1 Summary

Current deployments of Linux based laptops and netbooks are experiencing numerous failures to resume properly after a suspend or hibernation. Many of these failures manifest themselves as failures of the GUI to restart, leaving a blank screen or a text screen with a lone blinking cursor. The failure has been isolated to the interaction between the kernel suspend logic, the X server, and the Virtual Console driver. The main culprit is the VC driver which Linux inherited from early versions of Intel 386 based UNIX, both USL/SCO and BSD. The problem lies in its API which has numerous race conditions built into its design. These problems have been known for a long time.

This report will try to not go into the long sad history of these problems and only concentrate on how to fix the problem so laptop/netboot Linux platforms can be stable through suspend and resume cycles.

2 Race Conditions Analysis

All would be well if the target platform only had one CPU, a non-SMP kernel without kernel pre-emption and lacked the ability to enter into a power suspend state. In other words, the API seemed like a good idea at the time. It used the facilities available in the UNIX kernels and X server of its day. This meant that process IPC was via signals and device control was limited to `ioctl` calls.

2.1 Kernel Virtual Console

The original intent of the VC driver was to provide multiple sessions on a 386 class PC with only a VGA and keyboard. A user could switch from one "session" to another by a character sequence (ALT+F*n*). This processing takes the following flow:

1. The keyboard state machine decodes the key events, recognizes the ALT+F*n* sequence and calls `set_console()`.
2. The `set_console()` function sets the virtual display number in the global variable `want_console` and schedules a callback tasklet to process it. The switch will take some time and this call is done in this instance at the IRQ level. This is also the first opening for a race condition.
3. At the process level, the `console_callback()` tasklet is called. After checking arguments, it calls `change_console()` which starts the actual switch. It then resets `want_console` to -1, indicating that it has been processed. It also does some display processing like updating the keyboard LEDs, display blanking, and event notifies.
4. In this case, `change_console()` does some error checking and then calls `complete_change_processing()`. It does not return any error status so this call may not happen. This processing is different in the X server case which is described below.
5. The final step is `complete_change_processing()` which actually does the screen change. The change does a fair amount of work sync'ing the state of both consoles, saving

current state, and reloading the video ram with the new display contents. This also sets the `fg_console` global to the new console number and wakes up any processes that might be waiting for the switch to complete.

All this work is done because a keyboard initiated switch is an asynchronous event. It works just fine if it is only responding to keyboard events. The single keyboard serializes events. Where it starts to fall apart is in the `ioctl` interface, in particular, where a process wants to do a switch or wait for a switch to happen.

The sequence in the API starts with a `VT_ACTIVATE`. This `ioctl` does some error checking followed by a call to `set_console()`. The processing continues as for the keyboard event case. This is also where the first race opens up. As noted above, the `want_console` global holds the new console number and it gets reset to -1 "around the time" when the console actually switches. This variable is now exposed to simultaneous activates by other processes at any time after it is set by set_console(). This is not a problem unless a process, usually the one that did the original activate, is also waiting for the switch.

The next call in the sequence is with a `VT_WAITACTIVE`. This `ioctl` places the process on a `waitqueue` waiting for the switch to the console number passed into the syscall. This is typically where the hang occurs. The sequence "activate 3", "activate 4", "wait 3", "wait 4" hangs the process waiting for console 3 until some random time in the future when another process or the keyboard activates console 3 again. A number of bug reports include a note where the user started switching about and getting things moving again.

The problem lies in the two distinct `ioctl` calls, making the API not atomic. The `want_console` variable needs to be protected up to and including the wakeup. However, given the interface, particularly the one described below for the X server, this is not possible without opening even more opportunities for lockups. One approach that I took was to use a *cmpxchg* on `want_console` comparing it to -1. This would cause the `set_console()` to return an error that the caller could then either ignore or use to back off and re-try. Every call except the call by the suspend logic ignores the return value. In fact, the comments indicate that the error is "OK because nobody checks anyway". The proper handling is to set `want_console` to -1 upon exit from the `waitqueue`. This, however, is done by the waiting process which does not exist in the keyboard initiated switch case. It is currently reset in `change_console()` which is too early. Unfortunately, it cannot be delayed to `complete_console_change()` because it may not be called either. At best, it only closes down the window a little bit which is not much comfort on a dual core, hyperthreaded laptop.

> NOTE: the `chvt` command does an activate immediately followed by a waitactive. This is the typical sequence for all of these utilities where there is no retry logic, only fail exits if any of the `ioctl` calls fail.

2.2 Xorg X server

The X server introduces additional race issues. The first case is the way it does the switch. The server cannot have its console display silently ripped out from under itself. The transparent switch is fine for

simple text consoles because all of the display and keyboard state can be preserved in the driver but the X server needs to save its own state first. It configures this by doing a `VT_SETMODE` to enable a set of signal handlers for the notification and switch processing.

The first signal handler is for the case when another process activates another console. The signal handler causes the server to save its graphics state and reset the display hardware to text mode. The server indicates that it has completed its context saving with a `VT_RELDISP` command. This ioctl calls `complete_change_console()` to do the driver level console switch.

The second signal handler handles the reverse case and notifies the server that it the console has been switched back to the server. The server can then return the display to graphics mode and continue on with its display restore. This is equivalent to the `VT_WAITACTIVE` processing.

If the only interaction between the X server and virtual consoles is with keyboard switching or the occasional one-way switch by some utility, the race conditions rarely occur and if they do, the user simply uses a keyboard ALT+F*n* to continue. We do it all the time to switch in and out of X. The problem is magnified when some other subsystem enters into the picture.

The ALT+F*n* processing in the kernel is disabled when the X server is running. The server takes over the keyboard and does its own processing by way of the CTRL+ALT+F*n* sequence. The server itself handles this and releases the display back to the kernel by way of the `VT_RELDISP` call after it saves its state as noted above.

The second race condition is exposed between the time the server's console is activated (and waited for) and the display has been put into graphics mode with a `KD_SETMODE` call. This also applies to the `VT_SETMODE` call to set up the signal handlers and `VT_PROCESS` mode. Activations by other processes are not disabled until the call to set graphics mode leaving a window for other processes to attempt an activate which will cause the X server to exit with an error. In short, the switch and the setting of graphics mode must be either atomic or protected.

2.3 Suspend and Resume

The suspend subsystem must call the *prepare* method in each driver prior to powering down the device. The method is responsible for saving state into memory and doing whatever reset sequences are required to make the device safe for suspend/power-down. Since the X server is effectively a user-mode graphics driver, given its intimate messing about in the graphics device, it too must have a *prepare* method. The method of choice was to select an unused/unallocated virtual console, typically the last one in the set, and activate it. The activation would signal the X server as described above. The X server would then save its state and signal back when it was done with the switch. The suspend driver would save away the console number of the X server for the resume method to find later. When the system resumes, the X server gets its console back and it restores the graphics session.

This seems like a nice, clean way to do the job of suspending and resuming X. There are no changes to X; it simply thinks that the user has switched to a text console. It would work if the interface was not full of races that make it unreliable.

2.4 Race Internals

Every switch starts with a call to `set_console()` and this is the core of the problem. This worked for a kernel based switch based on asynchronous keyboard input but it falls apart with two processes trying to negotiate a switch. One method to fix this would be to fix up error returns so that the instigator of the switch can make sensible re-try attempts but the lowest common denominator is the keyboard and there is no context within the keyboard to contain the re-try. Everything else is constrained by the keyboard. The model is imperative, as in *Go switch!*, and although there is an *I'll wait until you are done*, there is no way for a response of *No* or *Not now*.

> NOTE: There is some error return logic but no one uses it and the wait may be waiting for nothing, aka the hang...

The X server needs a different protocol. It has an extensive amount of state to preserve that only it knows how to save and restore. A vendor specific GPU and display management is a whole lot more complex that the CGA/VGA adapter that the VC driver was built around. This means that the switch has to be in the form of a synchronized *request* → *save state* → *It's yours* sequence followed by another synchronized *grant* → *restore state* → *It's mine* sequence. This is a request model where the requester does not proceed until it gets a positive response and only one transaction can safely happen at a time. On the return side, the handoff back to the server is also synchronous and the releasing process does not proceed until it has a positive response that the server has successfully resumed. This later step resets the gate so that the next switch can proceed. The `VT_PROCESS` switch method comes close to this but the asynchronous keyboard and `VT_ACTIVATE` calls break the synchronization because they can insert a activation anywhere in the sequence.

A request model also protects the period between the `VT_WAITACTIVE` and `KD_SETMODE` because the server (or any other process holding the display) can simply ignore any further requests until after it has completed the low level (re)initialization. This is the second race condition that the current API model cannot fix.

3 API Requirements

The current virtual console API is adequate for a CGA/VGA console. It will even work in an SMP environment because there is always a human who can hit an ALT+F*n* sequence again to make the switch "stick". Extending it to include the X server has also worked for the same reason although it has more opportunity to break. Generally, it works and can be un-stuck by way of the human at the keyboard. It falls apart when extended further to a programmatic switch interface.

3.1 Server API

The server needs the request model to operate properly because it has to do its own state save and restore. Most of the current API comes close but itt cannot have the display ripped out from under it. It does need the following:

1. There must be a notification method to make the switch request to the server. The current API

uses UNIX signals. Signals cannot pass any additional data and, since the server uses `signal`, not the BSD `sigaction`, the can also be unreliable. This would be better if the notification could also send some information about the request, such as who is requesting and what is requested. Realtime signals could be used although the payload is a single integer.

2. The initialization, save, and restore logic must be atomic, or at least un-interruptible. In short, when the server owns the display, it owns the display until it gives it up and it can defer a request for it until it is good and ready to switch.
3. Any change or new API cannot extend beyond the already implemented system interface. There is an architecture and OS specific layer in the server. One part manages the hardware and the other manages the interface to the OS for acquiring and releasing the display. The Linux OS interface has calls to the VC driver to manage switching and signal handlers that pass the events that trigger the signals back into the server.
4. The server needs better error recovery. Any error in this process causes it to exit with an error. This is somewhat drastic, particularly for an error in suspend/resume. I assume that the server takes such drastic error handling because it really does not have many other options.

As an implementation requirement, the server changes should first try the new API and fall back to the current API if it fails. This would keep the server backward compatible with older kernels.

3.2 Kernel API

The kernel API for virtual consoles is broken. Fortunately, this API has limited use isolated to the console drivers, the X server and a small set of console utilities. Some installation tools such as Red Hat's anaconda installer uses virtual terminals although it may not be much more than the debian installer which simply uses the keyboard switched consoles.

1. The virtual console's user interface including the keyboard ALT+F*n* functionality must be preserved. The video console also has terminal emulation, unicode character processing and other tty and graphics capabilities that must be preserved in a running system.
2. There are interfaces to tty layer, frame buffer, and input subsystems. These interfaces must be preserved. Fortunately, the virtual console and its components can be configurable. If the external interfaces to these modules are preserved, a new VC driver would be a drop in replacement and have minimum impact on the rest of the kernel.
3. The `kernel/power/console.c` file is the interface between the suspend/resume subsystem and the console and X server. Its external interfaces must be preserved although its internal interface to the VC driver, currently a call to `set_console()`, can change.
4. There is a small set of console utilities, particularly `chvt` that use the kernel API. Their shell API must be preserved although their internal calls to the VC driver, all `ioctl` calls, may change. As an implementation requirement, the utility changes should first try the new API and fall back to the current API if it fails. This would keep them backward compatible with older kernels.

4 Workarounds

It is clear from the analysis that a kernel fix or workaround that did not change the API in any significant way would not work. Any workable change to the API may as well be the proposed solution in the next section. Viable workarounds are probably of the form error handling changes in the applications. In other words, now that you know how and why the switching API does not work, do not use it or if you are forced to use it, be prepared to take appropriate action if things break. In particular, avoid using the `VT_WAITACTIVE` call unless you also add something like an `alarm` signal to bail the process out of a very long if not interminable sleep.

5 Proposed Solution

My research uncovered an interesting proposal by Jonathan de Boyne Pollard called *Console daemons for Linux*[1]. I cannot find any references to in in Google or Yahoo other than his home page. It seems to be just something he wrote and posted to his personal home page in 2007. There is no indication that he sent it to LKML or anyone reviewed or commented on it. This proposal has some very useful ideas that apply to a proposed solution. His proposal has the kernel of a good idea even though the implementation details are a bit fuzzy. I can't find any actual code from him but that should not stop the exploration of some good ideas.

His proposal identifies two major problems with the virtual console in Linux. The first problem is the inherently racy `ioctl` interface. He does not go into the detail I describe above; he simply makes the common assertion that everyone else already knows.

The bloat in the terminal emulation itself is the second problem. The original emulation was a combination of CGA/VGA character set handling including the “graphics” characters and a VT102 emulation for keyboard and cursor movement. It has grown to have a keyboard handler for Unicode and support for alternate fonts. He notes that if it expands further into Unicode glyph processing, the kernel will be burdened with huge resident font tables in addition to the already large keymap tabless needed for Unicode et al (anything other than VT102 ASCII).

His proposal is to rip out both of these parts of the driver and put them into a console daemon. All of the extended terminal emulation becomes a user mode problem and the keyboard switching gets turned into a synchronization problem between two processes, the X server and the console server daemon. This is the key idea worth exploring.

One other point in his proposal is that most of the code, especially the difficult and detailed emulation code is already there. It is just in the wrong place. The actual console switching code can be refactored into something much smaller and more reliable if there was a usermode daemon for terminal emulation. In essence, get the good stuff where it belongs and clean up what is left.

---

1 There is no other reference other than http://homepages.tesco.net/J.deBoynePollard/Proposals/linux-console-daemon.html. He seems to have spent most of his time on other things like OS/2 and Windows. He also hates SMTP mail, insists on IM2000 email and will only respond to email via Fidonet. He does seem to comment on various blogs and programming websites.

5.1 Kernel Console

The kernel itself needs a basic console for early boot and `printk` output but it really needs little else. The terminal emulation is there for applications once the kernel is up. Everything else just takes up space. The kernel console has the following functions:

1. The driver would be a stripped down VT102 text console with output to the 80x25 CGA video ram. Cursor control would be optional depending on how difficult it would be to strip out of existing code. This terminal emulation would be sufficient to handle a basic shell prompt in single user mode. This is what you would get if you plugged a VT100 into the serial port.
2. The `/dev` devices for the consoles remain. This maintains compatibility with lots of things from `getty` to VGA graphics applications.
3. The new console switching code would manage two things. First, it would mediate the switching between user mode servers and, second, it would disable the basic console when the first server comes up and requests the console.
4. The switching API implements the request model described above. It may use newer Linux event and IPC mechanisms rather than the limited set of `signal` and `ioctl` options from USL UNIX.
5. The kernel console is the first owner of the console. It will release it to the first requesting process by way of the requestor calling the switch API. When a process exits or closes its console file descriptor without first explicitly granting it to another process, the exit/close event will grant the console back to the kernel console.
6. The old VC API would be disabled because it would be incompatible. It would be difficult to simply stub it off with error returns because current users barely use them. The X server simply reports and error and exits currently. The API could generate a deprecated warning to the kernel log and fail return the call.

5.2 Console Daemon

The console daemon is part new code and salvaged parts from `keyboard.c`, `vt.c` and bits of `vt_ioctl.c`. In many respects, we really don't care what is in here. Now that it is user mode code, anyone can hack it to their hearts content so long as the following functions work:

1. The daemon now processes the ALT+F*n* sequence. This change is significant because it now moves the asynchronous keyboard event out of the kernel and into usermode where it can be serialized. The switch to and from the X server is now symmetric. With this, the races disappear.
2. The whole notion of multiple text consoles is now within the daemon. Switching among text consoles is done by the daemon which only has to switch its attention among `/dev/tty`*n* devices. Saved screen buffers etc. are all in user space. The only thing that remains in the kernel is set of console devices.

3. The console devices are now equivalent to pseudo-terminals with one exception. The master side handling must keep the switched out consoles properly alive which is not a requirement on pseudo-terminals which hang up the slave side if the master side exits/closes.
4. The console daemon would start early in the boot sequence, probably right after `init` and `udev`. This would limit the ASCII only support to the early boot of the kernel where all anyone sees are `printk` kernel messages which are ASCII. Most of these are hidden anyway on GUI based systems.

Moving the specialized terminal emulation to user space opens up opportunities for improvement although non of them are any longer within the scope of the kernel. All non-ASCII handling is now in the daemon. This includes support for Unicode glyphs, I18N support and anything else one could think of for a non-english environment. The graphics video would also be available so the daemon could bit blit its own graphics and use higher resolutions than the VGA 80x25 format. The important point is none of these enhancements involve the kernel at all.

Adding a process into the console to application path would be less than the overhead of `sshd` or `telnetd`. This should be acceptable on text only systems because the interrupt rate is limited by the the user's typing rate.

5.3 Server Switching

Console switching in this model becomes symmetric IPC between real processes. One of the problems with the `ioctl` API is that there is no place to keep the context. Moving the switching process to process level keeps the necessary state persistent. The following functions become part of switching:

1. All switching is done among processes. The exception to this in the current VC driver is the ALT+F*n* handling which would be moved to the console daemon.
2. All IPC state between processes is maintained on the user mode and kernel stacks of the two processes involved in the switch. This maintains the context for error handling and recovery.
3. Each process will access the switching through the equivalent of a file descriptor on the console. When a process closes that descriptor, either intentionally or via a process exit, the IPC state will be reset and the console will be switched to another process. Error handling will give sufficient state to the receiving process to allow it to properly recover and resume.
4. There are two types of IPC interchanges to do the switch. The first type is a *request* where one process requests the console from the other. The second is a *grant* where the owner of the console gives control to another waiting process. The three step sequence for each is described above.
5. A system call for this API may wait for the other process to complete the switch but any switch failure will wake up the waiting process which will return an error to the call.

As described above, the kernel console is the first owner of the console and the owner of last resort.

5.3.1 Switching Utilities

The switching utilities, `chvt` in particular, are modified to use the new switching API. They would first request the console to themselves and then immediately grant it to the intended daemon/server.

5.4 Suspend and Resume

Suspend and resume is done with in the kernel at the process level. A suspend event would request the console and then store a reference to process that granted it the console. When the system resumes, the resume method would then grant the console back to that process.

5.5 Implementation Plan

The biggest obstacle to a smooth change is that this proposal spans at least two separate projects. Old interfaces cannot be deprecated overnight and some level of backward compatibility will have to be maintained for some period. Incremental changes would be difficult when an API overhaul is a significant part of the project. The better approach would be to have each component handle both APIs during the transition period. Once the community has moved off of the old API, it can then be deprecated and eventually removed.

This is by no means a detailed implementation plan. It is much too early for that. This plan is but a general, first cut proposal.

The two APIs cannot co-exist at runtime. This is primarily an issue for user mode. The new version can fall back to the old API if it cannot use the new one. This would make them usable in a mixed or old environment. Older versions will break if the interface has already switched over to the new API.

One advantage to this implementation is that the use of the current API is limited to a few subsystems, all of which are in the core of and under the control of the distribution. In fact, the user level documentation cautions against users programming to this interface at all.

5.5.1 Kernel Change

The introduction of the new API will be done in phases. This is a change to a device driver, not a change to the core kernel.

1. The first phase adds the new API to be co-resident with the old design. The first call to either API will disable the use of the other for the lifetime of the kernel boot. Use of the disabled API will immediately return an appropriate error code. The stripped down basic console will not be implemented at this time. The suspend logic will call the appropriate API based on which has been enabled. Adding the new API is a kernel configuration option.
2. The kernel configuration is modifed to make the old interface optional. This phase may be merged into the first phase as far as coding is concerned. At this point, the kernel would no longer respond to the old API, something that cannot happen until the user mode part catches up.

3. This phase cleans up the configuration by removing the dead code. This will simplify the codebase to improve maintainability.

5.5.2 X Server Change

The introduction of the change to the X server parallels the kernel phases.

1. The new API is introduced into the architecture/OS dependent modules of the server. The server will attempt the new API first. If it fails, the server will fall back to the old API for the lifetime of the server. This will ensure backward compatibility with unchanged kernels.
2. Once the old API is deprecated in the kernel, it will be removed from the Linux specific modules of the server.

5.5.3 Console Utilities

The console utilities are small; `chvt` is barely a single page. These utilities will track with the X server. Any scripts or other programs that call these utilities will see no API changes.

6 Open Issues

Moving console switching from an action upon a set of devices to an exchange among cooperating processes opens a few issues:

1. Console numbering becomes an artifact of ALT+F*n*. The current scheme has the x server pick the first console after the kernel initializes set. Since the kernel usually allocates 6 by default, X gets *tty7* although it never uses it as a tty. Suspend/resume uses max – 1 which is 63 for Ubuntu and 99 for Fedora. A console number only makes sense for text consoles because the ALT+F*n* matches the `/dev/tty`*n* opened by proocesses like `getty` and `/dev/console` gets mapped to the “current” console. Suspend/resume shouldn't care about this. All it wants to do is get all the console servers such as X to go to a quiet state. What switches is the process that owns the console subsystem.
2. This does change a long standing API – as bad as it is. This has impact on lots of things beyond just the kernel which is why it has lasted so long. Canonical would have to maintain its own versions until the upstream Xorg and kernel maintainers accept it. We could look at this in the same way we all looked at the *devfs* to *udev* change. Fortunately, this mess is much more localized than *devfs* and we aren't the only ones stuck with this problem. My contact over at Xorg (and Red Hat), Adam Jackson, would be happy to accept a change done by someone else that cleans up this mess.
3. We started off this work hoping for a kernel patch to make suspend work again. Sometimes we don't get what we want. We may have a workaround of sorts but the problem will not go away.
4. One of the pieces missing in this is a method where processes such as the Dell Launcher can find out what the state of the display and/or suspend resume is. I don't know enough details or

the history of why the launcher was doing `VT_WAITACTIVE`. I can assume there was good reason. A better API for finding out such information could be some attributes in sysfs or an interface to `inotify`. The current interface is based on what was available in the pre-Linux world. We have lots of event notification and kernel information interfaces now that would be much better than an `ioctl` from out-of-nowhere and an information starved `signal`.