linux Sound Architecture -- functional level audio API, now standard in 2.6 Linux kernels, replacing OSS.
API	Application Programmers Interface
ARIB	Association of Radio Industries and Businesses. Most relevant to AVG is the proposed graphics architecture proposed for High Definition TV Broadcast (the 5-plane model).
ATSC	Advanced Television Systems Committee. American standard body for digital television broadcasting.
Back-end Scaler	A Scaler which manipulates the graphics planes and data, but does not allow the host processor access to the (blended) end result, mainly for efficiency reasons.
CCIR 601	In 1982 CCIR 601 established a digital video standard, which uses the Y, Cr, Cb color space (often incorrectly referred to as YUV). Unlike YUV, Cr,Cb range [-0.5, -0.5]. A full conversion matrix is included below (*)
CE	Consumer Electronics: a class of devices used in the home or on the move. Includes DVD, DVR, PVR, PDA, TV, set-top box, cellular phones, etc.
DVB	Digital Video Broadcast: European standards body for digital television broadcasting.
DVD	Digital Versatile Disc: high capacity multimedia data storage medium.
DVR	Digital Video Recorder: a consumer electronic device.
FB,Framebuffer	Abstraction of video-out hardware with a low level (ioctl) API. Standard in >2.4 Linux kernel (see the /usr/src/linux/Documentation/fb kernel tree directory for more information).
Front-end Scaler	A Scaler which manipulates the graphics planes and data and allows the host processor access the (in-between and) end results.
HDTV	High Definition Television: provides a higher quality television broadcast, with progressive and interlaced (720p to 1080i) video and support for 16:9 aspect (movie) ratio.
JPEG	Joint Photographic Experts Group: (lossy) still image compression standard.
MHP	Multimedia Home Platform: an API used together with MPEG-2 transmissions.
MIME	Multipurpose Internet Mail Extension: a standard for identifying the type of data contained in a file. MIME is an Internet protocol that allows sending binary files across the Internet as attachments to e-mail messages. This includes graphics, photos, sound, video files, and formatted text documents.
MP3	MPEG-1 Audio Layer 3: a popular audio compression standard.
MPEG-1/2/4	Moving Picture Experts Group: a compression standard for digital audio & video with varying levels of complexity and achievable compression ratios.
NTSC	National Television Systems Committee: American standard for analog television broadcasting.
PAL	Phase Alternating Line: American standard for analog television broadcasting.
PNG	Portable Network Graphics: (lossless) still image compression standard.
PVR	Personal Video Recorder: a consumer electronic device.

RGB[A]	Colorspace representation commonly used in computer graphics. It uses three orthogonal components -- Red, Green and Blue -- to represent colors in to human visible spectrum, e.g. by combining red and green as additive colors it can fool the eye into seeing "yellow" light. An optional A at the end denotes the presence of per-pixel alpha. See also CCIR 601.
Scaler	Graphics hardware accelerator which may scale and reformat (e.g. convert from YCC to RGB) graphics data and merge multiple independent graphics planes for final display.
V4L	Video for Linux: low level (ioctl) video input and overlay API, standard in 2.4. Originally designed for control of analog video capture and tuner cards, as well as parallel port and USB video cameras. Incorporated in many other higher level APIs such as DirectFB.
V4L2	Video for Linux, second version, made to be more flexible and extensible. Added specifications for digital tuner control and capture.
YCbCr[A]	Colorspace representation commonly used in analog and digital video broadcasts, and video compression technologies such as MPEG. It uses three orthogonal components, one for luminance (Y) and two for the color-difference signals (Cr,Cb). Since the eye is less sensitive to color than luminance, the color difference signals often get a smaller bandwidth allocated (or lower pixel resolution in the digital domain). An optional A at the end denotes the presence of per-pixel alpha. See also CCIR 601.
YIQ	Colorspace representation commonly used in North American TV broadcast and is similar to YUV (see definition of YUV). The relation with YUV is: I = 0.74 V - 0.27 U and Q = 0.48 V + 0.41 U
YUV	Colorspace representation commonly used in European TV broadcast. It is similar to YCbCr and often meant to be the same (incorrectly) with U referring to Cb and V referring to Cr. With Y (luminance) defined as Y=0.299 R + 0.587 G + 0.114 B, by definition, U=B-Y, thus U represents colors from blue (U>0) to yellow (U<0). Likewise V=R-Y, thus V represents colors from magenta (V>0) to Cyan (blue green) (V<0).

(*) RGB to YCbCr conversion matrix:

Y		0.299 0.587 0.114	R
Cr	=	0.500 -0.419 -0.081	G
Cb		-0.169 -0.331 0.500	B

5.3.2 Compliance classifiers

Terminology conventions are adopted here as they are defined in IETF RFC 2119, "Key words for use in RFCs to Indicate Requirement Levels" (by S. Bradner, March 1997). A compliance classifier from the following set may be used:

- [M]ust, Required, Shall: This is the minimum set of requirements. The CELF based products are expected to comply with these requirements when expressed in unconditional form. A conditional requirement expressed in the form, "If X, then Y must be implemented", means that the requirement "Y" must be met when the conditional aspect "X" applies to a given implementation.
- [S]hould, Recommended: Recommended items are optional items that are strongly recommended for inclusion in CELF based products. The difference between "recommended" items and "optional" items, below, is one of priority. When considering features for inclusion in a product, recommended items should be included first.
- [O]ptional, May: Optional items are suggestions for features that will enhance the user experience or are offered as a less preferred choice relative to another recommended feature. If optional features are included, they should

comply with the requirement to ensure interoperability with other implementations.
- E[X]pressly Forbidden: This term means that an item must not be incorporated in a CELF based product.

5.4 Platforms

[O] Three target platforms are used or under consideration:
- Renesas SH4 host with SM501 graphics (See SzwgPlatform3)
- TI OMAP (See SzwgPlatform1)
- X86 generic with Matrox G450/550

For the first two, the SystemSizeSpec_R2 page has a full description under "Definition - Platform".

5.5 Audio Specification

[O] No additional Audio specifications have been defined. ALSA, defined in kernel 2.6, may be used. Further evaluation is required before it can be considered for recommendation (see work in progress). Future extensions relate to AV streaming and synchronization.

5.6 Video-in/Capture Specification

[O] No additional Video input (capture) specifications have been defined. V4L2, as defined in kernel 2.6, may be used.

[O] Proprietary solutions may also be used for video capture and digital tuners if V4L2 does not suffice.

[O] DirectFB may be used as a higher level API.

Note: Video output can be seen as an (interlaced) sub-set of graphics. See graphics specification below for more details.

5.7 Video-out/Graphics Specification

[S] The standard Framebuffer is recommended for use in embedded CE devices.

[O] DirectFB may also be used in combination with the framebuffer.

Extensions to both are under consideration (see work in progress).

5.7.1 Graphics formats

[O] The framebuffer supports CLUT, RGB and RGBA packet data formats, but not YCbCr[A]. Hardware capable of accelerating the display YCbCr[A] packed data may develop their own extensions to the framebuffer for now.

[O] Also, the DirectFB framework which supports these formats may be used.

5.7.2 Multi-plane support

[O] Graphics hardware capable of multiple planes may be implemented with a single or multiple device drivers with one device per plane e.g. /dev/fb0, /dev/fb1,.../dev/fb5 for a 5 plane capable device. Front-end based scalers are recommended to use the DirectFB framework.

[O] Back-end scalers may add ioctl's to their framebuffer drivers.

5.8 Work in progress

Both DirectFB and the Framebuffer **can** be extended with YCbCr formats and multi-plane blending features commonly found in embedded CE devices. However, it is likely that only one of them will be supported in the future.

5.8.1 Framebuffer specification

5.8.1.1 YCbCr Format

5.8.1.1.1 Resolution Support

The recommended formats are:

- 4:4:4 Equal number of samples of Y, Cb and Cr.
- 4:2:2 Cb/Cr are subsampled by a factor of two horizontally.
- 4:2:0 Cb/Cr are subsampled by a factor of two in both directions.
- 4:1:1 Cb/Cr are subsampled by a factor of four horizontally (used in DV).

If any of these formats are used, the CCIR 601 standard must be used. It defines how the data is interleaved and the relative positions of the Cb/Cr samples in relation to the Y samples.

5.8.1.1.2 Memory representation

YCbCr may be stored in e.g a framebuffer in various ways:

- packed YCbCrA 4:4:4 : 32-bit unit containing one pixel with alpha
- packed YCbCr 4:2:2 : 16-bit unit, two successive units contain two horizontally adjacent pixels, no alpha
- planar YCbCr[A] 4:2:2 : three [four] arrays, one for each component
- semi-planar YCbCr 4:2:2 : two arrays, one with all Ys, one with Cb and Cr.
- planar YCbCr[A] 4:2:0 : three [four] arrays, one for each component
- semi-planar YCbCr 4:2:0 : two arrays, one with all Ys, one with U and Vs

Following CCIR601, only the packed formats are recommended, with the possible exception of a separate alpha plane in some cases (see ARIB [O6] proposal).

5.8.1.2 Font rendering

- freetype [O5]

5.8.1.3 Basic 2D acceleration

- lines (horz./vert. vs. anti-aliased lines)
- rectangles (fill and copy)
- pixmaps (bitblt, scaling)

5.8.1.4 Video format control

- resolution
- interlaced/progressive

5.8.1.5 Multi-plane support

- Each plane is represented by... [/dev/fb0, /dev/fb1,...]

- Additional API (ioctl) calls... [display order, placement, scaling,...]

5.8.2 DirectFB specification

DirectFB overview [G2] provides a list of currently supported features, summarized below.

5.8.2.1 Important Terminology

5.8.2.1.1 Surface

Memory region physically reserved for rendering pixels. Surfaces are used for regular rendering of pixels, sprites and so on.

5.8.2.1.2 Sub-surface

Sub-region of surface. No physical memory allocated.

5.8.2.1.3 Primary Surface

Visible screen in full screen mode.

5.8.2.1.4 Layer

Each layer is different video memory. They are alpha-blended and displayed.

5.8.2.1.5 Window/Windowstack

Each layer may have multiple window. Windowstack is a stack of windows. Each window has surface. Their locations and orders may be changed.

5.8.2.2 YCbCr Format

5.8.2.2.1 Resolution Support

Supported formats are:

- 4:2:2 Cb/Cr are subsampled by a factor of two horizontally.
- 4:2:0 Cb/Cr are subsampled by a factor of two in both directions.

5.8.2.2.2 Memory representation

- packed YCbCr 4:2:2 : 16-bit unit, two successive units contain two horizontally adjacent pixels, no alpha
- planar YCbCr 4:2:0 : three arrays, one for each component

5.8.2.3 Font rendering

- DirectFB bitmap font
- TrueType (using FreeType2)
- No bold or italics support other than by specifying a different typeface from the same font family.(*)

(*) For example, 'Times New Roman Regular' and 'Times New Roman Italic' correspond to two different faces.

5.8.2.4 Basic 2D acceleration

- lines (anti-aliased)
- rectangles (fill and copy)
- triangle (fill and copy)
- pixmaps (bitblt, scaling)

- Per-pixel alpha blending (a.k.a. texture alpha)
- Per-plane alpha blending (a.k.a. alpha modulation)
- Colorizing (a.k.a. color modulation)
- Source and destination color keying

5.8.2.5 Video format control

- resolution
- interlaced/progressive support

5.8.2.6 Multi-plane support

- DirectFB layers (not surfaces) support the concept of planes.
- Layer API is provided through IDirectFBDisplayLayer Interface.
- Opacity is available through IDirectFBDisplayLayer::SetOpacity.
- IDirectFBDisplayLayer::SetScreenLocation() controls scaling of the plane. Back-End Scaler(BES) is used, for instance for Matrox. It requires hardware support.
- Explicit Front-End Scaler(FES) is not available. Thus, stretched blit to the primary surface should be used.
- To execute a specific graphics operation (e.g. blitting of a surface), the DirectFB driver will access the memory mapped io ports of the graphics hardware to submit the command to the acceleration engine (actual hardware acceleration is done entirely from user space).

5.8.2.7 GFX Card Driver

- DirectFB abstracts the video driver through GFX driver.
- Graphic operation is executed through IDirectFBSurface interface. The interface calls appropriate callback routine in gfxcard driver(src/core/gfxcard.c). The callback routine decides whether the video device has hardware acceleration capability or not, and invokes appropriate functions.
- The following is the model used in the core gfxcard driver. Blit, DrawLine, DrawRect and similar operations are implemented in this way:

```
void dfb_gfxcard_OPERATION()
{
        bool hw = false;
        lock();
        /* check if acceleration is available, and then acquire  */
        if (hardware_accel_available(OPERATION) && hardware_accel_acquire(OPERATION)) {
                hw = card->funcs.OPERATION();
        }
        /* if hardware acceleration is not available */
        if (!hw) {
                gAcquire();
                gOPERATION();
                gRelease();
        }
        unlock();
}
```

5.8.2.8 DirectFb benchmarks

You can refer 'DirectFB' benchmark on various environment from Benchmark section of **EvaluateDirectFbTaskPage**

5.9 References

G - Graphics/Video out:

1 Framebuffer

- http://www.kernel.org/ (1) KD26/fb
- http://linuxconsole.sourceforge.net/fbdev/HOWTO/
- http://www.tldp.org/HOWTO/Framebuffer-HOWTO.html

2 DirectFB (uses 1)

- http://www.directfb.org/
- http://www.directfb.org/documentation/DirectFB_overview_V0.2.pdf

3 NanoX

- http://www.microwindows.org/

4 SDL

- http://www.libsdl.org/

5 Gstreamer

- http://www.gstreamer.net/

6 OpenGL (OpenML)

- http://www.opengl.org/
- http://www.khronos.org/opengles/

V – Video in:

1 V4L[2]

- http://www.kernel.org/ (1) KD26/video4linux
- http://bytesex.org/v4l/

2 OpenML

- http://www.khronos.org/openml/

3 LinuxTV (DVB API)

- http://www.linuxtv.org

A – Audio in/out:

1 OSS

- http://www.kernel.org/ (1) KD26/sound/oss
- http://www.4front-tech.com/opensound.html

2 ALSA

- http://www.kernel.org/ (1) KD26/sound/alsa
- http://www.alsa-project.org

3 OpenAL

- http://www.openal.org/

U – Users of AVG:

1 VideoLan

- http://www.videolan.org

2 Freevo

- http://freevo.sourceforge.net

3 LinuxTV

- http://www.linuxtv.org/

4 MythTV

- http://www.mythtv.org/

5 DVR

- http://dvr.sourceforge.net/html/main.html

6 OpenPVR

- http://www.funktronics.ca/openpvr/
- http://sourceforge.net/projects/openpvr/

O – Other:

1 TV Linux Alliance

- http://www.tvlinuxalliance.com/

2 TV Anytime

- http://www.tv-anytime.org/

3 Digital Home Working Group

- http://www.dhwg.org/

4 BootSplash

- http://www.BootSplash.org/

5 FreeType

- http://freetype.sourceforge.net/freetype2/

6 ARIB architecture

- http://www.arib.or.jp/english/html/overview/ov/std_b24.html

Note (1) - KD26 refers to the Linux 2.6.X kernel tree, which has a "Documentation" sub-directory.

5.10 Remaining Issues

See Work in progress.

6 Realtime Specification

6.1 Introduction

The goal of the real-time working group is to improve the real-time capabilities of the Linux operating system to address the range of real-time requirements for consumer electronics devices.

6.1.1 Rationale

Consumer electronics devices need to perform many functions interactively and smoothly to meet user-expectations. For example, streaming data needs to be presented to the user without interruptions that would degrade the user's experience. Likewise, a recording device needs to record streaming audio/video data without loss of data. Responses to interactive user commands need to be completed quickly enough that the interface seems responsive to the user.

As an illustration, consider a set-top box playing a movie. The flow of audio and video data through the system has real-time requirements. As each frame of data is accessed from a hard drive it is delivered to subsequent systems (such as decoders, decompressors, decryptors, etc.) which manipulate the data stream and prepare the data for output. The hardware controlling the television signal must receive each frame of data at a fixed frequency in order to present each image in sequence. If a process in the chain of handlers cannot run due to overload of the system or scheduling problems, a frame might be delayed and lost, or the delay might result in the appearance of “glitches” or jerkiness in the images presented to the user.

6.1.2 Scope

This specification is primarily concerned with the Linux kernel, but Linux systems often provide support for functional API's using a mixture of C-library and the Linux kernel system calls. This is especially true for many of POSIX API's, including those that are part of this specification. The appropriate support for the various POSIX functionality elements in this specification may or may not exist in the various C-libraries available for Linux.

Broadly speaking, this specification focuses on the following improvements to the Linux kernel; all designed to make Linux a better real-time operating system.

- Enhance support for fixed priority preemptive scheduling.

  From an operating system perspective, this implies that the operating system should allow the system designer/administrator to specify the priorities of different activities in the system. Additionally, the operating system should minimize priority inversion in the system.

- Enhance coverage of POSIX real-time API's supported by Linux

  POSIX API's, including various real-time API's have become de-facto standard in the operating system communities and are supported by many general purpose as well as specialized real-time operating systems. Good support for POSIX real-time API's provides a convenient, well-understood API for middleware and application developers.

The Real-Time Working Group recognizes that there is a strong community of developers who believe that the "correct way" to do real-time in Linux is to do it with a "real-time kernel" such as RTAI or RTLinux such that any "real-time task" is effectively prioritized above any Linux activity. This version of the specification has not focused on these "hybrid" techniques, but it does not prevent the use of these hybrid techniques either.

6.2 Terminology

Context switch:
: the act of replacing the running task *A* by another task *B*. The state of task *A* is saved, and *A* is placed in the ready queue; the state of task *B* is restored, and *B* becomes a running task.

Critical section:
: a brief interval during which a task accesses shared data. During this interval, no other tasks are allowed to access the same data. One way of ensuring this is to prevent preemption during the critical section. If the critical section defers preemption for a bounded interval, the resulting priority inversion is bounded.

Deadline-monotonic priority setting:
: the task with the shortest relative deadline is assigned the highest priority.

Deadline requirement:
: a real-time requirement that requires the response to be completed within a fixed time interval from the triggering event. A *relative deadline* is the duration of the above mentioned interval; an *absolute deadline* is the moment in time at which the response must be completed.

Fixed priority preemptive scheduling:

the task with the highest priority is always running. If a task with a higher priority becomes runnable, the running task will be preempted immediately. The priority of a task, process or Interrupt Service Routine (ISR) is explicitly determined at creation, or by an explicit set-priority command. No implicit priority changes by the scheduler are assumed. For an exception to this rule see **Priority inheritance.**

Note: Fixed priority scheduling is typically designed to be used for a single coherent application.

Granularity:

the time range of a certain interval. We can talk about the granularity of a timing requirement, of a non-interruptible code segment, etc.

Hard deadline requirement:

missing the deadline is considered an error.

Hard real-time system:

system with hard real-time requirements.

Hybrid real-time system:

system with both hard and soft real-time requirements. CE systems are expected to be of this kind. The challenge in designing a hybrid real-time system is to get good soft real-time performance while meeting the hard real-time requirements. This is typically achieved by using a technique called reservation.

Interrupt latency:

time passed between interrupt occurrence and activation of interrupt handler.

Interrupt masking:

Making certain interrupts invisible to the software.

Interrupt response time (worst-case):

(worst-case) time passed between interrupt occurrence and either completion of interrupt service routine (ISR) or wake up of dependent task.

Jitter – absolute:

deviation of the occurrence of an event (e.g. completion of frame) from expected occurrence.

Jitter – relative:

deviation of the interval between two successive occurrences of an event (e.g. completion of frame) from expected interval.

Mutual exclusion:

prevent multiple tasks or ISRs from accessing the same data concurrently. Mutual exclusion is used to protect the integrity of the data.

Preemption:

a running thread or process can be temporarily suspended. The state of the thread or process (including e.g., program counter, and register values) is saved. Until the thread is resumed, it remains runnable (active, ready). When the process or thread is later resumed, the saved state is restored.

Priority inheritance:

if a high priority-task blocks for a critical section, a low-priority task that has holds the lock for the section gets a priority boost. It inherits the priority of the blocked task. This prevents the unbounded priority inversion that could occur if medium priority tasks would preempt the lower-priority task that holds the lock. Note that, with priority inheritance, the medium-priority tasks suffer priority inversion as well. But, and this is important in real-time systems, the priority inversion for all tasks is bounded (can be determined without knowing the exact run-time schedule)

Priority inversion:

the highest priority task is not running. There can be several reasons for priority inversion. One of them is the absence of full preemption. Priority inversion is one of the main reasons for deadlines being missed.

Real-time requirement:

a requirement on the completion time of a response, generally measured relative to the event that triggered the response.

Real-time system:

system with one or more real-time requirements.

Response time (worst-case):

(worst-case) time passed between event occurrence and completion of the response to that event. The event may be an interrupt. The response typically involves an interrupt handler and one or more synchronized tasks.

Runnable task:

(also ready/active task) a task that can run from a logical perspective, but is prevented from running physically.

Semaphore:

a synchronization primitive often used to achieve mutual exclusion.

Soft deadline:

missing deadlines is sometimes acceptable. Compared to hard deadlines, where there is no reason to consider the value of a late result, the value of a late result for a soft deadline is of interest. The value of the result may, for instance, decrease linearly after the deadline.

Soft real-time requirement:

soft deadline, or average-case response time requirement.

Note that hard and soft real-time requirements are orthogonal to the temporal granularity that is required. Meeting a soft requirement in the microsecond domain may be more difficult than meeting a hard requirement in the milliseconds domain.

Soft real-time system:

system with soft real-time requirements

6.3 Preemptible Kernel

6.3.1 Introduction

Even though Linux allows preemptive priority scheduling of processes, the Linux kernel itself is non-preemptible through version 2.4.x. A preemptible kernel allows a (low-priority) process to be preempted even while it is executing in the kernel.

6.3.1.1 Rationale

Fixed priority preemptive scheduling for real-time tasks is considered a basic characteristic of real-time operating systems. This scheduling behavior is also specified in the POSIX real-time scheduling specifications. Linux has supported POSIX real-time scheduling classes (SCHED_FIFO and SCHED_RR) for a long time, but real-time tasks in Linux can still experience significant amounts of priority inversion. One major cause of this priority inversion has been that while Linux supports fixed priority preemptive scheduling, the Linux kernel itself has been non-preemptible up until version 2.4.x. Making the Linux kernel preemptible is a significant step towards enhancing the real-time capabilities of the Linux kernel.

6.3.2 Specifications

During normal operation of the system the kernel shall be preemptible except under the circumstances indicated below.

1. Normal operation of the system shall include the execution of the system after the boot process is complete and before shutdown is initiated.
2. Preemption may be disabled when the kernel is running inside a critical section.
3. Preemption may be disabled when the kernel is running in a non process context.
4. Preemption may be disabled when the kernel is being interactively debugged.

6.3.3 Notes (Informational and Non-Normative)

While the specification takes steps towards improving the real-time characteristics, the user still needs to be aware of the potential for significant priority inversion due to the possibilities of preemption being disabled in the kernel under various circumstances. There are various techniques to further improve the kernel preemptibility.

One such technique is to divide (large) critical sections in the kernel into multiple (shorter) critical sections. Various kernel patches that do this are often termed as "low-latency" or "lock-breaking" patches. Several such patches have been available for 2.4 Linux kernel. Many of those patches were adapted and integrated into 2.6 Linux kernel.

Another technique is to enable preemption within some critical sections by protecting the critical section using a semaphore or mutex (ideally with priority inheritance support).

6.3.4 References

Support for preemptible Linux kernel is integrated into the CELF kernel.

Patches for 2.4 Linux kernels are generally also available from http://kpreempt.sourceforge.net

Support for Linux kernel preemption is also integrated in 2.6 Linux kernels.

6.4 O(1) Scheduler

6.4.1 Introduction

The 2.4 Linux scheduler maintains the active queue as a list of processes sorted by priority. Inserting a process in this list is an *O(n)* operation, where *n* is the number of active (runnable) tasks. Since insertion in the active queue is a component of preemption time, this makes preemption time dependent on the number of active processes in the system. Schedulers with performance that is independent of the number of active processes have been available for years, and the standard Linux 2.6 scheduler meets that requirement.

6.4.1.1 Rationale

Preemption time is an important factor in real-time performance, and directly effects real-time performance metrics such as interrupt response time. When the scheduler can take time that is proportional to the number of tasks in the system, then the interrupt response time is adversely effected when there are a large number of tasks in the system. While many CE devices may not benefit from a constant-time scheduler (given that they often have a limited number of tasks), it is expected that higher end CE-devices may very well have a significant number of tasks, where using a constant-time scheduler would be a necessary requirement to achieve good real-time behavior.

6.4.2 Specification

The Linux kernel source SHALL include a scheduler whose performance is independent of the number of tasks (i.e., O (1)) in the system.

6.4.3 Notes

The reference *O(1)* scheduling mechanism is *O(1)* in the number of active tasks. It is not independent of the number of priorities. It is also worth noting that a kernel that uses a "constant-time" scheduler may still have context switching performance that depends on the number of tasks. This specification only requires that the algorithmic performance of the operations on the active queue (such as wakeup, schedule, etc.) does not depend on the number of tasks in the system. Other factors such as maintenance of the TLB, maintenance of the cache, demand paging working sets, and OS hints may cause large dependencies on the number or complexity of active processes. These issues are of great interest for real-time systems with small scheduling granularity, but are outside the scope of this specification.

It is also important to note that a constant-time scheduler offers no benefits when the number of active tasks in the system is known to be small. Furthermore, it is likely that a constant-time scheduler has higher overheads.

6.4.4 References

An *O(1)* scheduler is integrated into the CELF kernel.

The scheduler in 2.6 Linux kernels is also *O(1)*. It maintains an array of double-ended linked lists, with each array entry corresponding to a priority. It can, thus, enqueue and dequeue tasks without a search that depends on the number of tasks. The *O(1)* scheduler in the CELF kernel is a backport of the 2.6 Linux kernel's *O(1)* scheduler.

6.5 Interrupt Threads

6.5.1 Introduction

Interrupt service routines executing at hardware interrupt level (or hard-irq context, as it is called in Linux) are effectively prioritized above tasks in the system. Interrupt Threads allows an ISR to be executed within a kernel thread context.

6.5.1.1 Rationale

Even with a preemptible kernel, preemption is disabled when the kernel is executing in interrupt (hard-irq) context. When the interrupt load is small, this does not have a significant effect on the latencies for real-time tasks.

However, for various reasons, it may not always be possible to keep the interrupt load small. While system implementors are encouraged to keep interrupt service routines brief, this is not always possible. For example, sometimes a device does not support DMA properly, and a driver may have to move a significant amount of data in hard interrupt context. Or, a device that is generating very frequent requests may cause numerous brief interrupt service routines to sum to an extended amount of processing in the hard interrupt context. Whatever the reason, heavy use of interrupt context can adversely affect latencies for real-time tasks.

Especially in a system like Linux, where "off the shelf" device drivers are a major attraction, it is important to have a mechanism that makes unmodified interrupt service routines execute in a thread context. With this capability, a system designer can prioritize the execution of interrupt service routines lower than critical real-time tasks.

6.5.2 Specification

1. The kernel SHALL provide a configurable option to execute the interrupt service routine in a kernel thread. With this option enabled, a separate kernel thread SHALL be used to service each interrupt line (IRQ).
2. The kernel SHALL provide a "flag" that can be used by a device-driver to specify that a particular interrupt service routine should not be executed in a thread context.
3. The kernel MAY provide configuration time specification of thread priorities for threads supporting the IRQ-thread option for each IRQ. When no such specification is provided, the thread priorities will default to SCHED_FIFO scheduling class with priority set to one less than the maximum real-time scheduling priority.
4. No modifications SHOULD be required to device drivers to execute their interrupt service routine in thread context.

6.5.3 Notes

Device drivers may implement parts of their interrupt service routines in kernel threads, either by using the workqueue abstraction in 2.6 Linux (and backported to CELF 2.4 kernel), or by directly using kernel threads. This facility adds the flexibility of executing ISRs of unmodified device drivers in kernel threads. Additionally, it allows the system designer (except when expressly disallowed by the device driver) to determine whether the interrupt service routine should execute in hard-irq context or in thread context.

It is worth noting that executing interrupt service threads in kernel threads adds little or no value unless the Linux kernel is preemptible. Therefore, it is recommended that this facility be used in conjunction with preemptible kernel. Also, in most cases, it would be desirable to also use the SoftIRQ Threads facility as well.

6.5.4 References

Kernel patches that implement Interrupt Threads on 2.6 Linux kernel have been posted to the CELF-RTWG mailing list. These patches are also available through the CELF-RTWG public wiki pages.

6.6 Soft-IRQ Threads

6.6.1 Introduction

Linux uses a softirq mechanism to execute code in system context with interrupts enabled. This is often used by the Linux kernel as a way to execute functions that are not directly related to user tasks.

6.6.1.1 Rationale

The soft IRQ facility runs its load in interrupt state until the load becomes too great, then it schedules it in thread context at a low priority. Neither heavy use of interrupt mode nor hard-to-predict demotion of interrupt-priority work to a low priority is good practice for a real-time OS.

6.6.2 Specification

- The kernel SHALL provide a configurable option to run the soft IRQ actions in one or more kernel threads.

6.6.3 References

The CELF kernel includes a mechanism to emulate Soft IRQ actions using a workqueue, which essentially executes the actions using a kernel thread.

Patches to execute Soft IRQ actions using kernel threads for 2.6 Linux kernel have been posted on the CELF-RTWG mailing list, and are also available on the CELF-RTWG public wiki site.

6.7 POSIX Timers

6.7.1 Introduction

The passage of time is, almost by definition, of interest to real-time software. Specific examples include:

- Periodic threads with or without deadlines
- Watchdog timers
- Sporadic servers
- Other CPU-consumption budgeting.
- Deadlines for aperiodic processing

6.7.1.1 Rationale

The POSIX specification is mature and generally accepted. It includes a set of time-related APIs that provide a strong basis to build software that needs these services. The POSIX timers functions are adopted for this specification without modification.

6.7.2 Specification

1. The kernel MUST conform to the POSIX specification IEEE Std 1003.1-2001 for timers. This includes functions marked with the following margin codes:
   - The TMR margin code for POSIX timers
   - The CS margin code for clock selection
   - The MON margin code for a monotonic clock
2. The CPT margin code for process CPU time clocks is not required.
3. The kernel timer support MAY include support for one or more clocks with high-resolution utilizing hardware counters or timers to achieve timer resolution with 100us or lower.

6.7.3 Notes

The list of required functions includes:

- clock_gettime, clock_settime, and clock_getres
- timer_create, timer_delete, timer_settime, timer_gettime, timer_getoverrun
- nanosleep, clock_nanosleep

6.7.4 References

Support for POSIX timers is included in the CELF kernel. This also includes support for high-resolution timers on selected architectures.

Support for POSIX timers is also included in 2.6 Linux kernels. A patch for high-resolution timers is available at http://high-res-timers.sourceforge.net.

6.8 POSIX Message Queues

6.8.1 Introduction

Message passing is a convenient and powerful communication mechanism that is widely used by real-time applications.

6.8.1.1 Rationale

Linux has supported the System V Message Passing API for a long time, but the System V Message Passing facility does not provide for prioritized delivery of messages. Other Linux IPC mechanisms, often inherited from Unix, such as pipes, named pipes, Unix domain sockets, etc. have the same problem.

The message passing API defined in the POSIX specification is simple and widely accepted and supports prioritized delivery of messages.

6.8.2 Specification

The kernel MUST conform to the POSIX specification IEEE Std 1003.1-2001 for Message Queues. This includes functions marked with the following margin codes:

- The MSG margin code for POSIX message queues functionality
- The MSG_TMO margin code for POSIX message queues functionality with timeout capability

6.8.3 Notes

The list of required functions includes:

- mq_open, mq_close, mq_unlink
- mq_send, mq_timedsend
- mq_receive, mq_timedreceive
- mq_notify
- mq_setattr, mq_getattr

6.8.4 References

There is some code for POSIX message passing in the CELF kernel, but it is clearly not functional. It appears to be a stale snapshot of the POSIX message passing patches available from http://www-users.mat.uni.torun.pl/~wrona/posix_ipc. This project has since made significant enhancements to the code, and support is available in the form of patches for both 2.4 and 2.6 Linux kernels. As of 4/30/2004, the message queue support has been merged into the 2.6.6 kernel tree, and necessary library support has been merged into glibc 2.3.4 tree.

6.9 Priority Inheritance on User Mutexes

6.9.1 Introduction

Priority inheritance protocol is the easiest-to-use standard priority inversion avoidance algorithm. It is widely supported by standard RTOSs, and priority inheritance is specified in POSIX.

6.9.2 Specification

Mutex locks in user state SHALL support priority inheritance protocol as specified in POSIX specification IEEE Std. 1003.1-2001 under the PRIO_INHERIT symbolic constant.

6.9.3 References

Priority inheritance protocol is supported by the Robust Fast Mutex project: http://developer.osdl.org/dev/robustmutexes/

6.10 Work In Progress

This section represents some of the work that was actively discussed in the real-time working group, but is not being formally specified at this time. Nonetheless, these techniques are considered as useful extensions to the specifications and fall within the scope of this specification.

6.10.1 Interrupt priorities

This section represents interrupts that are prioritized, not the schedulable interrupt services routines addressed in the IRQ threads and Soft IRQ threads sections. In its simplest form prioritized interrupts are an issue only for the kernel's interrupt dispatching code, more specifically, on the code that operates the PIC (Programmable Interrupt Controller). It lets the system be configured with a priority order on interrupts. While an interrupt is being serviced, that interrupt and all lower-priority interrupts are masked, but higher priority interrupts may preempt the ISR. This gives more control to a real-time application than the alternatives: masking all interrupts while ISRs execute, or masking only the interrupt that is being serviced. Patches have been submitted to LKML and CELF-RTWG mailing list that implement Interrupt priorities on 2.6.

An alternate approach to dealing with interrupt priorities is provided by the ADEOS project (http://www.opersys.com/adeos). ADEOS supports the creation of an interrupt pipeline through the use of ADEOS domains. Interrupts for a lower-priority domain are not serviced unless all higher priority domains relinquish control of the CPU. ADEOS can be used to implement interrupt handlers that reside in a domain that is higher priority than Linux domain. This can be used, for e.g., to achieve very low interrupt latencies for specific devices. Also, ADEOS can be used to implement higher level facilities such as an RTAI kernel that can be used to provide "hybrid" real-time solutions.

6.10.2 Prioritized wait queues

A real-time system with FIFO wait queues cannot be protected from priority inversion, but it may not make sense to simply require that all wait queues become priority queues. Priority queues are expensive. The implementation choices are *O(n)* with simple implementation, *O(log n)* for a balanced tree or heap, or *O(1)* time for an implementation like the standard *O(1)* scheduler.

One possibility is to require that all queues in the kernel (and possibly in kernel mode), be ordered on priority of the queue entries (by default, on the priority of the task that enqueued the entry). This is aggressive, but it is the correct approach from a strict real-time viewpoint. Requiring priority queues only for blocking POSIX functions would probably require fewer queues to become priority queues, and it might be a simple way to specify "the important queues."

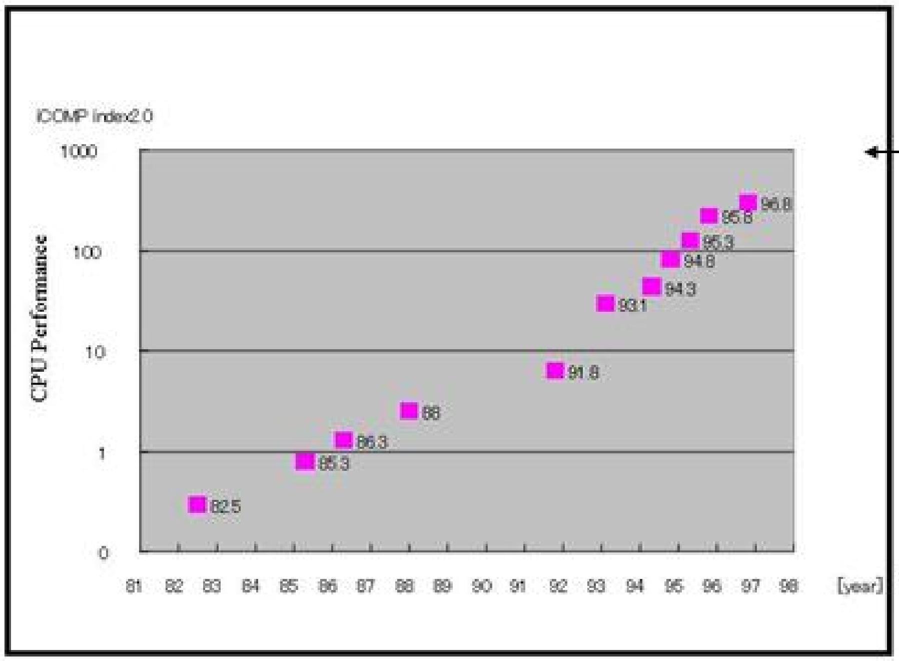
7 System Size Specification

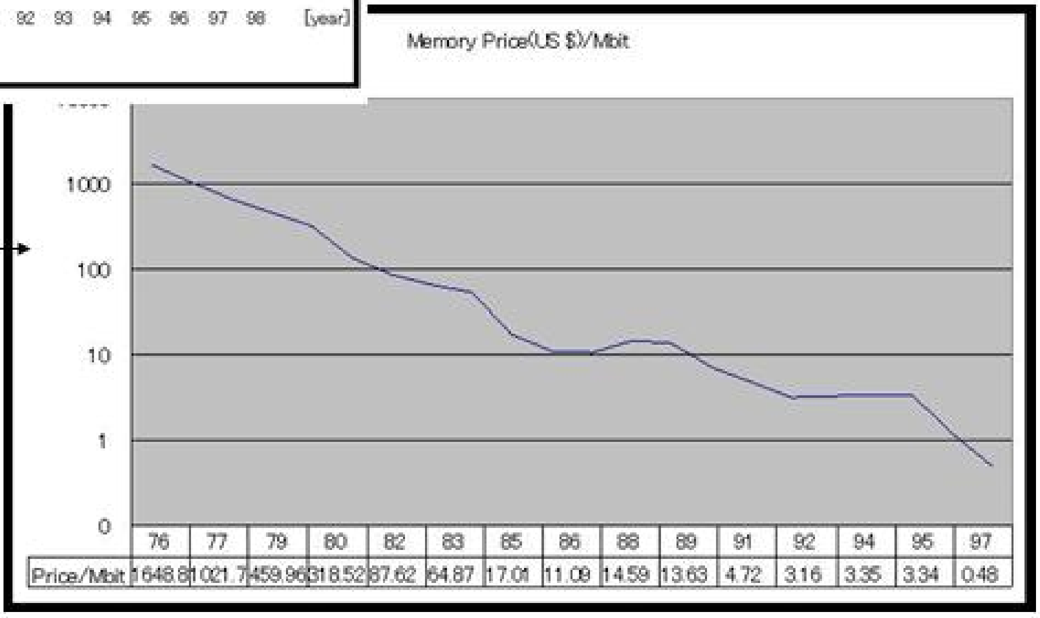
7.1 Introduction

7.1.1 Overview of "CE Linux"

Ten years ago, most people would have laughed at the idea of using Linux in embedded applications. Five years ago, some people started thinking about using Linux for non-PC applications. Today no one would be surprised at the idea of using Linux in embedded applications - in fact, there are already several real products in the field. What has changed? Some of the key changes are:

1. CE products require PC-like feature
   - It is said that we are about to enter a new world with a new life style. "Ubiquitous life" is the concept frequently used. Although "Ubiquitous" may still be too broad, everyone would agree that digital CE products - such as cellular phones, digital cameras, PVRs, DVD-Rs - are part of our life. Such CE products require pc-like digital data computing - such as GUI, network, picture or movie compression/decompression and so on. From software development point of view, it is quite natural that CE software developers would want the same or similar environment as a PC.
   - If such an environment is NOT available, software developers would have to spend extra time, which impacts time-to-market and product cost badly.
2. Open standard platform
   - If such CE products are not open to each other and have limited inter reaction or limited data sharing, the end-user benefit shall be limited. It is not good for user and it is not good for CE market either.
3. Semiconductor technology
   - The price of memory (RAM) has dropped dramatically and constantly, and embedded processor speed has risen while the price has gone down. Please see the following chart to see the trend. Please note that the scale of the vertical is "logarithmic" scale.
   - Today's $10 embedded processor performs 10+ times faster than the one used in PC desktops 10 years ago. Therefore,even if Linux is still big and slow compared to a minimal RTOS, the relative disadvantage is becoming small.





	76	77	79	80	82	83	85	86	88	89	91	92	94	95	97
Price/Mbit	1648.8	1021.7	459.96	318.52	87.62	64.87	17.01	11.09	14.59	13.63	4.72	3.16	3.35	3.34	0.48

Although Linux looks best candidate for digital CE products, there are several issues we have to address and overcome - Realtime Performance, Bootup/Shutdown time, Power Management, Audio Video, Security, System Size and so on.

7.1.2 Introduction of this document

Having said that Linux has been used in some of the actual products, Linux has issues that can not be accepted by some CE products. Size is one such issue. The hardware configuration of CE products and PCs are very different. You probably will find big differences between CE products and enterprise PCs.

-	CE Product example1	CE Product example2	Enterprise
-	Set Top Box(STB)	Cellular Phone	Desktop PC
CPU(frequency)	MIPS(30MHz)	ARM(50MHz)	Pentium(2,200MHz)
RAM size	32Mbyte	8Mbyte	512Mbyte
Flash/ROM	16Mbyte	4Mbyte	-
Storage	HDD 15Gbyte	none	HDD 80Gbyte

In the CE market, cost competition is extremely tough. Extra memory costs extra. As CE products are varied with different feature, capability, performance and/or price. CE product system architects want to find out the best sweet spot among size, feature and some other related performance. The WG have examined the existing method and characterize each method.

The purpose of this document is;

- Define a method to measure and compare the size between different Linux configurations and/or techniques which may reduce the size.
- Define a set of standard mechanisms for reducing the system size and characterize each method -- Advantage and Disadvantage.

Target audience for this document is;

- CE System architects trying to understand if CE Linux is appropriate for their size constraints and the side effect by using the method.
- OS developers looking to optimize for CE environments.
- Commercial OS developer or tool developers looking to improve existing issue.

7.2 Rationale

The WG planed to take the following steps;

- Profile Linux Size Issue
- Define the comparison method
- Examine the existing techniques and characterize each method
- Invest new technique to see the numeric differences
- Discussion on the possibility of subset feature
- Invest on the tools

We only have one profile data so far - we shall continue to collect more profile data - According to the profile, the ratio of kernel/userland is 37% which is very high number. But looking into the contents, kernel is 4% while userland/glibc is 28%. We probably have to prioritize userland issue. By the way, bigger surprise for me was that middleware needs 47%. As middleware is not SZWG scope, this topics shall be discussed at other places (maybe new WG). There are several parameters which are related to system size. XIP, for example, contribute RAM size reduction but require more ROM size and may impact execution speed negatively. We defined the items we compare and defined how to measure the number. We just finished examine the existing techniques and got the numeric number. Frankly speaking, I wonder how deeply we can analyze the data by the deadline of this draft but we'd like to open up all the data we have.

7.3 Terminology and Definition

7.3.1 Terminology

Boot Loader:

The primary role of Boot Loader is to execute Linux Kernel. Most of the Boot Loader has the debug feature, such as download hex/binary, set breakpoint, step execution, disassemble ..etc At the early stage of the development, Linux kernel (and file system) is downloaded manually by using boot loader. In the final application, boot loader(or subset of boot loader) is used to load (from ROM/Flash or HDD) and execute Linux kernel||

ICE:

In-Circuit Emulator / There are several types of ICEs. Traditional type require special CPU - so called evachip - which provides not only basic debug feature but also very rich hardware debug feature such as hardware breakpoint(e.g. data access break), pc trace,profiling, realtime RAM monitor ...etc The problems of traditional type are 1) it is becoming difficult to follow higher frequency 2.it is becoming difficult to adapt to finer pitch package 3. the price is expensive. JTAG based ICE is becoming more popular||

JTAG ICE:

JTAG was originally designed for boundary scan. It allows to read/write internal registers via limited number of pins(typically 5 pins). It is becoming popular to use JTAG interface and use DCU(Debug control Unit) inside the cpu. JTAG ICE provide very basic debug feature such as run-control, mem/reg view/edit, download ..etc||

XIP:

eXecution In Place / Usually Linux kernel is executed on RAM in order to get best performance. XIP is, however, the technique that Kernel is executed on ROM. XIP has the advantage of fast boot and smaller RAM usage while it has the disadvantage of slower performance.

7.3.2 Definition

7.3.2.1 Platform

Since embedded CE products are varied - in sense of cpu ,hw configuration and so on - it is difficult to pick one platform. Instead of pick one platform as "standard", we selected one "primary" platform and several other as "case studies". TI/Omap Innovator was selected as "primary" platform.

Primary Platform

- TI/Omap Innovator SzwgPlatform1

Case Study Platform

- KMC SH4 SzwgPlatform2
- Renesas SH4 SzwgPlatform3
- NEC Electronics VR5500A Core SOC SzwgPlatform4

7.3.2.2 Measurement Method

When we examine the technique, we have to evaluate several points. We need to clarify whether the technique is good for ROM or RAM. We need to clarify whether it is good for Kernel, File System or both. Also, we need to be careful about the side effect caused by the technique. We defined the following six(6) items to characterize the method.

- Kernel ROM size
- Kernel RAM size
- FS ROM size
- FS RAM size

- Bootup Time
- Execution Time

The following pages describe how to measure each item. SzwgMeasurementMethod

7.4 Typical Embedded Boot

7.4.1 Introduction

- Embedded Linux boot procedure is very different from enterprise/pc linux. First of all, in most cases, there is no mass storage devices. Typically, Linux kernel is stored in ROM/Flash on the target system. In order to save the area where Linux kernel is stored, the kernel is compressed as zImage.
- The following is the procedure.
    1. Boot loader copies zImage kernel from ROM/Flash to the specified RAM area and let zImage kernel execute.
    2. zImage kernel (precisely speaking, the routine attached to zImage kernel) decompress the zImage Kernel and load the image on the specified RAM address.
    3. Boot loader let the newly loaded kernel start.
- Depending on the conditions, several different file systems are used. We picked ext2 as typical file system and as normalizer.

7.4.2 Rational

- This method saves ROM size (a little bit) because the kernel is stored compressed.

7.4.3 Specifications

< Recommended implementation for boot loader >

- Boot loader SHALL initialize accessible RAM
    - Boot loader SHALL initialize CPU - cache,MMU,interrupt
    - Boot loader SHALL initialize memory controller,if necessary
    - Boot loader MAY initialize serial controller to get debug message
- Boot loader MAY check Linux kernel stored in ROM/Flash whether it is XIP executable image or zImage
- Boot loader SHALL copy zImage kernel from ROM/Flash to the specified RAM area
- Boot loader SHALL let zImage kernel (in RAM) start.

< Specification for the Kernel >

- zImage Linux kernel SHALL copy the executable Linux image to the specified address.
- zImage Linux kernel SHALL let the executable Linux image start.

7.4.4 Notes(informational and non-normative)

Pros and Cons:

- This method saves Kernel ROM size because the kernel is stored compressed, but require RAM space for Kernel image to run.
- It takes time to make kernel copy(from ROM to RAM) and decompress (RAM to RAM) before kernel boot.
- Kernel execution speed shall be fast as the kernel code is located on RAM. note memory access speed to/from

RAM is faster than ROM/Flash.

- In this configuration, File System is located in ROM/Flash as ext2 file system and it shall be used as Read Only, unless the driver has special implementation.

  Actual data is shown in Appendix (wiki:AppendixA_R2)

7.4.5 References

- This feature is implemented in CELF 1.0 as well as master Linux (2.4 kernel at least)

7.4.6 Remaining Issues

- Some data on the platform are still missing.

7.5 Kernel XIP

7.5.1 Introduction

- This method is for Kernel RAM size reduction.
- XIP(eXecution In Place) is the technique that kernel image is directly executed from ROM/Flash, instead of copying.

7.5.2 Rationale

- This technique saves Kernel RAM usage by not making copy on RAM, while it requires a little bit more ROM because the kernel in ROM is the executable image - not compressed.

7.5.3 Specifications

- Please refer to the BTWG Kernel XIP page:

  KernelXIPSpecification_R2

7.5.4 Notes(informational and non-normative)

Pros and Cons:

- Compared to typical boot(normalizer), RAM usage decreases more than 80% (or 1Mbyte to 1.4Mbyte).
- ROM usage increases about twice (or 600Kbyte to 680Kbyte).
- As positive side effect, the boot up time becomes 30% faster. This is because of the following reason.
  1. XIP avoids copying the executable and read-only data portions, but not the read/write data segments;
  2. Technically, a non-XIP kernel need not be stored compressed, although some amount of additional time would be needed to copy the uncompressed data from ROM to RAM.
- As negative side effect, the execution speed is slow. (note: bigger number shows faster execution in Unix Bench. The number of 10 correspond the performance of SPARC 60MHz). This is because of the memory access speed.
- There may be some limitation on Kernel debug. Usually, debugger replace the program code with "trap instruction" to set breakpoint. This method can be used only when the code is in re-writable memory (RAM). Flash can be re-writable memory but it requires sector by sector write. It would be too slow if the debugger does that way.

  Actual data is shown in Appendix (wiki:AppendixA_R2)

7.5.5 References

XIP is implemented in CELF 1.0.

7.5.6 Remaining Issues

- Some data on the platform are still missing.

7.6 Compress FS(initrd)

7.6.1 Introduction

- This technique can be used for FS ROM size reduction.
- Initrd is a Kernel feature that allows root file system to be in RAM disk. Use this feature and use compress (gzip) feature to save ROM area where original root file system is stored.
- Root file system image is compressed and stored in ROM/Flash. Boot loader (or Kernel) copies the compressed image to RAM and decompress it to RAM (RAM disk). The kernel mount the root file system image as file system.

7.6.2 Rationale

- The original file system image should be stored in ROM/Flash. Instead of saving image as is, compressed (gzip) file system saves the ROM usage.

7.6.3 Specifications

- Kernel SHOULD support initrd to mount root file system in RAM disk.
- Kernel SHOULD support gunzip.

7.6.4 Notes(informational and non-normative)

Pros and Cons:

- ROM size is reduced by 50% - 65% (or 1.36Mbyte to 2.3Mbyte).
- It requires full size of FS image on RAM.
- As side effect, the bootup time becomes slower by 15%.
- The FS can be accessed as Read and Write, however, need to be very careful as initrd itself doesn't have the feature to write it back to Flash.

  Actual data is shown in Appendix (wiki:AppendixA_R2)

7.6.5 References

- This feature is implemented in CELF 1.0 as well as master Linux (2.4 kernel at least)

7.6.6 Remaining Issues

- Some data on the platform are still missing.

7.7 Compress FS(Cramfs)

7.7.1 Introduction

- This method is useful for FS ROM size reduction.

- Cramfs, which is read-only FS, shrinks ROM size of programs while it takes longer loading time of them from ROM into RAM.

7.7.2 Rationale

- Cramfs is compressed file system and it makes FS's ROM size smaller. The file system can be used as Read Only.

7.7.3 Specifications

- Kernel SHOULD support file systems with compression feature (CONFIG_CRAMFS)
- Kernel SHOULD support block device access feature (CONFIG_MTD_BLOCK)

7.7.4 Notes(informational and non-normative)

Pros and Cons:

- There is almost no impact on Kernel ROM size and RAM size.
- The FS ROM size can be reduced by 40%-55%(or 1.1Mbyte-2Mbyte).

  Actual data is shown in Appendix (wiki:AppendixA_R2)

7.7.5 References

- This feature is implemented in CELF 1.0 as well as master Linux (2.4 kernel at least)

7.7.6 Remaining Issues

- Some data on the platform are still missing.

7.8 Compress FS(jffs2)

7.8.1 Introduction

- This method is for FS ROM size reduction.
- jffs2 compresses file systems as well as manages flash memory which has limited lifetime.

7.8.2 Rationale

- jffs2 is compressed file system for flash memory. Use this method to reduce the ROM/Flash usage.

7.8.3 Specifications

- Kernel SHOULD support block device access feature (CONFIG_MTD_BLOCK).
- Kernel SHOULD support Journaled file system and flash memory devices.

7.8.4 Notes(informational and non-normative)

Pros and Cons:

- Kernel ROM size increases about 5% and Kernel RAM size increases about 5%.
- The FS ROM size decreases about 40%-50% (or 1Mbyte - 1.8Mbyte).
- jffs2 can be used as Read and Write file system.
- Flash memory is to be treated carefully to save limited lifetime by means of log structured file system,memory technology device(MTD).

Actual data is shown in Appendix (wiki:AppendixA_R2)

7.8.5 References

- This feature is implemented in CELF 1.0 as well as master Linux (2.4 kernel at least)

7.8.6 Remaining Issues

- Some data on the platform are still missing.

7.9 Work in Progress

7.9.1 Clarification of Linux Size Issue

7.9.1.1 Profiling of CE products

7.9.1.1.1 Cellular phone

- Here is the data of the actual Linux based cellular phone. Unfortunately, the absolute number of the size can not be shared , we can indirectly guess the size using the table below and the table shown in "Introduction" section.

Application	Mobile	10%	17%
	Internet	5%	
	Multimedia	2%	
Middleware	Mobile	26%	47%
	X/XIM/Fonts/Toolkit	17%	
	Multimedia	2%	
	Internet	1%	
Kernel/Userland	Command/glibc	28%	37%
	Driver module	4%	
	Kernel	4%	

- A usage ratio of program(text or code) area in current Linux-based cellular phone (pre-product version) is presented in the table, where `mobile' means programs for mobile cellular phones.
- Since this table is obtained from the pre-product version of the phone, the application layer will become larger.
- The table shows that Linux kernel size is not dominated in the cellular phone usage. The kernel occupies only about 5% of the total program area. The size of the OS/Kernel, which includes the kernel, device drivers, and command glibc, is comparable with one of Middle Layer.

7.9.1.2 Benchmarking with other hi-end RTOS

No data available

7.9.2 Candidate Projects

The followings are the candidate projects which we have not started yet.

1. Application XIP
2. Kernel Message Reduction
3. Kernel Data Configuration
4. Inline to function calls
5. Symbol Weight Bridge
6. Kernel Config Weight
7. uClibC vs glibc
8. Subset definition of Shared lib
9. ARM Thumb, MIPS Snow, etc. Kernel should now support 100% ARM Thumb user space.(http://lists.arm.linux.org.uk/pipermail/linux-arm-kernel/2004-February/019862.html)
10. Compile Option -Os(Size Optimization)

8 Security Specification

This is the specification of Linux security technologies and features, of the Security Working Group of the CE Linux Forum.

8.1 Introduction

The specifications of the Security Working Group deal with making consumer electronics products more secure, both from the consumers' and the manufacturers' point of view. The protections should cover

- Data security,
- Privacy protection,
- System stability, and
- Data integrity.

The purpose of this specification is to define requested or required features of Linux that improve the security of the system for such products. This version of the specification includes only one feature, a protected and persistent RAM file system.

8.2 Rationale

A consumer that buys and uses a CE product expects it to function properly. Additionally there is an implicit assumption that confidential data entered into the product will remain confidential. To this end the specifications help CE Linux attain that goal.

8.3 Terminology

The following is a list of terms used in this specification related to security technology and features.

Asynchronous I/O:
I/O where control is returned to the calling program after the I/O process has started, but before the I/O is completed. The I/O transfer runs in parallel with respect to the processor work. The user program continues executing at the same time the I/O operation is executing.

Blocking I/O:
I/O where control is not returned to the calling program until all requested data is transferred. The I/O transfer runs serially with respect to the processor work.

File system:

The methods and data structures that an operating system uses to keep track of files on a disk or partition; the way the files are organized on the disk. Also used about a partition or disk that is used to store the files or the type of the file system.

Linux Security Modules (LSM):

A framework to support security systems as loadable Linux modules.

Non-blocking I/O:

I/O where control is returned to the calling program after the I/O process has started, but before the I/O is completed. The I/O transfer runs in parallel with respect to the processor work. The user program continues executing at the same time the I/O operation is executing.

Non-volatile storage:

(NVS, persistent storage, memory) A term describing a storage device whose contents are preserved when its power is off. Storage using magnetic media (e.g. magnetic disks, magnetic tape or bubble memory) is normally non-volatile by nature whereas semiconductor memories (static RAM and especially dynamic RAM) are normally volatile but can be made into non-volatile storage by having a (rechargeable) battery permanently connected.

RAM-based file system:

A file system built on RAM as the storage medium.

Stack guarding:

A mechanism for protecting the system from buffer overrun ("stack smashing") attacks.

Synchronous I/O:

I/O where control is not returned to the calling program until all requested data is transferred. The I/O transfer runs serially with respect to the processor work.

8.4 Protected RAM File System Specification

8.4.1 Introduction

This page specifies a file system that will enhance security of system data in the presence of kernel bugs or rogue programs.

8.4.2 Rationale

A single bug in the Linux kernel may cause catastrophic damage to a system. If a product holds irreproducible security keys, financial data, or account information, then loss of such data could render the product unusable, or worse. The customer could suffer financial or legal harm (from account theft or identity theft).

It is not possible to guarantee with certainty that there are no bugs in the Linux kernel. However, it is possible to decrease the probability that a bug in the kernel will cause damage to a particular area of memory or storage. This protected area could then be used with greater confidence to hold sensitive user or product data.

Portions of the product memory and storage should be made resistant to kernel bugs. A protected RAM file system would remain consistent if any of the kernel data pointers are corrupted, or if the kernel starts executing unexpectedly in the wrong location.

8.4.3 Specifications

1. A configuration option for the Linux kernel SHALL be provided which controls whether or not the kernel supports the PRAM file system. This option MUST be called CONFIG_PRAM_FS.
2. A full-featured read/write file system for Linux that is RAM-based.

    2.1 If the memory is non-volatile, the file system SHALL be persistent.

3. File I/O in PRAMFS is always direct, synchronous, and never blocks.
4. PRAMFS should be write-protected.

    4.1 In case there are systems where the write protection is not possible, this feature can be disabled with the CONFIG_PRAMFS_NOWP configuration option.

8.4.4 Notes (informational and non-normative)

There is only minimal effort required to back-port the 2.4.22 version of the PRAMFS patch set to the CELF source tree.

8.4.5 References

Patches are available for PRAMFS against the kernel.org trees for kernel versions 2.4.22 and 2.6.4 at http://pramfs.sourceforge.net.

8.4.6 Remaining Issues

9 Appendix A - System Size Data

9.1 Appendix

Note:

1. The following number shall be used as normalizer. Other method shall be compared with the number.
2. Comparing the number between different platforms doesn't make sense as different platform has different hw configuration. Our intention is to compare between the methods.

9.1.1 Typical Embedded Boot

TI/Omap Platform

Name of Method	ROM for Kernel	RAM for Kernel	ROM for FS	RAM for FS	Boot Time	Execution Speed

	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	actual (ms)	ratio (%)	actual	ratio (%)
Typical BT	557	100	1251	100	2556	100	0	100	2521	100	TBF	100

Note - File System is located in ROM/Flash as ext2 file system.

KMC / SH4 Platform

Name of Method	ROM for Kernel		RAM for Kernel		ROM for FS		RAM for FS		Boot Time		Execution Speed	
	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	actual (ms)	ratio (%)	actual	ratio (%)
Typical BT	687	100	1502	100	2640	100	0	100	9587	100	38.3	100

Note: The end point for the bootup time result is the dummy function celf_sz_boottime_dummy().

Renesas / SH4 Platform

Name of Method	ROM for Kernel		RAM for Kernel		ROM for FS		RAM for FS		Boot Time		Execution Speed	
	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	actual (ms)	ratio (%)	actual	ratio (%)
Typical BT	654	100	1425	100	2644	100	0	100	3995	100	TBF	100

NEC Electronics / VR5500A SOC Platform

Name of Method	ROM for Kernel		RAM for Kernel		ROM for FS		RAM for FS		Boot Time		Execution Speed	
	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	actual (ms)	ratio (%)	actual	ratio (%)
Typical BT	807	100	1637	100	3548	100	0	100	3494	100	TBF	100

Note: File System is located in ROM/Flash as ext2 file system.

9.1.2 Kernel XIP

TI/Omap Platform

Name of Method	ROM for Kernel		RAM for Kernel		ROM for FS		RAM for FS		Boot Time		Execution Speed	
	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	actual (ms)	ratio (%)	actual	ratio (%)
Typical BT	557	100	1251	100	2556	100	0	100	2521	100	TBF	100
XIP	1150	206	207	16	2556	100	0	-	*1)	*1)	TBF	TBF

**1) Can't mount rootfilesytem on FlashROM using XIP kernel.*

KMC / SH4 Platform

Name of Method	ROM for Kernel		RAM for Kernel		ROM for FS		RAM for FS		Boot Time		Execution Speed	
	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	actual (ms)	ratio (%)	actual	ratio (%)
Typical BT	687	100	1502	100	2460	100	0	100	9587	100	38.3	100
XIP	1367	199	277	18	2640	100	0	-	6677	70	5.6	15

Note: The end point for the bootup time result is the dummy function celf_sz_boottime_dummy().

Renesas / SH4 Platform

Name of Method	ROM for Kernel		RAM for Kernel		ROM for FS		RAM for FS		Boot Time		Execution Speed	
	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	actual (ms)	ratio (%)	actual	ratio (%)
Typical BT	654	100	1425	100	2644	100	0	100	3995	100	TBF	100
XIP	1317	201	245	17	2644	100	0	-	2082	52	TBF	TBF

NEC Electronics / VR5500A SOC Platform

Name of Method	ROM for Kernel		RAM for Kernel		ROM for FS		RAM for FS		Boot Time		Execution Speed	
	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	actual (ms)	ratio (%)	actual	ratio (%)
Typical BT	807	100	1637	100	3548	100	0	100	3494	100	TBF	100
XIP	1438	178	271	17	3548	100	0	-	2470	71	TBF	TBF

9.1.3 Compress FS(initrd)

TI/Omap Platform

Name of Method	ROM for Kernel		RAM for Kernel		ROM for FS		RAM for FS		Boot Time		Execution Speed	
	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	actual (ms)	ratio (%)	actual	ratio (%)
Typical BT	557	100	1251	100	2556	100	0	100	2521	100	TBF	100
initrd	565	101	1265	101	1189	46	2556	-	-	-	TBF	TBF

KMC / SH4 Platform

Name of Method	ROM for Kernel		RAM for Kernel		ROM for FS		RAM for FS		Boot Time		Execution Speed	
	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	actual (ms)	ratio (%)	actual	ratio (%)
Typical BT	687	100	1502	100	2640	100	0	100	9587	100	38.3	100
initrd	678	99	1484	99	1276	48	2640	-	10988	115	37.3	97

Note: The end point for the bootup time result is the dummy function celf_sz_boottime_dummy().

Renesas / SH4 Platform

Name of Method	ROM for Kernel		RAM for Kernel		ROM for FS		RAM for FS		Boot Time		Execution Speed	
	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	actual (ms)	ratio (%)	actual	ratio (%)
Typical BT	654	100	1425	100	2644	100	0	100	3995	100	TBF	100
initrd	663	101	1446	101	1276	48	2644	-	-	-	TBF	TBF

NEC Electronics / VR5500A SOC Platform

Name of Method	ROM for Kernel		RAM for Kernel		ROM for FS		RAM for FS		Boot Time		Execution Speed	
	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	actual (ms)	ratio (%)	actual	ratio (%)

Typical BT	807	100	1637	100	3548	100	0	100	3494	100	TBF	100
initrd	816	101	1654	101	1249	35	3548	-	-	-	TBF	TBF

9.1.4 Compress FS(Cramfs)

TI/Omap Platform

Name of Method	ROM for Kernel		RAM for Kernel		ROM for FS		RAM for FS		Boot Time		Execution Speed	
	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	actual (ms)	ratio (%)	actual	ratio (%)
Typical BT	557	100	1251	100	2556	100	0	100	2521	100	TBF	100
cramfs	551	99	1272	102	1380	54	0	-	2513	99.7	TBF	TBF

KMC / SH4 Platform

Name of Method	ROM for Kernel		RAM for Kernel		ROM for FS		RAM for FS		Boot Time		Execution Speed	
	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	actual (ms)	ratio (%)	actual	ratio (%)
Typical BT	687	100	1502	100	2640	100	0	100	9587	100	38.3	100
cramfs	697	101	1554	103	1472	56	0	-	9679	101	40.5	106

Note: The end point for the bootup time result is the dummy function celf_sz_boottime_dummy().

Renesas / SH4 Platform

Name of Method	ROM for Kernel		RAM for Kernel		ROM for FS		RAM for FS		Boot Time		Execution Speed	
	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	actual (ms)	ratio (%)	actual	ratio (%)
Typical BT	654	100	1425	100	2644	100	0	100	TBF	100	TBF	100
cramfs	645	99	1443	101	1507	60	0	-	TBF	TBF	TBF	TBF

NEC Electronics / VR5500A SOC Platform

Name of Method	ROM for Kernel		RAM for Kernel		ROM for FS		RAM for FS		Boot Time		Execution Speed	
	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	actual (ms)	ratio (%)	actual	ratio (%)
Typical BT	807	100	1637	100	3548	100	0	100	3494	100	TBF	100
cramfs	799	99	1653	101	1536	43	0	-	3494	100	TBF	TBF

9.1.5 Compress FS(jffs2)

TI/Omap Platform

Name of Method	ROM for Kernel		RAM for Kernel		ROM for FS		RAM for FS		Boot Time		Execution Speed	
	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	actual (ms)	ratio (%)	actual	ratio (%)
Typical BT	557	100	1251	100	2556	100	0	100	2521	100	TBF	100
jffs2	590	106	1326	106	1516	59	0	-	4831	192	TBF	TBF

**1) Unstable for mounting JFFS2 filesystem.*

KMC / SH4 Platform

Name of Method	ROM for Kernel		RAM for Kernel		ROM for FS		RAM for FS		Boot Time		Execution Speed	
	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	actual (ms)	ratio (%)	actual	ratio (%)
Typical BT	687	100	1502	100	2640	100	0	100	9587	100	38.3	100
jffs2	740	108	1606	107	1600	61	0	-	10350	108	35.6	93

Note: The end point for the bootup time result is the dummy function celf_sz_boottime_dummy().

Renesas / SH4 Platform

Name of Method	ROM for Kernel		RAM for Kernel		ROM for FS		RAM for FS		Boot Time		Execution Speed	
	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	actual (ms)	ratio (%)	actual	ratio (%)

Typical BT	654	100	1425	100	2644	100	0	100	3995	100	TBF	100
jffs2	690	106	1496	105	1644	62	0	-	6069	152	TBF	TBF

NEC Electronics / VR5500A SOC Platform

Name of Method	ROM for Kernel		RAM for Kernel		ROM for FS		RAM for FS		Boot Time		Execution Speed	
	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	Size (Kbyte)	ratio (%)	actual (ms)	ratio (%)	actual	ratio (%)
Typical BT	807	100	1637	100	3548	100	0	100	3494	100	TBF	100
jffs2	844	105	1718	105	1726	48	0	-	5824	167	TBF	TBF