Reprinted from the

Proceedings of the Linux Symposium

Volume One

July 21th–24th, 2004
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc.*
Stephanie Donovan, *Linux Symposium*
C. Craig Ross, *Linux Symposium*

Review Committee

Jes Sorensen, *Wild Open Source, Inc.*
Matt Domsch, *Dell*
Gerrit Huizenga, *IBM*
Matthew Wilcox, *Hewlett-Packard*
Dirk Hohndel, *Intel*
Val Henson, *Sun Microsystems*
Jamal Hadi Salimi, *Znyx*
Andrew Hutton, *Steamballoon, Inc.*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*

Cooperative Linux

Dan Aloni
da-x@colinux.org

Abstract

In this paper I'll describe Cooperative Linux, a port of the Linux kernel that allows it to run as an unprivileged lightweight virtual machine in kernel mode, on top of another OS kernel. It allows Linux to run under any operating system that supports loading drivers, such as Windows or Linux, after minimal porting efforts. The paper includes the present and future implementation details, its applications, and its comparison with other Linux virtualization methods. Among the technical details I'll present the CPU-complete context switch code, hardware interrupt forwarding, the interface between the host OS and Linux, and the management of the VM's pseudo physical RAM.

1 Introduction

Cooperative Linux utilizes the rather underused concept of a Cooperative Virtual Machine (CVM), in contrast to traditional VMs that are unprivileged and being under the complete control of the host machine.

The term **Cooperative** is used to describe two entities working in parallel, e.g. coroutines [2]. In that sense the most plain description of Cooperative Linux is turning two operating system kernels into two big coroutines. In that mode, each kernel has its own complete CPU context and address space, and each kernel decides when to give control back to its partner.

However, only one of the two kernels has control on the physical hardware, where the other is provided only with virtual hardware abstraction. From this point on in the paper I'll refer to these two kernels as the host operating system, and the guest Linux VM respectively. The host can be every OS kernel that exports basic primitives that provide the Cooperative Linux portable driver to run in CPL0 mode (ring 0) and allocate memory.

The special CPL0 approach in Cooperative Linux makes it significantly different than traditional virtualization solutions such as VMware, plex86, Virtual PC, and other methods such as Xen. All of these approaches work by running the guest OS in a less privileged mode than of the host kernel. This approach allowed for the extensive simplification of Cooperative Linux's design and its short early-beta development cycle which lasted only one month, starting from scratch by modifying the vanilla Linux 2.4.23-pre9 release until reaching to the point where KDE could run.

The only downsides to the CPL0 approach is stability and security. If it's unstable, it has the potential to crash the system. However, measures can be taken, such as cleanly shutting it down on the first internal Oops or panic. Another disadvantage is security. Acquiring root user access on a Cooperative Linux machine can potentially lead to root on the host machine if the attacker loads specially crafted kernel module or uses some very elaborated exploit in case which the Cooperative Linux kernel was compiled without module support.

Most of the changes in the Cooperative Linux patch are on the i386 tree—the only supported architecture for Cooperative at the time of this writing. The other changes are mostly additions of virtual drivers: cobd (block device), conet (network), and cocon (console). Most of the changes in the i386 tree involve the initialization and setup code. It is a goal of the Cooperative Linux kernel design to remain as close as possible to the standalone i386 kernel, so all changes are localized and minimized as much as possible.

2 Uses

Cooperative Linux in its current early state can already provide some of the uses that User Mode Linux[1] provides, such as virtual hosting, kernel development environment, research, and testing of new distributions or buggy software. It also enabled new uses:

- **Relatively effortless migration path from Windows.** In the process of switching to another OS, there is the choice between installing another computer, dual-booting, or using a virtualization software. The first option costs money, the second is tiresome in terms of operation, but the third can be the most quick and easy method—especially if it's free. This is where Cooperative Linux comes in. It is already used in workplaces to convert Windows users to Linux.

- **Adding Windows machines to Linux clusters.** The Cooperative Linux patch is minimal and can be easily combined with others such as the MOSIX or OpenMOSIX patches that add clustering capabilities to the kernel. This work in progress allows to add Windows machines to super-computer clusters, where one illustration could tell about a secretary workstation computer that runs Cooperative Linux as a screen saver—when the secretary goes home at the end of the day and leaves the computer unattended, the office's cluster gets more CPU cycles for free.

- **Running an otherwise-dual-booted Linux system from the other OS.** The Windows port of Cooperative Linux allows it to mount real disk partitions as block devices. Numerous people are using this in order to access, rescue, or just run their Linux system from their ext3 or reiserfs file systems.

- **Using Linux as a Windows firewall on the same machine.** As a likely competitor to other out-of-the-box Windows firewalls, iptables along with a stripped-down Cooperative Linux system can potentially serve as a network firewall.

- **Linux kernel development / debugging / research and study on another operating systems.**

  Digging inside a running Cooperative Linux kernel, you can hardly tell the difference between it and a standalone Linux. All virtual addresses are the same—Oops reports look familiar and the architecture dependent code works in the same manner, excepts some transparent conversions, which are described in the next section in this paper.

- **Development environment for porting to and from Linux.**

3 Design Overview

In this section I'll describe the basic methods behind Cooperative Linux, which include

complete context switches, handling of hardware interrupts by forwarding, physical address translation and the pseudo physical memory RAM.

3.1 Minimum Changes

To illustrate the minimal effect of the Cooperative Linux patch on the source tree, here is a diffstat listing of the patch on Linux 2.4.26 as of May 10, 2004:

```
CREDITS                                |    6
Documentation/devices.txt              |    7
Makefile                               |    8
arch/i386/config.in                    |   30
arch/i386/kernel/Makefile              |    2
arch/i386/kernel/cooperative.c         |  181 +++++
arch/i386/kernel/head.S                |    4
arch/i386/kernel/i387.c                |    8
arch/i386/kernel/i8259.c               |  153 ++++
arch/i386/kernel/ioport.c              |   10
arch/i386/kernel/process.c             |   28
arch/i386/kernel/setup.c               |   61 +
arch/i386/kernel/time.c                |  104 +++
arch/i386/kernel/traps.c               |    9
arch/i386/mm/fault.c                   |    4
arch/i386/mm/init.c                    |   37 +
arch/i386/vmlinux.lds                  |   82 +-
drivers/block/Config.in                |    4
drivers/block/Makefile                 |    1
drivers/block/cobd.c                   |  334 ++++++++++
drivers/block/ll_rw_blk.c              |    2
drivers/char/Makefile                  |    4
drivers/char/colx_keyb.c               | 1221 ++++++++++++*
drivers/char/mem.c                     |    8
drivers/char/vt.c                      |    8
drivers/net/Config.in                  |    4
drivers/net/Makefile                   |    1
drivers/net/conet.c                    |  205 ++++++
drivers/video/Makefile                 |    4
drivers/video/cocon.c                  |  484 ++++++++++++++++
include/asm-i386/cooperative.h         |  175 +++++
include/asm-i386/dma.h                 |    4
include/asm-i386/io.h                  |   27
include/asm-i386/irq.h                 |    6
include/asm-i386/mc146818rtc.h         |    7
include/asm-i386/page.h                |   30
include/asm-i386/pgalloc.h             |    7
include/asm-i386/pgtable-2level.h      |    8
include/asm-i386/pgtable.h             |    7
include/asm-i386/processor.h           |   12
include/asm-i386/system.h              |    8
include/linux/console.h                |    1
include/linux/cooperative.h            |  317 +++++++++
include/linux/major.h                  |    1
init/do_mounts.c                       |    3
init/main.c                            |    9
kernel/Makefile                        |    2
kernel/cooperative.c                   |  254 +++++++
kernel/panic.c                         |    4
kernel/printk.c                        |    6
50 files changed, 3828 insertions(+), 74 deletions(-)
```

3.2 Device Driver

The device driver port of Cooperative Linux is used for accessing kernel mode and using the kernel primitives that are exported by the host OS kernel. Most of the driver is OS-independent code that interfaces with the OS dependent primitives that include page allocations, debug printing, and interfacing with user space.

When a Cooperative Linux VM is created, the driver loads a kernel image from a vmlinux file that was compiled from the patched kernel with `CONFIG_COOPERATIVE`. The vmlinux file doesn't need any cross platform tools in order to be generated, and the same vmlinux file can be used to run a Cooperative Linux VM on several OSes of the same architecture.

The VM is associated with a per-process resource—a file descriptor in Linux, or a device handle in Windows. The purpose of this association makes sense: if the process running the VM ends abnormally in any way, all resources are cleaned up automatically from a callback when the system frees the per-process resource.

3.3 Pseudo Physical RAM

In Cooperative Linux, we had to work around the Linux MM design assumption that the entire physical RAM is bestowed upon the kernel on startup, and instead, only give Cooperative Linux a fixed set of physical pages, and then only do the translations needed for it to work transparently in that set. All the memory which Cooperative Linux considers as physical is in that allocated set, which we call the Pseudo Physical RAM.

The memory is allocated in the host OS using the appropriate kernel function—`alloc_pages()` in Linux and `MmAllocatePagesForMdl()` in Windows—so it is not mapped in any address space on the host for not wasting PTEs. The allocated pages are always resident and not freed until the VM is downed. Page tables

```
--- linux/include/asm-i386/pgtable-2level.h     2004-04-20 08:04:01.000000000 +0300
+++ linux/include/asm-i386/pgtable-2level.h     2004-05-09 16:54:09.000000000 +0300
@@ -58,8 +58,14 @@
 }
 #define ptep_get_and_clear(xp) __pte(xchg(&(xp)->pte_low, 0))
 #define pte_same(a, b)         ((a).pte_low == (b).pte_low)
-#define pte_page(x)            (mem_map+((unsigned long)(((x).pte_low >> PAGE_SHIFT))))
 #define pte_none(x)            (!(x).pte_low)
+
+#ifndef CONFIG_COOPERATIVE
+#define pte_page(x)            (mem_map+((unsigned long)(((x).pte_low >> PAGE_SHIFT))))
 #define __mk_pte(page_nr,pgprot) __pte(((page_nr) << PAGE_SHIFT) | pgprot_val(pgprot))
+#else
+#define pte_page(x)            CO_VA_PAGE((x).pte_low)
+#define __mk_pte(page_nr,pgprot) __pte((CO_PA(page_nr) & PAGE_MASK) | pgprot_val(pgprot))
+#endif

 #endif /* _I386_PGTABLE_2LEVEL_H */
```

Table 1: Example of MM architecture dependent changes

are created for mapping the allocated pages in the VM's kernel virtual address space. The VM's address space resembles the address space of a regular kernel—the normal RAM zone is mapped contiguously at 0xc0000000.

The VM address space also has its own special fixmaps—the page tables themselves are mapped at 0xfef00000 in order to provide an O(1) ability for translating PPRAM (Psuedo-Physical RAM) addresses to physical addresses when creating PTEs for user space and `vmalloc()` space. On the other way around, a special physical-to-PPRAM map is allocated and mapped at 0xff000000, to speed up handling of events such as pages faults which require translation of physical addresses to PPRAM address. This bi-directional memory address mapping allows for a negligible overhead in page faults and user space mapping operations.

Very few changes in the i386 MMU macros were needed to facilitate the PPRAM. An example is shown in Table 1. Around an #ifdef of `CONFIG_COOPERATIVE` the `__mk_pte()` low level MM macro translates a PPRAM struct page to a PTE that maps the real physical page. Respectively, `pte_page()` takes a PTE that was generated by `__mk_pte()` and returns the corresponding struct page for it. Other macros such as `pmd_page()` and `load_cr3()` were also changed.

3.4 Context Switching

The Cooperative Linux VM uses only one host OS process in order to provide a context for itself and its processes. That one process, named colinux-daemon, can be called a Super Process since it frequently calls the kernel driver to perform a context switch from the host kernel to the guest Linux kernel and back. With the frequent (HZ times a second) host kernel entries, it is able able to completely control the CPU and MMU without affecting anything else in the host OS kernel.

On the Intel 386 architecture, a complete context switch requires that the top page directory table pointer register—CR3—is changed. However, it is not possible to easily change both the instruction pointer (EIP) and CR3 in one instruction, so it implies that the code that changes CR3 must be mapped in both contexts for the change to be possible. It's problematic to map that code at the same virtual address in both contexts due to design limitations—the two contexts can divide the kernel and user ad-

dress space differently, such that one virtual address can contain a kernel mapped page in one OS and a user mapped page in another.

In Cooperative Linux the problem was solved by using an intermediate address space during the switch (referred to as the 'passage page,' see Figure 1). The intermediate address space is defined by a specially created page tables in both the guest and host contexts and maps the same code that is used for the switch (passage code) at both of the virtual addresses that are involved. When a switch occurs, first CR3 is changed to point to the intermediate address space. Then, EIP is relocated to the other mapping of the passage code using a jump. Finally, CR3 is changed to point to the top page table directory of the other OS.

The single MMU page that contains the passage page code, also contains the saved state of one OS while the other is executing. Upon the beginning of a switch, interrupts are turned off, and a current state is saved to the passage page by the passage page code. The state includes all the general purpose registers, the segment registers, the interrupt descriptor table register (IDTR), the global descriptor table (GDTR), the local descriptor register (LTR), the task register (TR), and the state of the FPU / MMX / SSE registers. In the middle of the passage page code, it restores the state of the other OS and interrupts are turned back on. This process is akin to a "normal" process to process context switch.

Since control is returned to the host OS on every hardware interrupt (described in the following section), it is the responsibility of the host OS scheduler to give time slices to the Cooperative Linux VM just as if it was a regular process.

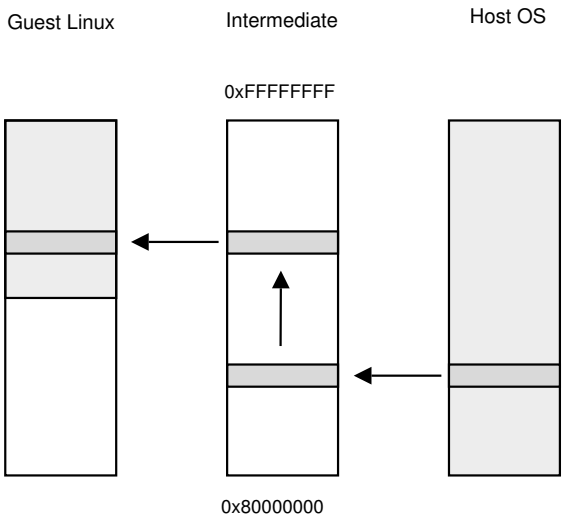

Figure 1: Address space transition during an OS cooperative kernel switch, using an inter-mapped page

3.5 Interrupt Handling and Forwarding

Since a complete MMU context switch also involves the IDTR, Cooperative Linux must set an interrupt vector table in order to handle the hardware interrupts that occur in the system during its running state. However, Cooperative Linux only forwards the invocations of interrupts to the host OS, because the latter needs to know about these interrupts in order to keep functioning and support the colinux-daemon process itself, regardless to the fact that external hardware interrupts are meaningless to the Cooperative Linux virtual machine.

The interrupt vectors for the internal processor exceptions (0x0–0x1f) and the system call vector (0x80) are kept like they are so that Cooperative Linux handles its own page faults and other exceptions, but the other interrupt vectors point to special proxy ISRs (interrupt service routines). When such an ISR is invoked during the Cooperative Linux context by an external hardware interrupt, a context switch is made to the host OS using the passage code. On the

other side, the address of the relevant ISR of the host OS is determined by looking at its IDT. An interrupt call stack is forged and a jump occurs to that address. Between the invocation of the ISR in the Linux side and the handling of the interrupt in the host side, the interrupt flag is disabled.

The operation adds a tiny latency to interrupt handling in the host OS, but it is quite neglectable. Considering that this interrupt forwarding technique also involves the hardware timer interrupt, the host OS cannot detect that its CR3 was hijacked for a moment and therefore no exceptions in the host side would occur as a result of the context switch.

To provide interrupts for the virtual device drivers of the guest Linux, the changes in the arch code include a virtual interrupt controller which receives messages from the host OS on the occasion of a switch and invokes `do_IRQ()` with a forged `struct pt_args`. The interrupt numbers are virtual and allocated on a per-device basis.

4 Benchmarks And Performance

4.1 Dbench results

This section shows a comparison between User Mode Linux and Cooperative Linux. The machine which the following results were generated on is a 2.8GHz Pentium 4 with HT enabled, 512GB RAM, and a 120GB SATA Maxtor hard-drive that hosts ext3 partitions. The comparison was performed using the dbench 1.3-2 package of Debian on all setups.

The host machine runs the Linux 2.6.6 kernel patched with SKAS support. The UML kernel is Linux 2.6.4 that runs with 32MB of RAM, and is configured to use SKAS mode. The Cooperative Linux kernel is a Linux 2.4.26 kernel and it is configured to run with 32MB of RAM, same as the UML system. The root file-system of both UML and Cooperative Linux machines is the same host Linux file that contains an ext3 image of a 0.5GB minimized Debian system.

The commands ‘dbench 1’, ‘dbench 3’, and ‘dbench 10’ were run in 3 consecutive runs for each command, on the host Linux, on UML, and on Cooperative Linux setups. The results are shown in Table 2, Table 3, and Table 4.

System	Throughput	Netbench
Host	43.813	54.766
	50.117	62.647
	44.128	55.160
UML	10.418	13.022
	9.408	11.760
	9.309	11.636
coLinux	10.418	13.023
	12.574	15.718
	12.075	15.094

Table 2: output of dbench 10 (units are in MB/sec)

System	Throughput	Netbench
Host	43.287	54.109
	41.383	51.729
	59.965	74.956
UML	11.857	14.821
	15.143	18.929
	14.602	18.252
coLinux	24.095	30.119
	32.527	40.659
	36.423	45.528

Table 3: output of dbench 3 (units are in MB/sec)

4.2 Understanding the results

From the results in these runs, ‘dbench 10’, ‘dbench 3’, and ‘dbench 1’ show 20%, 123%, and 303% increase respectively, compared to UML. These numbers relate to the number

System	Throughput	Netbench
Host	158.205	197.756
	182.191	227.739
	179.047	223.809
UML	15.351	19.189
	16.691	20.864
	16.180	20.226
coLinux	45.592	56.990
	72.452	90.565
	106.952	133.691

Table 4: output of dbench 1 (units are in MB/sec)

of dbench threads, which is a result of the synchronous implementation of cobd[1]. Yet, neglecting the versions of the kernels compared, Cooperative Linux achieves much better probably because of low overhead with regard to context switching and page faulting in the guest Linux VM.

The current implementation of the cobd driver is synchronous file reading and writing directly from the kernel of the host Linux—No user space of the host Linux is involved, therefore less context switching and copying. About copying, the specific implementation of cobd in the host Linux side benefits from the fact that `filp->f_op->read()` is called directly on the cobd driver's request buffer after mapping it using `kmap()`. Reimplementing this driver as asynchronous on both the host and guest—can improve performance.

Unlike UML, Cooperative Linux can benefit in the terms of performance from the implementation of kernel-to-kernel driver bridges such as cobd. For example, currently virtual Ethernet in Cooperative Linux is done similar to UML—i.e., using user space daemons with tuntap on the host. If instead we create a kernel-to-kernel implementation with no user space daemons in between, Cooperative Linux has the potential to achieve much better in benchmarking.

[1] ubd UML equivalent

5 Planned Features

Since Cooperative Linux is a new project (2004–), most of its features are still waiting to be implemented.

5.1 Suspension

Software-suspending Linux is a challenge on standalone Linux systems, considering the entire state of the hardware needs to be saved and restored, along with the space that needs to be found for storing the suspended image. On User Mode Linux suspending [3] is easier—only the state of a few processes needs saving, and no hardware is involved.

However, in Cooperative Linux, it will be even easier to implement suspension, because it will involve its internal state almost entirely. The procedure will involve serializing the pseudo physical RAM by enumerating all the page table entries that are used in Cooperative Linux, either by itself (for user space and vmalloc page tables) or for itself (the page tables of the pseudo physical RAM), and change them to contain the pseudo value instead of the real value.

The purpose of this suspension procedure is to allow no notion of the real physical memory to be contained in any of the pages allocated for the Cooperative Linux VM, since Cooperative Linux will be given a different set of pages when it will resume at a later time. At the suspended state, the pages can be saved to a file and the VM could be resumed later. Resuming will involve loading that file, allocating the memory, and fix-enumerate all the page tables again so that the values in the page table entries point to the newly allocated memory.

Another implementation strategy will be to just dump everything on suspension as it is, but on resume—enumerate all the page table entries and adjust between the values of the old RPPFNs[2] and new RPPFNs.

Note that a suspended image could be created under one host OS and be resumed in another host OS of the same architecture. One could carry a suspended Linux on a USB memory device and resume/suspend it on almost any computer.

5.2 User Mode Linux[1] inside Cooperative Linux

The possibility of running UML inside Cooperative Linux is not far from being immediately possible. It will allow to bring UML with all its glory to operating systems that cannot support it otherwise because of their user space APIs. Combining UML and Cooperative Linux cancels the security downside that running Cooperative Linux could incur.

5.3 Live Cooperative Distributions

Live-CD distributions like KNOPPIX can be used to boot on top of another operating system and not only as standalone, reaching a larger sector of computer users considering the host operating system to be Windows NT/2000/XP.

5.4 Integration with ReactOS

ReactOS, the free Windows NT clone, will be incorporating Cooperative Linux as a POSIX subsystem.

5.5 Miscellaneous

- Virtual frame buffer support.
- Incorporating features from User Mode Linux, e.g. humfs [3].
- Support for more host operating systems such as FreeBSD.

6 Conclusions

We have discussed how Cooperative Linux works and its benefits—apart from being a BSKH[4], Cooperative Linux has the potential to become an alternative to User Mode Linux that enhances on portability and performance, rather than on security.

Moreover, the implications that Cooperative Linux has on what is the media defines as ‘Linux on the Desktop’—are massive, as the world’s most dominant albeit proprietary desktop OS supports running Linux distributions for free, as another software, with the aimed-for possibility that the Linux newbie would switch to the standalone Linux. As user-friendliness of the Windows port will improve, the exposure that Linux gets by the average computer user can increase tremendously.

7 Thanks

Muli Ben Yehuda, IBM

Jun Okajima, Digital Infra

Kuniyasu Suzaki, AIST

References

[1] Jeff Dike. User Mode Linux. `http://user-mode-linux.sf.net`.

[2] real physical page frame numbers

[3] A recent addition to UML that provides an host FS implementation that uses files in order to store its VFS metadata

[4] Big Scary Kernel Hack

[2] Donald E. Knuth. *The Art of Computer Programming*, volume 1. Addison-Wesley, Reading, Massachusetts, 1997. Describes coroutines in their pure sense.

[3] Richard Potter. Scrapbook for User Mode Linux. `http://sbuml.sourceforge.net/`.