термоядерная отладка в Linux и xBSD
обзор отладчиков ядерного уровня

крис касперски, ака мыщъх, a.k.a. nezumi, a.k.a. souriz, a.k.a. elraton, no-email

отладчиков уровня ядра под никсы — много, хороших из них мало (если такие вообще есть) и нужно быть нереально крутым хакером, чтобы с первого напаса выбрать такой дебагер, которым можно отлаживать, а не обламываться. перепробовав кучу отладчиков, мыщъх решил составить внятный обзор для начинающих, рассказывающий чем один отладчик отличается от другого и какой из них торкает, а какой никуда не канает кроме как в /dev/nul

введение

Существует множество интегрированных отладчиков для Linux'а, но ни один из них не включен в основную ветвь ядра, что выглядит странно, если не сказать подозрительно, особенно если учесть, что xBSD-системы включают в себя ядерный отладчик изначально (правда, во всех известных мне дистрибутивах он по умолчанию задисаблен и его активация требует перекомпиляции ядра).

Причина в том, что Линус Торвальдс (до сих пор стоящий у руля и принимающий решение о включении тех или иных компонентов в ядро) не доверяет интерактивным отладчикам и считает, что у "правильных" программистов таких потребностей просто не возникает. Типа, есть же отладочная печати (см . man printk) и ее, типа, вполне достаточно.

С Линусом, однако, согласны далеко не все разработчики и в определенных ситуациях без дебагера не обойтись, особенно если приходится отлаживать чужие модули, поставляемые без исходных текстов или ломать защиты, противостоящие отладчикам прикладного уровня. Так что, хочет того Торвальдс или нет, но "термоядерные" отладчики для Линуха все-таки есть, причем в количестве намного большим одного, причем, практически все они распространяется в исходных текстах и не требует денег. Казалось бы какая проблема — скачал, поставил, запустил….

Между тем, проблемы все-таки есть. Это и плохая совместимость неофициальных отладчиков в различными версиями официальных ядер и сложность выбора хорошего отладчика из кучи заброшенных проектов… Ситуация усугубляется тем, что различные отладчики предназначены для решения различных задач и потому на вопрос: "какой ядерный отладчик самый лучший" даются сильно неодинаковые ответы, зачастую без всяких пояснений! Ну и какая польза от таких советов?!

типы отладчиков

Классические ядерные отладчики (как для UNIX, так и для Windows) требуют наличия _двух_ компьютеров, на одном из которых устанавливается отлаживаемое ядро, а на другом — сам отладчик. Обмен данными обычно осуществляется по последовательному порту через нуль-модем, хотя можно встретить и другие варианты. Такие отладчики называются **удаленными** и к ним, в частности, относится знаменитый gdb (клиентская часть). Другая часть отладчика находится непосредственно в ядре и если там ее нет (а Linux'е ее нет), у нас ничего не получится. Очевидный недостаток удаленных отладчиков — необходимость приобретения второго компьютера, что далеко не всегда приемлемо (особенно для "домашних" хакеров). Виртуальные машины в какой-то мере снижают остроту проблемы, но…

Локальные отладчики выгодно отличаются тем, что позволяют отлаживать ядро на одной машине с отладчиком. Чаще всего они работают только в текстовом режиме. Поддержка консоли в графическом режиме (не говоря уже про X'ы) требует специальных "агентов", работающих далеко не везде и не всегда, а потому в ряде случаев приходится прибегать к удаленной отладке.

Конструктивно отладчики могут быть реализованы либо как неофициальный патч ядра (требующий его перекомпиляции), либо как драйвер, загружаемый в ядро налету зачастую даже без перезагрузки системы. Примером отладчиков первого типа служит KDB, второго — LinIce.

Однако, независимо от особенностей своей реализации, все Linux-отладчики страдают хронической проблемой совместимости с ядрами, поскольку, разработчики ядра не координируют свои действия с разработчиками отладчиков и потому последние поддерживают

только фиксированный набор версий ядер, причем, поддержка новых версий как правило происходит с большим опозданием.

У xBSD таких проблем вообще по жизни нет, поскольку ядерный отладчик разрабатывается (и поставляется) вместе с сами ядром, и нормально поддерживает все графические режимы, в которых только работает сама xBSD.

использование отладочных возможностей VMWare

Начиная с версии 6.0 RC2 в популярной виртуальной машине VM Ware появился механизм Record/Replay, позволяющий (среди прочих возможностей) осуществлять удаленную ядерную отладку даже для тех операционных систем, которые поставляются без интегрированного отладчика: Linux, xBSD с выключенным отладчиком, NT, etc.

Просто добавляем в vmx-файл (описывающий конфигурацию виртуальной машины) строку "debugStub.listen.guest32=1" (или "debugStub.listen.guest64=1" для 64-разрядных платформ), после чего в vmware.log файле появляется следующая запись "VMware Workstation is listening for debug connection on port 8832", означающая, что виртуальная машина слушает 8832-порт с которым готова общаться по gdb-протоколу. Остается запустить сам gdb, приконнектится к порту и приступить к отладке безо всяких танцев с бубном, без наложения заплаток, без перекомпиляции ядра, etc.

Отладчик gdb может быть запущен как на хосте (основной операционной системе), так и на соседней виртуальной машине.

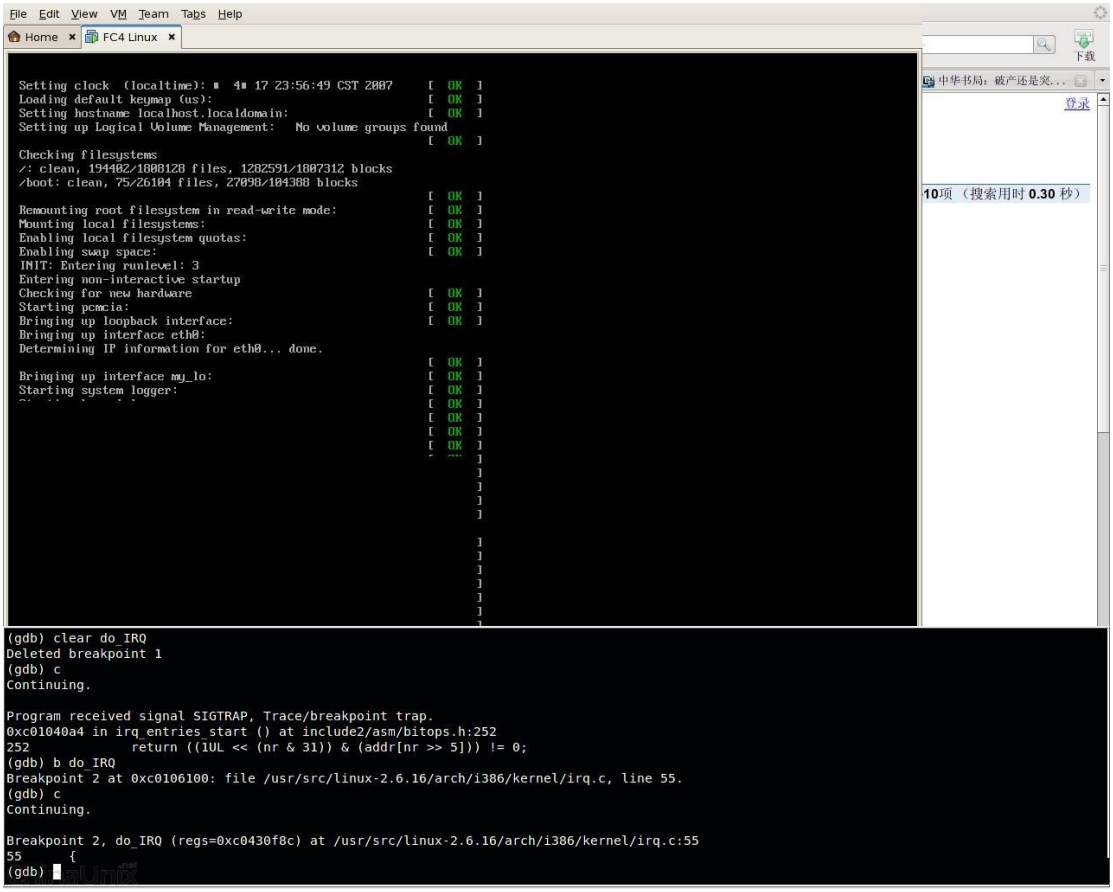

Рисунок 1 использование отладочных возможностей VM Ware 6.0 для отладки Linux'а без интегрированного отладчика

Для отладки нам потребуется _два_ комплекта ядер. Одно – установленное на отлаживаемой системе и другое – установленное на машине (реальной или виртуальной) где работает gdb. Отлаживаемое ядро может быть скомпрессированно и постripано (как обычно и бывает), а вот ядро для gdb в обязательном порядке должно быть разжато (ну не понимает gdb сжатых ядер — что тут поделаешь!) и желательно откомпилировано с отладочной

информацией. Как минимум, на машине с gdb должен присутствовать файл System.map. Естественно, версии обоих ядер должны совпадать, иначе начнется полный хаос.

Пара примеров работы с gdb представлена ниже:

```
# запускаем gdb
% gdb

# указываем путь к нескопрессированному 32-битному ядру
(gdb) file vmlinux-2.4.69-27.EL.debug

# коннектимся к отлаживаемому ядру
(gdb) target remote localhost:8832

# все! с этого момента можно начинать
# отладку ядра!!!
```

Листинг 1 отладка x86-ядра Linux'а под VM Ware

```
# запускаем gdb
% gdb

# указываем путь к нескопрессированному 64-битному ядру
(gdb) file vmlinux-2.6.96-17.EL.debug

# переводим gdb в 64-разрядный режим
(gdb) set architecture i386:x86-64

# коннектимся к отлаживаемому ядру
(gdb) target remote localhost:8832

# все! с этого момента можно начинать
# отладку ядра!!!
```

Листинг 2 отладка x86-64 ядра Linux'а под VM Ware

Естественно, ядро запущенное на эмуляторе, "видит" только виртуальное железо (исключение составляют USB-устройства, жесткие диски и сетевые карты) к которым VM Ware позволяет давать прямой доступ, однако, например, отладить драйвер видео-карты таким образом уже не получится.

Подробнее на эту тему можно почитать: http://stackframe.blogspot.com/ и http://blogs.vmware.com/sherrod/2007/04/index.html, а триальную версию VMWare скачать — http://www.vmware.com/download/ws

использование отладочных возможностей QEMU

Эмулятор QEMU так же позволяет отлаживать ядра без интегрированных отладчиков, но, в отличии от VM Ware, он бесплатен и распространяется вместе с исходными тестами, которые находятся на http://fabrice.bellard.free.fr/qemu/

Пример командной строки, реализующий форсированную отладку, приведен ниже:

```
# запускам QEMU с ядром, которое мы собираемся отлаживать
$ qemu -kernel /boot/bzImg -append "root=/dev/hda" -std-vga -m 256m -s -hda hdd.img &

# запускам gdb на основной машине и коннектимся на порт 1234
$ gdb (gdb) target remote localhost:1234

# подключаем образ ядра (должен совпадать с отлаживаемым ядром)
(gdb) file vmlinux
```

Листинг 3 отладка Linux'а без интегрированного отладчика под QEMU

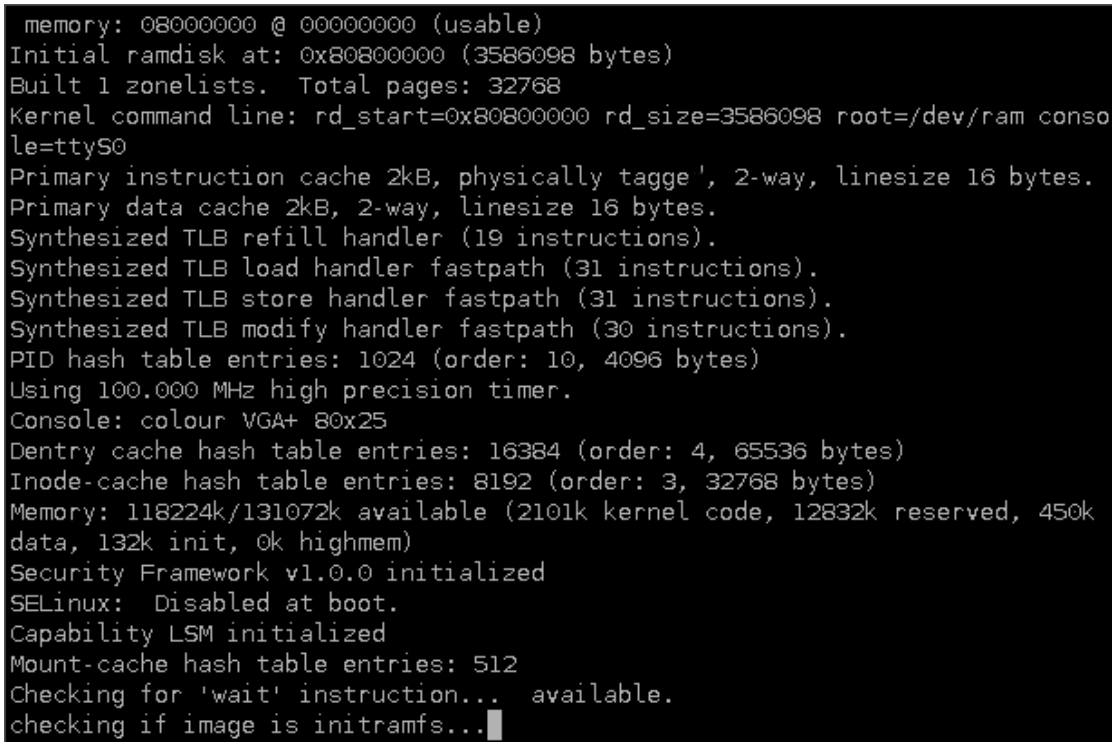

Рисунок 2 загрузка Linux-ядра на виртуальной машине бесплатного эмулятора QEMU

Novell Linux Kernel Debugger (NLKD)

На сегодняшний день, NLKD является, пожалуй, самым продвинутым и мощным ядерным отладчиком для Linux, поддерживающим как локальную, так и удаленную отладку. Другие его достоинства — бесплатность и наличие исходных текстов. К сожалению, он работает _только_ с **SUSE Linux Enterprise Server v9 SP1/SP2** и требует перекомпиляции ядра, что является существенным недостатком, ограничивающим область его применения. Зато NLKD имеет документированный расширяемый интерфейс плагинов и свободно работает в как в текстовом, так и в графическом режиме.

Скачать последнюю версию отладчика (вместе с документацией) можно по ссылке: http://forge.novell.com/modules/xfmod/project/?nlkd

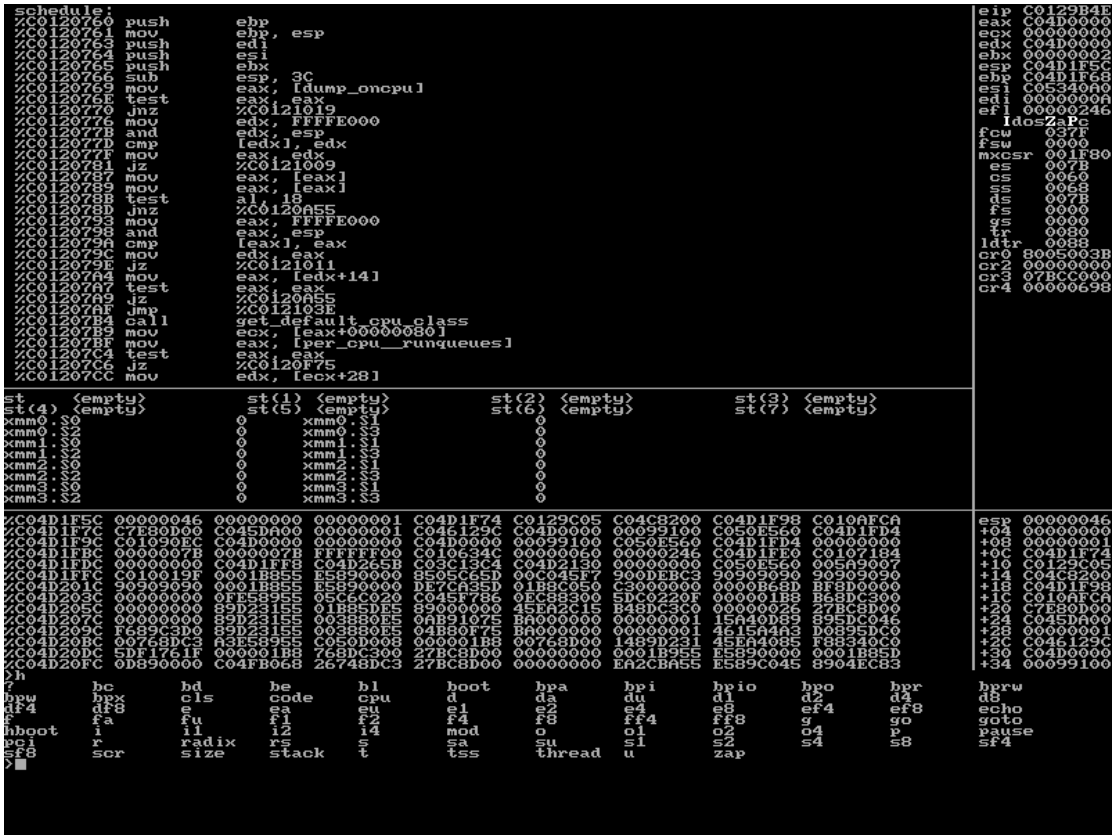

Рисунок 3 внешний вид отладчика NLKD

отладчик KDB

KDB – самый популярный ядерный отладчик для Linux-систем, разработанный компанией SGI, распространяющий его в исходных текстах на бесплатной основе как побочный продукт своей деятельности — http://oss.sgi.com/projects/kdb/

Отладчик реализован как патч к ядру версий 2.[234], устанавливаемый следующим образом:

```
$ cd linux
$ patch -p1 < kdb-xxx
$ make *config
```

Листинг 4 установка отладчика KDB путем патчка исходных текстов ядра

Активируем флаги CONFIG_KDB и CONFIG_FRAME_POINTER (что можно сделать при помощи утилит типа xconfig, menuconfig, oldconfig или ручного задания опций make-файла), перекомпилируем ядро и радуемся жизни.

KDB поддерживает локальный/удаленный режимы отладки и работает на процессорах семейства x86 и IA64, однако, в режиме локальной отладки имеет большие проблемы с различными графическими режимами и некоторыми контроллерами клавиатурами, что не есть хорошо, поэтому использование NLKD в ряде случаев оказывается все же предпочтительнее.

Переход в текстовой режим осуществляется по ALT-CTRL-F1, а возвращение — ALT-CTRL-F7. Клавиша <Pause> вызывает всплытие локальной консоли отладчика (аналог <CTL-D> в soft-ice), позволяя нам просматривать память, устанавливать точки останова, дизассемблировать машинный код и т.д. Короче, практически все как в soft-ice, вот только отладки на уровне исходных текстов KDB не представляет, что является существенным недостатком для обычных разработчиков, но нас, хакеров, совершенно не волнует, поскольку, если бы у нас были исходные тексты, разве стали бы мы заниматься онанизмом и часами дрочиить в отладчике?!

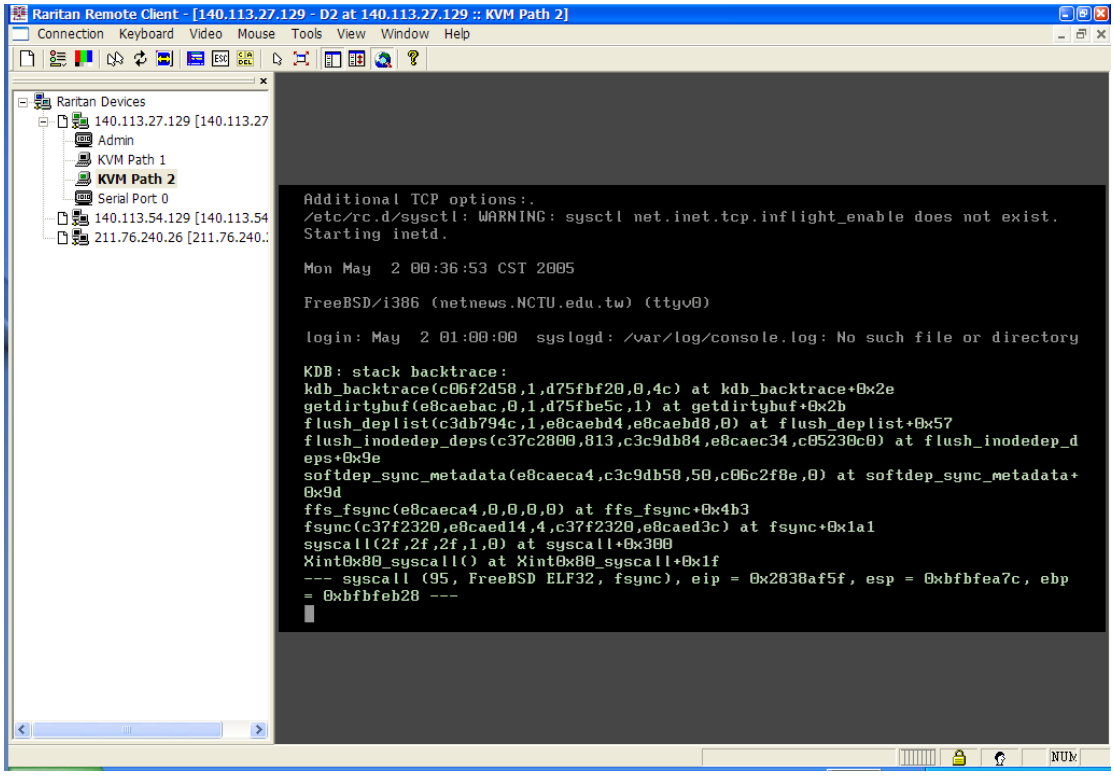

Рисунок 4 сеанс удаленной отладки в KDB

отладчик KGDB

Читая обзоры, можно подумать, что KGDB – это конкурент (или, если не конкурент, то альтернатива) KDB. На самом деле это не так и KGDB представляет собой интегрированный отладчик удаленного (не локального!!!) типа, поддерживающий несколько версий ядер от 2.4.6 до 2.6.0 и реализованный на платформах i386, x86_64, PPC и S390.

KGDB устанавливается путем патча ядра с последующей перекомпиляцией и реализует только серверную часть. В качестве клиента обычно используется gdb или другой отладчик, поддерживающий его нуль-модемный протокол.

Если на удаленной машине (там, где находится gdb) положить ядро, откомпилированное с отладочной информацией (ключ -g), мы получим отладку на уровне исходных текстов, обеспечиваемую средствами gdb, а отнюдь не KGDB, как утверждают некоторые руководства.

Скачать последнюю версию отладчика можно с http://kgdb.linsyssoft.com/

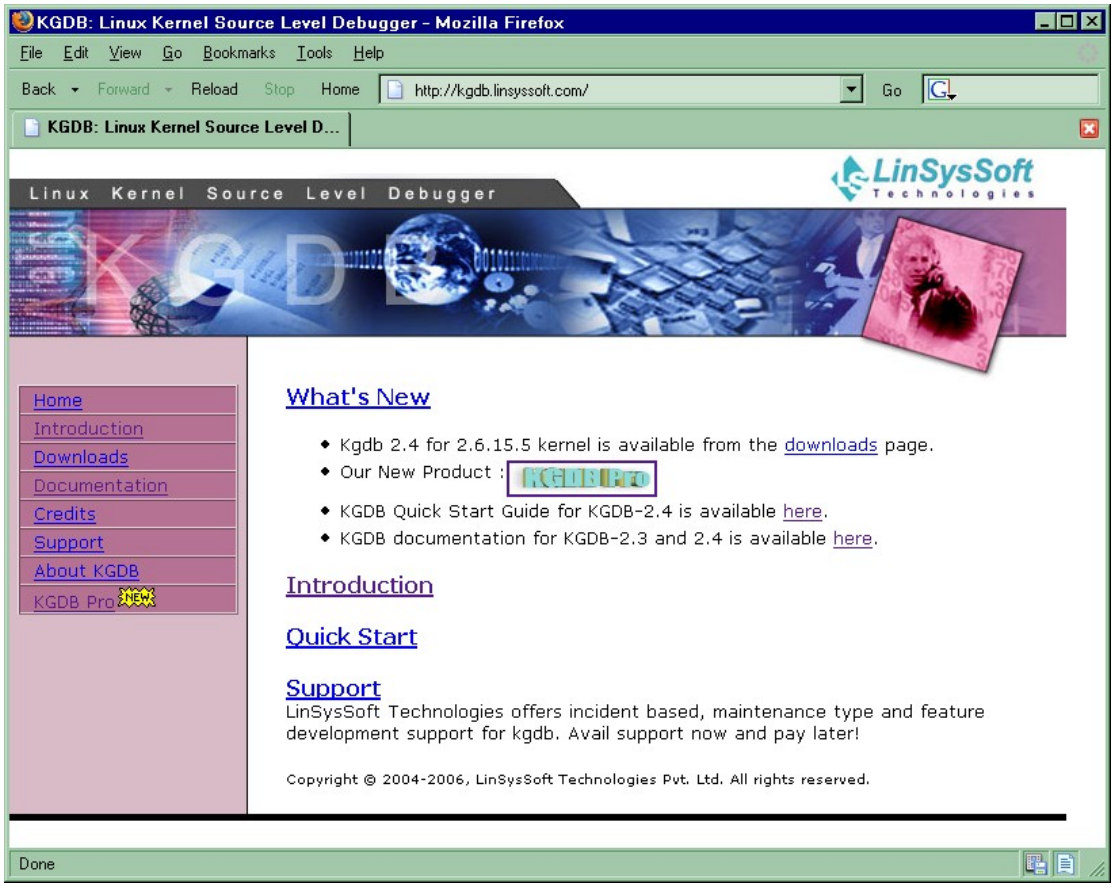

Рисунок 5 здесь можно надыбать KGDB

отладчик LinIce

LinIce представляет собой некоторую пародию на soft-ice под Linux, конструктивно реализованную как загружаемый модуль ядра, не требующую ни наличия второй машины, ни перекомпиляции, что дает ему сто очков вперед. К сожалению, у LinIce имеется множество проблем. Ядра 2.6.9 и выше не поддерживаются, как не поддерживаются Super-VGA и frame-buffer режимы. К тому же имеется множество проблем с различными клавиатурными контроллерами…

Тем не менее, LinIce вполне пригоден для хакерства, особенно если необходимо что-то быстро отломать, а времени/желания/возможности перекомпиляции ядра у нас нет. Однако, следует помнить, что по своим функциональным возможностям LinIce самый бедный отладчик, из всех рассмотренных выше и потому для серьезного хакинга все-таки лучше поставить NLKD или KDB.

Последнюю версию отладчика можно бесплатно скачать с http://www.linice.com.

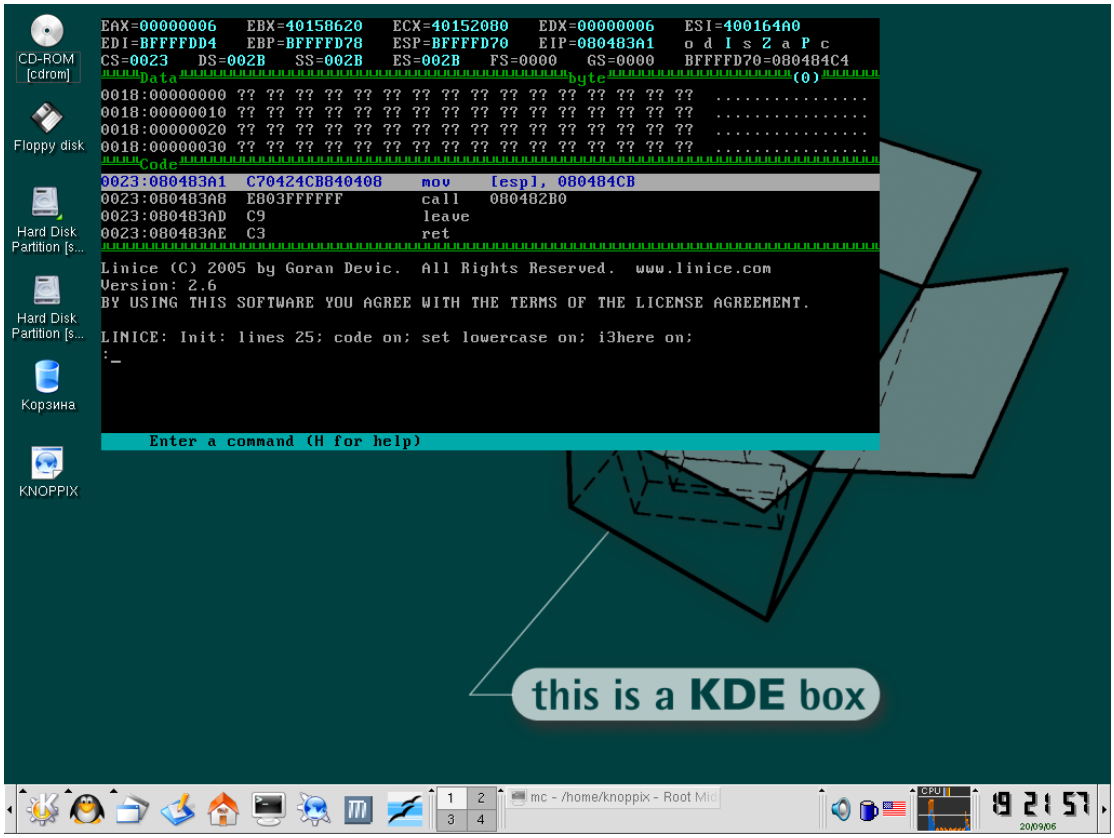

Рисунок 6 внешний вид отладчика LinIce

ядерная отладка в xBSD системах

Ядро xBSD систем включает в себя интегрированный отладчик по имени DDB, поддерживающий как локальную, так и удаленную отладку. По умолчанию ядро собирается без отладчика и чтобы исправить эту вопиющую несправедливость необходимо добавить строку "options DDB" в файл конфигурации ядра (обычно находится в /usr/src/sys/i386/conf/GENERIC) и перекомпилировать его.

Вызывать отладчик можно различными путями: набрав флаг "-d" в приглашении загрузчика мы попадем в DDB на самой ранней стадии загрузки ядра до начала распознавания и подключения устройств, что очень полезно для отладки драйверов.

А вот если хочется взломать какую-нибудь программу, то для вхождения в DDB с консоли (как текстовой, так и графической) достаточно отдать команду "sysctl debug.enter_debugger=ddb" или (если в конфигурационном файле обозначена опция "options BREAK_TO_DEBUGGER") нажать на <CTRL-ALT-ESC>. (***Внимание**: в некоторых раскладках клавиатуры эта комбинация изменена!*)

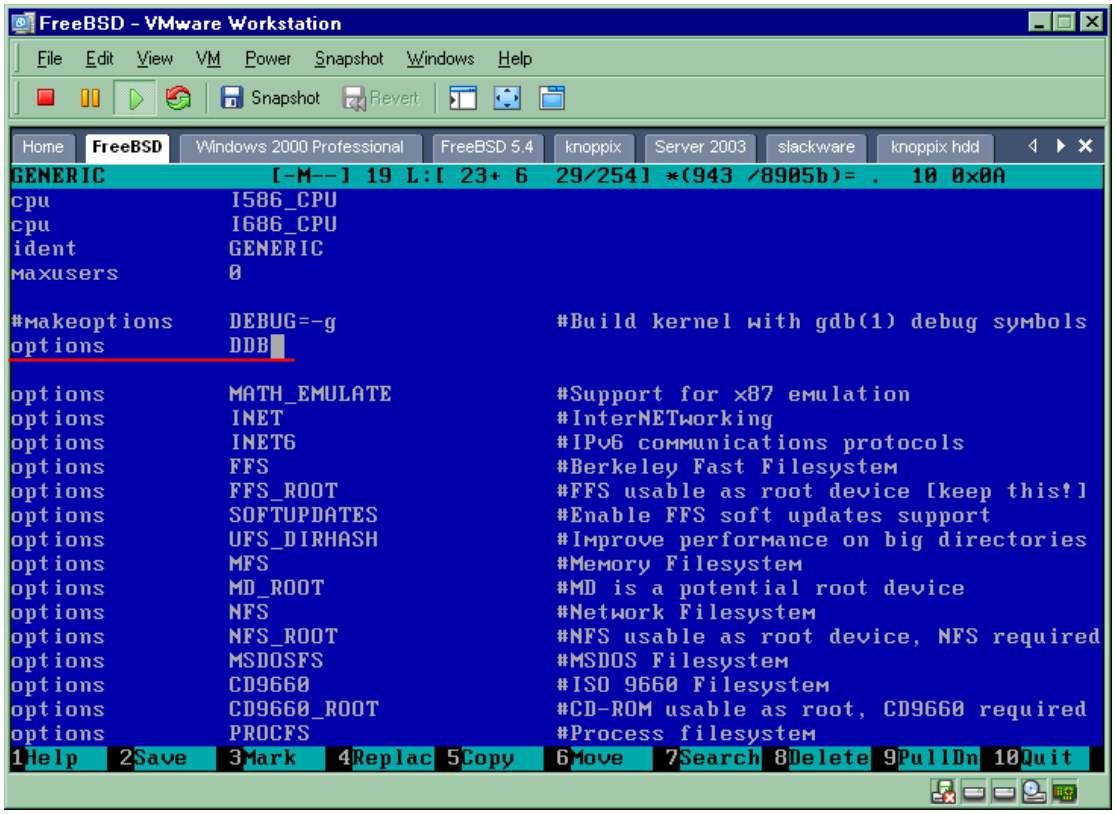

Рисунок 7 изменение конфигурации ядра FreeBSD для подключения отладчика DDB

При желании (и наличии второй машины) можно осуществить удаленную отладку, если возможностей локальной вдруг окажется недостаточно (мне трудно представить такую ситуацию, но… чего в жизни не случается). Официальная документация по FreeBSD говорит, что для этого нам понадобиться две копии ядра: одна — установленная на отлаживаемой системе (пострипанная), другая — удаленная, положенная в один каталог с gdb (откомпилированная с отладочной информацией). На самом деле, наличие отладочной информации совершенно необязательно, gdb будет работать и без нее, пускай и не сможет выйти на уровень исходных текстов — а оно нам нужно? Достаточно, чтобы версии ядер совпадали. Более того, в качестве удаленной системы не обязательно использовать именно FreeBSD. Пригодна _любая_ система, под которую имеется порт gdb (например, Linux, или даже NT), главное — скопировать копию ядра на удаленную систему и загрузить ее в gdb через команду "file".

Но прежде, чем запускать gdb необходимо соединить обе машины COM-шнурком, причем в файле конфигурации ядра потребуется исправить строку, отвечающую за параметры данного порта (она находится в строке "device siox", где x – номер последовательно порта, считая от нуля), уставив флаги в значение 0x80.

Включаем отлаживаемую машину. В командной строке загрузчика набираем "-d", чтобы остановить загрузку системы и войти в отладчик. Включаем удаленную машину и набираем в командной строке "gdb -k kernel" (или же "kgdb kernel" при использовании KGDB), где "kernel" – имя (с путем) к файлу ядра.

Цепляемся отладчиком к последовательному порту, набрав следующую команду "target remote /dev/cuaa0", где cuaa0 — первый последовательный порт, после чего возвращаемся к отлаживаемой машине (которая находится в состоянии ожидания загрузки внутри DDB) и говорим "gdb", сообщая системе, что мы передаем бразды правления удаленному отладчику gdb. Вернуть управление обратно можно при помощи той же самой команды "gdb", фактически представляющий собой триггер между локальным и удаленным режимом отладки.

заключение

Разумеется, мы описали далеко не все существующие ядерные отладчики и оставили без ответа вопрос: какой же из них все-таки самый лучший. Но ведь отладчик — это не религия. Использование нескольких отладчиков вполне нормальное явление.

Лично мыщъх предпочитает такую стратегию: если ломаемая программа запускается под FreeBSD 4.5 (любимая версия мыщъх'а), то задействуется DDB, если же нет, то загружается виртуальная машина с SuSE Linux, где установлен NLDK. Под остальными системами приходится использовать KDB или же KGDB, соединенный с соседней виртуальной машиной, на которой запущен gdb.

LinIce используется в основном для несложных экспериментов (например, наблюдением за rootkit'ми и прочей малварью). Ломать программы в нем мыщъх перестал как только въехал в gdb, рядом с которым soft-ice и рядом не валялся.

Отладочные возможности эмуляторов мыщъх пока еще не раскурил до конца и только начинает их использовать, но чем больше он их использует, тем больше они ему нравятся.

>>> врезка магические SysRq клавиши

Ядро Linux'а, начиная с версии 2.2 поддерживает ряд магических комбинаций клавиш, вызываемых по <ALT-SysRq-Key>, где "Key" – магическая клавиша, например, <s>. Магические клавиши полезны для отладки и борьбы с малварью (например, клавиша <k> – убивает все процессы в текущей консоли).

Чтобы включить магические SysRq клавиши при компиляции ядра параметр CONFIG_MAGIC_SYSRQ необходимо установить в состояние "yes" (в большинстве дистрибутивов это уже сделано за нас, если же это не так, а перекомпилировать ядро лень, то включить магические клавиши можно и на лету, отдав команду "echo 1 > /proc/sys/kernel/sysrq").

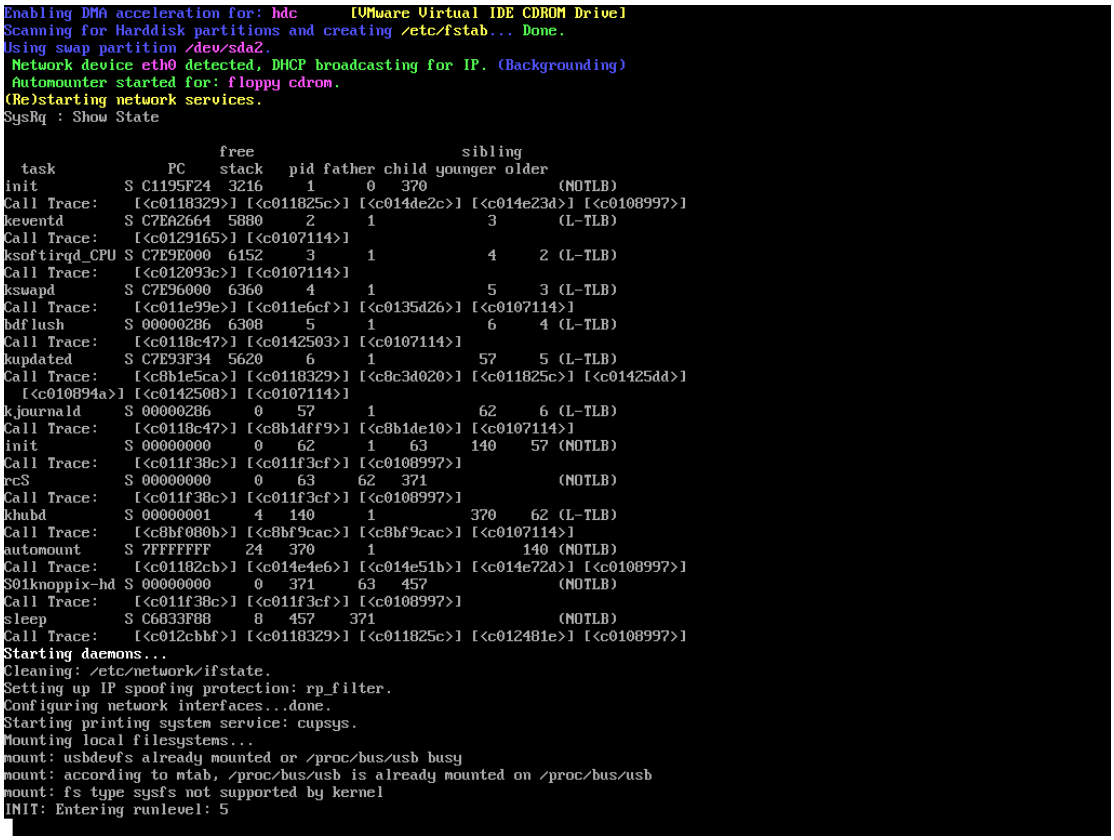

Рисунок 8 вывод списка процессов при помощи магической комбинации <ALT-SysRq-t>

Перечень магических клавиш лежит в файле /src/Documentation/sysrq.txt, а так же может быть получен по ссылке: http://www.gelato.unsw.edu.au/lxr/source/Documentation/sysrq.txt

- 'r' – отключение сырого клавиатурного ввода;
- 'k' —процедура "Secure Access Key" (сокращенно — SAK), убивающая все процессы на текущей виртуальной консоли;
- 'b' — немедленная перезагрузка без размонирования дисков;
- 'c' — создание crashdump'а, пригодного для последующего анализа;
- 'o' — нормальный шатдаун системы;

- 's' — сброс дисковых кэшей для всех смонтированных томов;
- 'u' — перемонтирование всех томов "только-на-чтение";
- 'p' — отображение содержимого регистров процессора;
- 't' — отображение текущего списка процессов;
- 'm' — отображение об использовании памяти;
- '0'-'9' — установка уровня отладочного вывода printk;
- 'e' — посылка сигнала SIGTERM всем процессам кроме init;
- 'i' — посылка сигнала SIGKILL всем процессам кроме init;
- 'l' — посылка сигнала SIGKILL всем процессам, включая init;
- 'h' — вывод списка магических SysRq клавиш;