РАЗРАБОТКА ПОД LINUX*, ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ*, ЗАНИМАТЕЛЬНЫЕ ЗАДАЧКИ, БЛОГ КОМПАНИИ НЕОБИТ

Виртуальные твари и места их обитания: прошлое и настоящее TTY в Linux

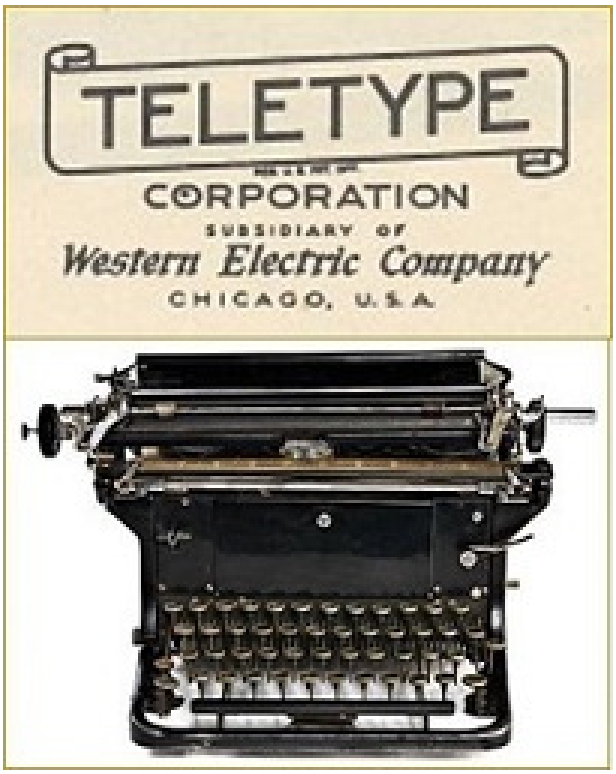
NWOcs 23 июня 2017 в 09:03 25,5k

Ubuntu интегрирована в Windows 10 Redstone, Visual Studio 2017 обзавелась поддержкой разработки под Linux – даже Microsoft сдает позиции в пользу растущего числа сторонников Торвальдса, а ты всё еще не знаешь тайны виртуального терминала в современных дистрибутивах?

Хочешь исправить этот пробел и открываешь исходный код? TTY, MASTER, SLAVE, N_TTY, VT, PTS, PTMX… Нагромождение понятий, виртуальных устройств и беспорядочная магия? Всё это складывается в довольно логичную картину, если вспомнить, с чего всё началось…

1. START FROM SCRATCH & KEEP CALM

TTY: ПАЛЕОЗОЙ

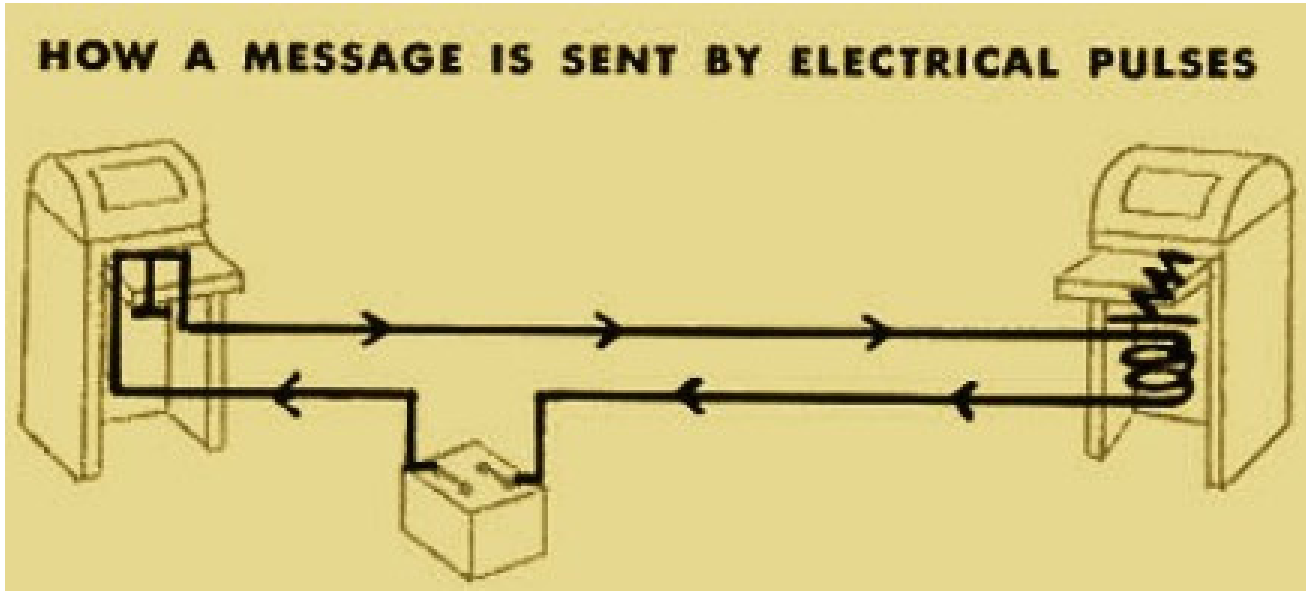


Мы шагнули прямиком в тридцатые годы XX века и оказались в совсем еще молодой Teletype Corporation. Прямо перед нами перед нами Тот-С-Которого-Всё-Началось – телетайп, представляющий из себя «буквопечатающий телеграф», который передает текстовые сообщения между двумя абонентами.

Абонент А набирает на клавиатуре символы, которые преобразуются в электрические сигналы. По самому обычному кабелю сигналы «бегут» на телетайп абонента Б и уже там печатаются на самой обычной бумаге. Если сигнал дуплексный, то нам крупно повезло, и абонент Б может сразу написать свой ответ; если нет – то ему потребуется сначала подсоединить второй провод для обратной связи.

Здесь, на Teletype Corporation, еще не знают, какое будущее в скором времени ждет их продукт, и уж конечно не подозревают, что аббревиатура TTY намного переживет сам телетайп. Не будем портить для них интригу, пойдем дальше.

TTY: МЕЗОЗОЙ



Прошло сорок лет, мы в лаборатории Digital Equipment Corporation, любуемся первым мини-компьютером (интерактивным!) PDP-1. Для ввода и вывода информации, а также для обеспечения взаимодействия с пользователем к нему подключен уже знакомый нам телетайп.

Дело в том, что ведущие инженерные умы решили велосипед не изобретать и приспособить уже имеющийся дешевый и доступный механизм под новые нужды. Телетайп напрямую подключили к компьютеру (а не к другому телетайпу, как это было раньше) и назвали это дело **консолью**. Оператор, осуществляющий ввод, видит, как

набираемые символы мгновенно печатаются на бумаге, но происходит это без участия ОС – благодаря сохранению в **консоли** принципа печатающей машинки.

TTY: ПАЛЕОГЕН

Оказываемся в самом начале 80-ых годов, на этот раз в Bell Laboratories. Здесь только что выпущен один из важнейших релизов «раннего» UNIX – Version 7 для PDP-11. Особенности у этого релиза следующие: вводимая пользователем команда теперь отображается по принципу ECHO (набранный на клавиатуре символ сначала попадает в буфер накопления и только потом ОС отправляет инструкцию вывести этот символ на печать), поддерживаются простые возможности редактирования вводимых команд (можно «стирать» символ или целую строку, перемещать каретку), появляется разделение режимов:

- **raw mode** (редактирование строки не производится; управляющие последовательности распознаются как обычные символы; введенный символ немедленно передается процессу);
- **cooked mode** (происходит распознавание специальных символов и генерирование сигналов остановки и прерывания

для процесса; передача готовой строки процессу осуществляется только после нажатия клавиши Return).

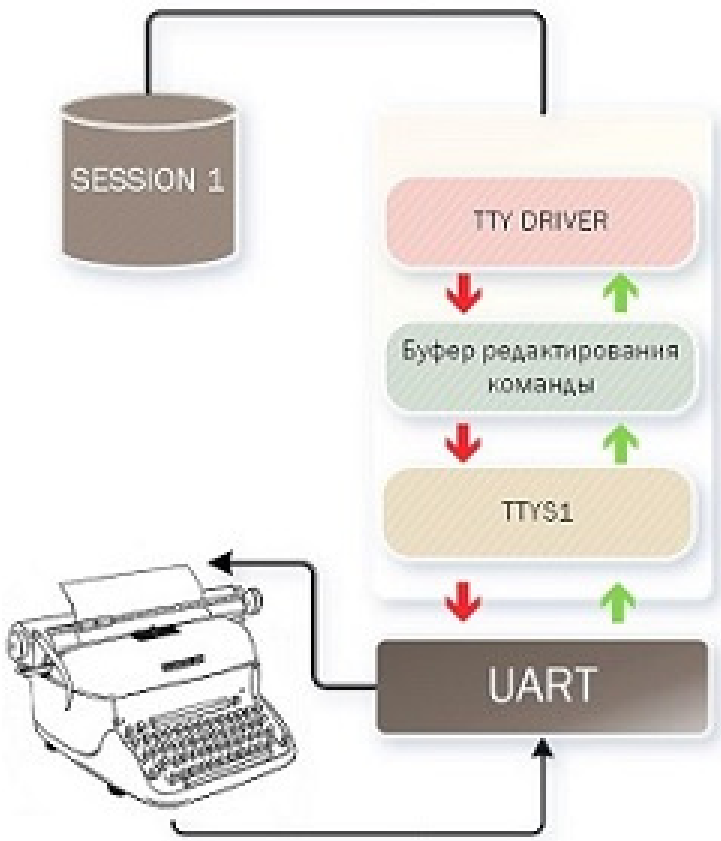
Вполне ожидаемый вопрос: как же можно «стереть» то, что уже напечатал телетайп? Для наглядного выполнения операций редактирования Unix Version 7 предусмотрена печать определенных символов: например, @ — стереть всю строку, # — стереть последний символ. То есть, если наш телетайп напечатал ld@lk#s, и оператор нажал Return, то на исполнение пошла команда ls. Это еще не TTY LINE DISCIPLINE (о ней речь пойдет дальше), но уже большой шаг вперед в отношении обработки ввода на уровне ОС.

Кстати говоря, Digital Equipment Corporation за эти 20 лет не только разработала упомянутый PDP-11, но и подумала о том, как усовершенствовать телетайп: появились так называемые умные терминалы.

Смотрим направо: это VT100, один из первых терминалов, умеющих работать в любви и согласии с PDP-11 и поддерживаемый Unix Version 7.

На уровне ОС и консоль, и умный терминал сейчас воспринимаются как символьные устройства, которые подключаются через интерфейс UART, преобразовывающий

асинхронный поток данных в последовательность символов. Для ОС они в принципе идентичны, разница лишь в том, стирать ли символы на экране терминала или печатать символ забоя с помощью телетайпа.



Слева показана в общих чертах схема взаимодействия компьютера и консоли (или умного терминала). У этой схемы есть один недостаток, который совсем не радует оператора консоли PDP-11: с одной консолью ассоциируется одна сессия (или сеанс), в котором пользователь может в фоновом режиме запустить несколько процессов, однако активным в один момент времени на одном TTY будет только один.

И наш бедный оператор вынужден в прямом смысле этого слова переходить от одной консоли к другой, если вдруг ему придет в голову поработать с несколькими сессиями.

TTY: НЕОГЕН

Мы очутились в редакции журнала «PC MAGAZINE»,

рассматриваем свежий выпуск от 13 января 1987 года. Один из разворотов активно убеждает нас не жалеть денег на ПК с UNIX System V. Каковы аргументы? В частности – grep, awk, sort, split, cut, paste, vi, ed – word processing явно шагнул вперед. И самое интересное: к нашим услугам сейчас эмуляторы терминалов! Благодаря виртуальным консолям, уже можно запустить целых четыре сессии без нужды подключать всё новые и новые физические телетайпы.

Кроме того, жизнь монохромных терминалов нынче можно расцветить: поддерживается CGA, Hercules, EGA графика. Бедолага-оператор может вздохнуть спокойно, телетайп (а также умный терминал) в виде железного зверя угрожает ему только в ночных кошмарах.

Посоветуем оператору лишь одно: ни за что не лезть в директорию /dev – ведь его ждут там аж несколько ttyX и напоминают о том, что под капотом виртуальной консоли живет всё тот же старый-добрый телетайп.

TTY: АНТРОПОГЕН

На предыдущем шаге мы убедились: консоль сделали виртуальной (не будет же солидное издание «PC MAGAZINE» лгать). Что это значит – весь механизм ввода/вывода переписан и в корне изменен? Тогда почему виртуальное устройство – всё еще ttyX? Всё просто: виртуальная консоль эмулируется как самая что ни на есть физическая, а место UART-драйвера, вероятно, занимает кто-то другой. Изменившуюся схему подробно обсудим чуть дальше.

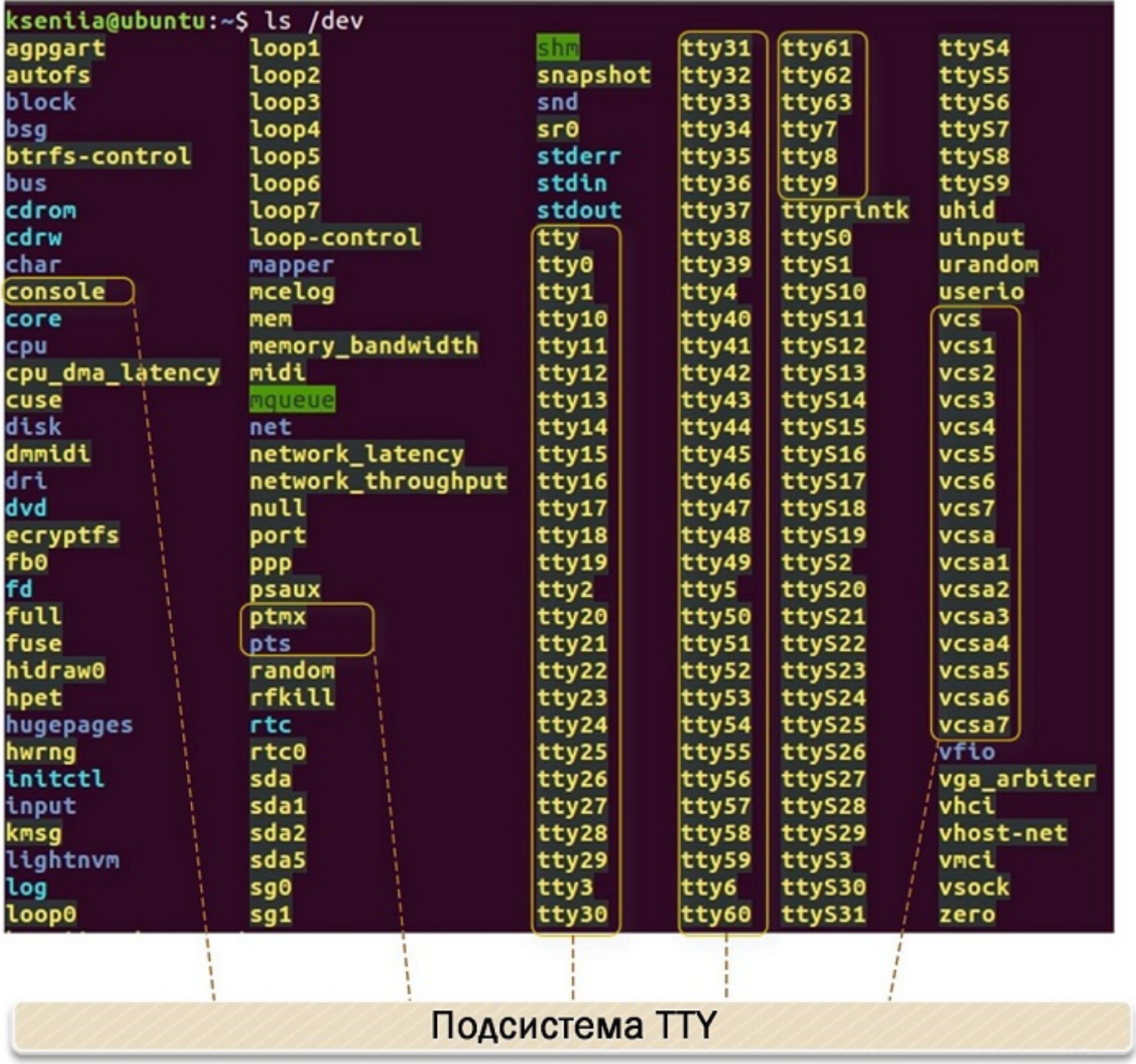
До финишной прямой один шаг, но пропустить его мы не можем хотя бы из уважения к Линусу Торвальдсу. Сейчас 1993 год, и мы наконец имеем счастье рассматривать исходный код Linux 0.95. Почему именно этот релиз? Именно в нем уже сформировалась TTY-абстракция, наиболее близкая к тому, что мы имеем в самых последних дистрибутивах: оформились три обособленных слоя (**TTYX — TTY_LINE_DISCIPLINE — TTY_DRIVER**).

Кроме того, спустя всего год будет выпущен Linux 1.0, где появится оконный интерфейс, предоставленный проектом XFree86. С этого момента к **виртуальным консолям** добавятся в придачу еще **виртуальные терминалы**, которые пользователь (в почти не ограниченном количестве) сможет запускать, не покидая графическую оболочку… Однако прежде, чем окунуться в тонкости и детали усовершенствованной io-магии, вернемся в наше настоящее к Ubuntu 16.04.

2. STOP BEAT AROUND THE BUSH & LOOK INSIDE

ВИРТУАЛЬНЫЕ ТВАРИ И МЕСТА ИХ ОБИТАНИЯ

Лишь некоторые устройства директории /dev/ используются повседневно: /dev/sdaX, /dev/mem, /dev/zero, /dev/random… Но есть несколько групп устройств, которые не часто привлекают наше внимание, однако более чем его заслуживают. Это устройства /ttyX, /vcsX, /vcsaX, а также /ptmx и /pts/X. Собственно говоря, о них и пойдет речь дальше.



И первый наш объект – виртуальная консоль. Каждому такому объекту присущи как минимум ~~сакральное число~~ идентификатор и ~~тотемное животное~~ файл виртуального устройства /tty, коих в виртуальном лесу директории /dev встречают аж 64.

Проверим, есть ли у нас возможность пообщаться с ними. Выполняем Ctrl-Alt-FX (или chvt X, где X – номер консоли, например, Ctrl-Alt-F1) и замечаем, что X может быть равно 1, 2 …

6. При этом перед нами открывается виртуальная консоль, при первом запуске нам предлагают ввести имя пользователя и пароль и создают для нас новый сеанс работы. Если X равен 7, то мы возвращаемся в родные графические пенаты и понимаем, что /tty7 связан с XServer’ом. Идем дальше. Восемь, девять, десять и так далее до 63 — признаков жизни не подают.

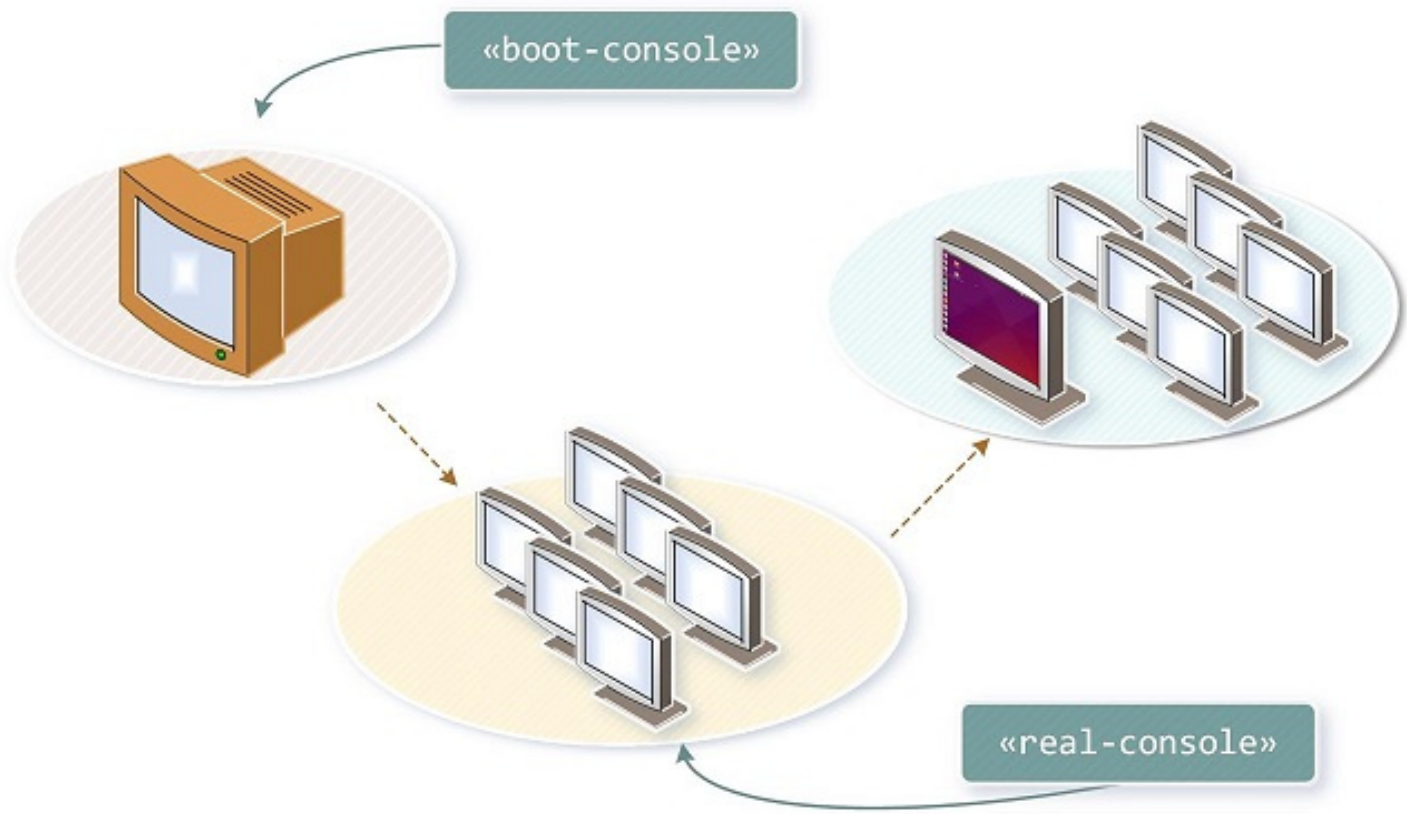
```
root@ubuntu:/etc/default# cat console-setup
# CONFIGURATION FILE FOR SETUPCON

# Consult the console-setup(5) manual page.

ACTIVE_CONSOLES="/dev/tty[1-6]"

CHARMAP="UTF-8"

CODESET="guess"
FONTFACE="Fixed"
FONTSIZE="8x16"

VIDEOMODE=

# The following is an example how to use a braille font
# FONT='lat9w-08.psf.gz brl-8x8.psf'
```

Дело в том, что в Linux есть макрос MAX_NR_CONSOLES (64), определяющий максимально допустимое число виртуальных консолей, которые и представлены 64-мя файлами виртуальных устройств /dev/ttyX. Однако последнее слово остается за параметром ACTIVE_CONSOLES (/etc/default/console-setup), и параметр этот по умолчанию равен шести.

Инициализация консолей происходит в несколько стадий. Сперва ядро, получив управление от Grub’a, в ходе инициализации подсистем вызывает функцию «console_init», которая создает первичную консоль – «boot console», предназначенную для вывода отладочной информации. Это консоль осуществляет вывод символов самым примитивным образом: через «putchar», которая

напрямую обращается к BIOS, инициализируя и заполняя структуру biosregs, и осуществляет вывод символа в консоль, используя прерывание 0x10.

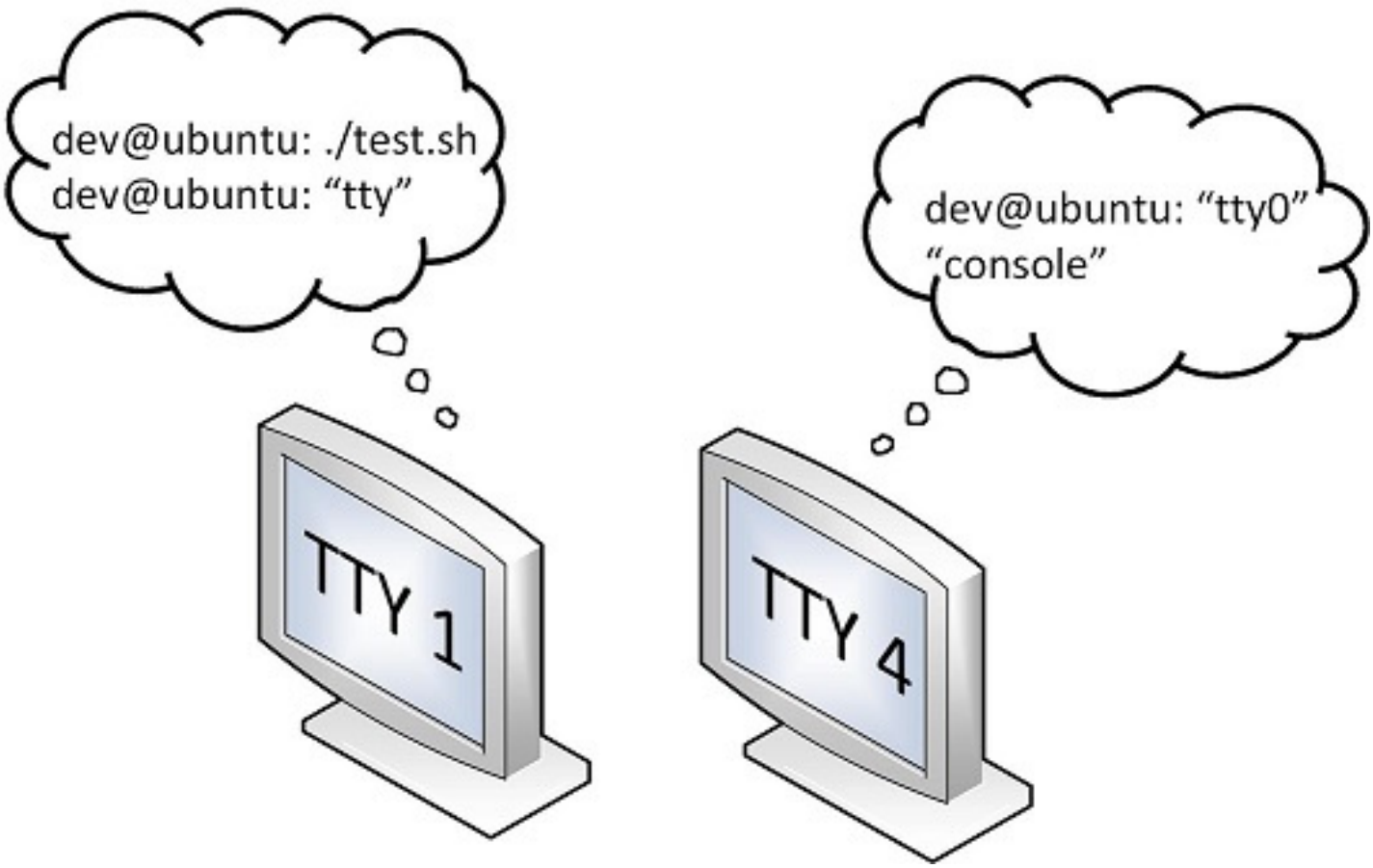
Позже, в ходе выполнения «fs-initcall» и «console-initcall» происходит создание виртуальных устройств и структур под 6 полноценных виртуальных консолей – «real console». Активацию этих консолей выполняет первый запущенный ядром процесс /sbin/init, запускающий программу getty, которая выполняет чтение конфигурационных файлов /etc/init/console.conf и /etc/init/ttyX.conf и впоследствии отображает на консоль содержимое файла-приветствия etc/issue и запускает login. Далее XServer инициирует активацию консоли на dev/tty7, на которой запускается графическая оболочка.



Однако у нас есть еще вопросы. Что за неведомый объект /dev/tty0? И если каждый /dev/ttyX — это виртуальное устройство консоли, то зачем нужны /dev/console и /dev/tty? За ответом переходим на tty1 (нажимая Ctrl-Alt-F1) и в фоновом режиме запускаем такой скрипт:

```
sleep 10
echo “tty0” > /dev/tty0
echo “tty” > /dev/tty
echo “console” > /dev/console
```

Затем переходим на, скажем, tty4 и ждем несколько секунд. По истечении видим следующую картину:

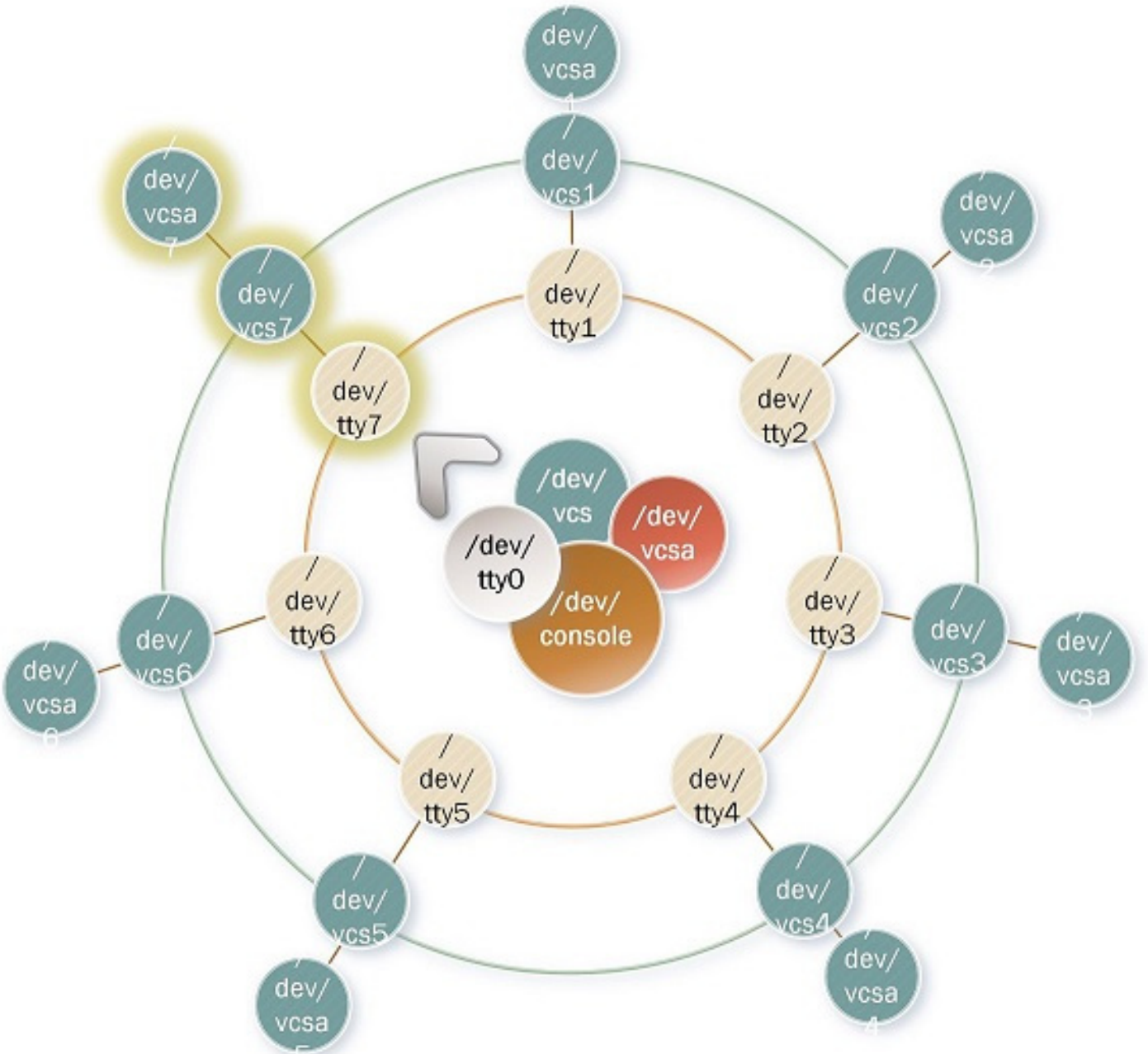


Распределение ролей становится понятно: /dev/tty0 = /dev/console = текущая консоль, т.е. оба всегда ассоциированы с той консолью,

которую мы в данный момент видим перед собой, а /dev/tty «помнит» консоль, с которой стартовал процесс. Поэтому пока наш процесс выполнялся, /dev/tty0 и /dev/console определялись для него по ходу пьесы в зависимости от текущей активной консоли, а вот /dev/tty оставался неизменным.

Не спешим покидать директорию /dev. Здесь еще чуть больше дюжины любопытных объектов: /dev/vcsX (virtual console screen) и /dev/vcsaX (virtual console screen with attributes). Еще один опыт: перемещаемся на tty5 и оставляем какие-нибудь следы своего пребывания, затем переходим в любую другую консоль (пусть ее номер 3), делаем «cat» на /dev/vcs5 и видим именно то состояние консоли 5, в каком мы оставили ее несколько секунд назад. При этом, соответственно, /dev/vcs3 и /dev/vcs (а также /dev/vcsa) относятся к консоли 3, на которой мы находимся в данный момент.

Понимаем, что /dev/vcsX — не что иное как ~~омут памяти~~ устройство виртуальной памяти консоли, позволяющее нам без потерь перемещаться между экземплярами tty. В паре с ним — /dev/vcsaX, который предоставляет базовые сведения о состоянии экрана: цвета, различные атрибуты (напр. мерцание), текущее положение курсора, конфигурацию экрана (количество строк и столбцов). Подытожим увиденное схемой:



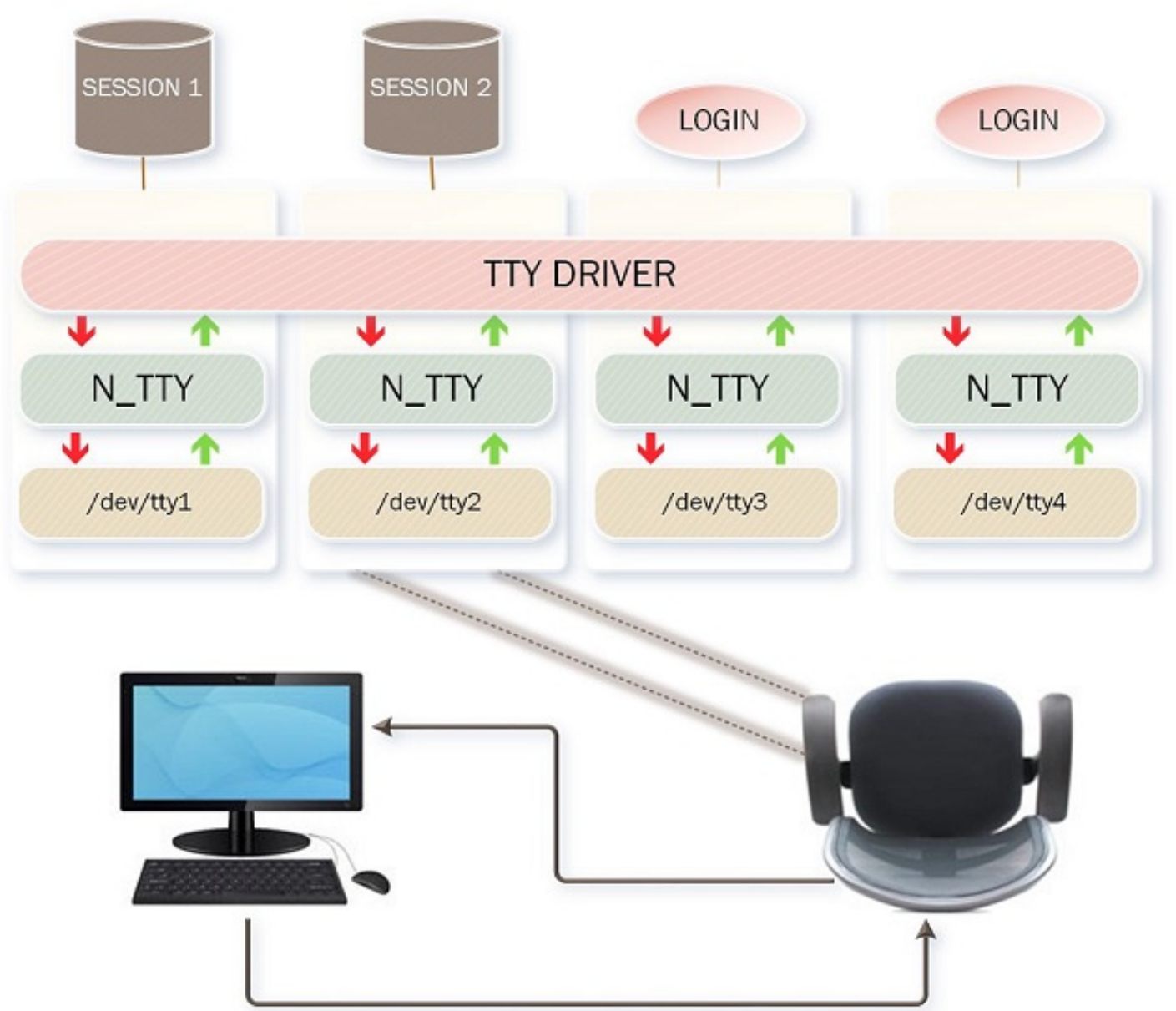
ВИД СВЕРХУ ЛУЧШЕ

Теперь остановимся с изучением зоологии tty на какое-то время и перейдем к самой tty-абстракции, частью которой и являются наши виртуальные устройства. Посмотрим на общую структуру tty-комплекса и выделим три компонента:

1. **/dev/ttyX** – виртуальное устройство консоли в файловой системе, которое заняло место UART-драйвера и с которым мы уже знакомы. На этом же уровне располагаются устройства /dev/vcsX и /dev/vcsaX, общение с ними осуществляется непосредственно через /dev/ttyX.
2. **TTY Line Discipline** — драйвер, который делает ECHO набираемой команды и дает нам возможности ее редактирования. Также драйвер этого слоя генерирует сигналы при наборе управляющих последовательностей (^C, ^Z и т.д.). По умолчанию здесь царствует N_TTY, однако этот модуль можно заменить, например, своим драйвером – с этим поэкспериментируем немного позже;
3. **TTY driver** – драйвер, который предоставляет набор методов инициализации и открытия консоли, а также методы, обрабатывающие операции ввода/вывода, приостановку консоли при переключении и возобновление ее работы и, конечно, обеспечивает «передачу» полученной от пользователя команды активному процессу.

Помните жалобы оператора PDP-11? Ему не нравилось тратить время на переходы от одной физической консоли к другой. Сейчас дело обстоит следующим образом: у нас есть по умолчанию 7 виртуальных консолей, а перед ними — офисное кресло на колесиках (разумеется, тоже виртуальное). Когда мы переключаемся с одной консоли на другую, операционная система перемещает наше кресло к нужному tty, а вместе с креслом «переключаются» на него и комплекс физических io-устройств: на мониторе теперь состояние нашей новой консоли, на нее же поступает ввод с клавиатуры и т.п.

При этом процессы от первого tty продолжают работать: считывают команды с файла своей виртуальной консоли, пишут в этой файл, но – так как они оторваны от «кресла» – не получают никаких событий (те же ^C и ^Z) и – так как физические устройства «уехали» вместе с «креслом» – могут только накапливать свой «вывод» в буфере, чтоб отправить его на монитор, как только «кресло» вернется.

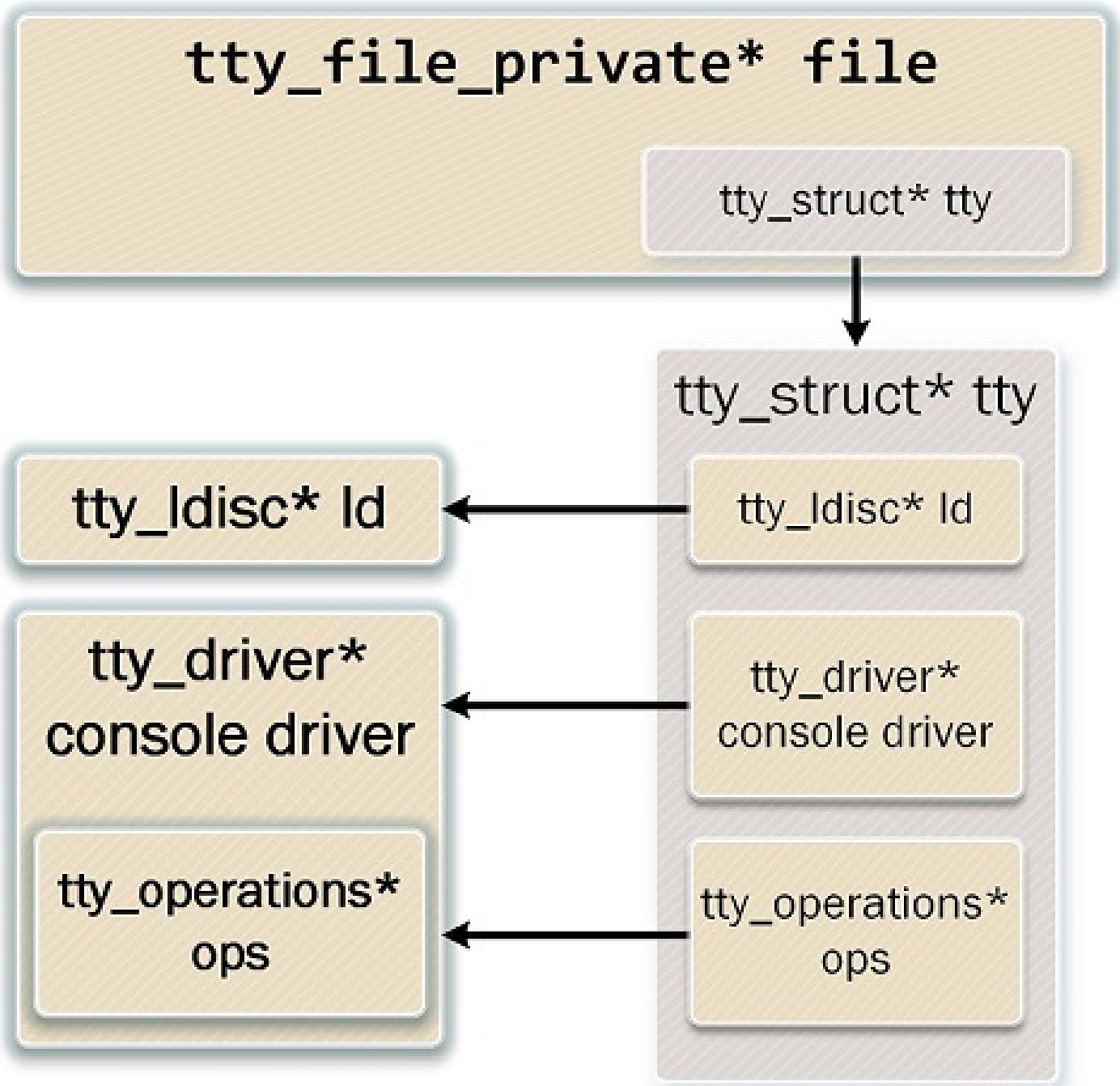


ПУТЕШЕСТВИЕ К ЦЕНТРУ ЗЕМЛИ

Да, сверху всё смотрится вполне презентабельно. Но тебе, %username%, вероятно, хочется увидеть, как трехуровневое взаимодействие tty-компонентов реализовано непосредственно в коде? За ответом придется опуститься с небес на землю, даже лучше сказать – под землю, в недра исходного кода Ubuntu (работать будем с ядром версии 4.4).

Сделаем упреждающий ход – разберемся, через какие структуры происходит связывание tty-абстракции в единое целое.

Во-первых, это «tty_struct», у которой есть поле «tty_ldisc» (это структура методов драйвера 2-ого слоя), поле «tty_driver» (это драйвер 3-ого слоя) и тут же «tty_operations» (это структура методов драйвера 3-ого слоя, ради удобства вынесенная прямо в «tty_struct»).



То есть, «tty_struct» обеспечивает доступ к слоям TTY_LINE_DISCIPLINE и TTY_DRIVER. Получили к ней доступ – 2/3 стека tty-абстракции, считай, перед нами. Теперь нам нужно понять, как осуществляется переход от файлов виртуальных устройств к этой самой структуре. Ответ прост: у структуры «tty_file_private» как раз есть поле типа «tty_struct». Следовательно, обращаясь к файлу виртуального устройства на 1-ом уровне, мы с легкостью получаем доступ к уровням повыше.

Пока пазл складывается, но нам этого недостаточно. Продебажим ядро (с помощью qemu и cgdb) и рассмотрим backtrace вывода (эхо) единичного символа, введенного пользователем с клавиатуры:

```
690  static void vgacon_cursor(struct vc_data *c, int mode)
691 >{
692          if (c->vc_mode != KD_TEXT)
693                  return;
694
695          vgacon_restore_screen(c);
696
697          switch (mode) {
698          case CM_ERASE:
699                  write_vga(14, (c->vc_pos - vga_vram_base) / 2);
700                  if (vga_video_type >= VIDEO_TYPE_VGAC)
701                          vgacon_set_cursor_size(c->vc_x, 31, 30);
/usr/src/linux-4.4/drivers/video/console/vgacon.c
ole/vgacon.c:691
(gdb) backtrace
#0  vgacon_cursor (c=0xffff880007086000, mode=2) at drivers/video/console/vgacon
.c:691
#1  0xffffffff8151a808 in hide_cursor (vc=0xffff880007086000) at drivers/tty/vt/
vt.c:605
#2  0xffffffff815200b3 in do_con_write (tty=0xffff8800066e1000, buf=<optimized o
ut>, count=<optimized out>) at drivers/tty/vt/vt.c:2218
#3  0xffffffff815206ac in do_con_write (count=<optimized out>, buf=<optimized ou
t>, tty=<optimized out>) at drivers/tty/vt/vt.c:2779
#4  con_write (tty=0xffff8800066e1000, buf=<optimized out>, count=<optimized out
>) at drivers/tty/vt/vt.c:2775
#5  0xffffffff815091d5 in process_output_block (nr=<optimized out>, buf=<optimiz
ed out>, tty=<optimized out>) at drivers/tty/n_tty.c:610
#6  n_tty_write (tty=0xffff8800066e1000, file=<optimized out>, buf=0xffff8800067
40800 "f # ase press Enter to activate this console. ", nr=1) at drivers/tty/n_t
ty.c:2375
#7  0xffffffff81505521 in do_tty_write (count=<optimized out>, buf=<optimized ou
t>, file=<optimized out>, tty=<optimized out>, write=<optimized out>) at drivers
/tty/tty_io.c:1164
#8  tty_write (file=0xffff8800066f4c00, buf=0x7f332a2f7000 "f # ", count=<optimi
```

Итак, мы на I уровне tty-стека. Происходит системный вызов «write», который на нашем tty обрабатывается функцией «tty_write». В нее передаются указатель на структуру файла виртуального устройства и буфер с символом. В функции «tty_write» по файлу происходит получение экземпляра «tty_struct». Принимая игру, «tty_struct» первым делом вызывает

драйвер TTY_LINE_DISCIPLINE – «tty_ldisc», место которой по умолчанию занимает N_TTY. Первый уровень пройден!

N_TTY принимает эстафету: в свою очередь вызывает метод «n_tty_write», а затем передает буфер функции «output_process_block», которая, удостоверившись, что мы ввели не символ позиционирования каретки, просит «tty_struct» позвать «tty_driver». Всё верно, мы переходим на III уровень.

«Tty_struct» успешно играет роль посредника, и вот – уже запускается метод «con_write» tty-драйвера по имени «console_driver». Драйвер III-его уровня рад бы выполнить своё дело, но он один, консолей много – с какой надо работать? На помощь опять приходит «tty_struct» и вручает драйверу нужный экземпляр структуры «vc» (она отвечает за состояние своей конкретной консоли и содержит её клавиатурные, экранные установки, а также набор методов графического отображения).

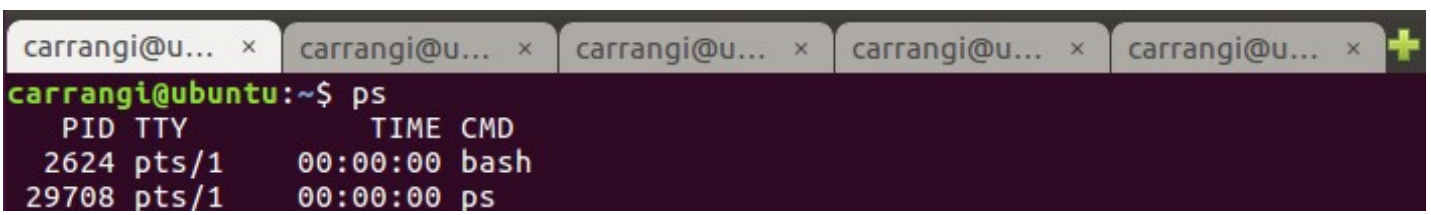
«Console_driver» блокирует консоль и призывает «vc_data» выполнить наконец эхо символа. «Vc_data» с ужасом осознает: к ней обратились не ради вопроса о самочувствии вверенной ей консоли, а ради действия. Это значит лишь одно: пора звать на помощь методы «consw», которые в нашем случае представляет VGA (при другой конфигурации ядра это может быть, например, framebuffer). И точно – VGA споро берется за дело, скрывает курсор, печатает символ, при необходимости прокручивает экран или переходит на новую строку, перемещает и отображает курсор. «Vc» выдыхает: «console_driver» работу принял и консоль разблокировал. Можем выдохнуть и мы, ведь все три уровня

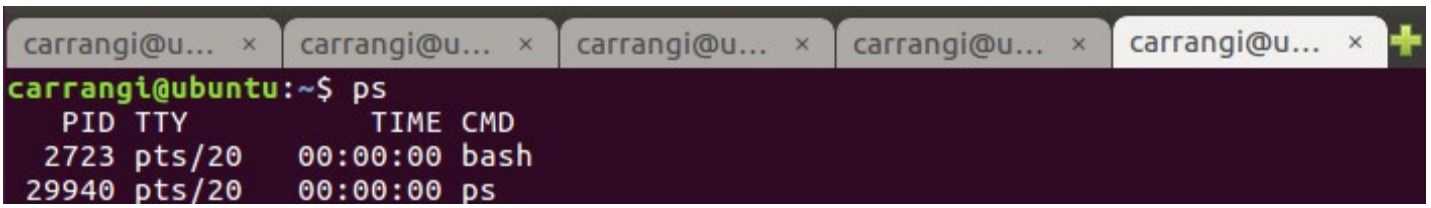
успешно пройдены, каждый компонент свою миссию выполнил.

3. KEEP AN EYE ON VIRTUAL TERMINAL

Но это еще не всё: пришло время познакомиться с представителями еще одного класса обитателей /dev – с **эмуляторами терминалов**. Это те самые xterm или gnome-terminal, которые мы запускаем с консоли, оснащенной графической оболочкой, используя, например, Ctrl-Alt-T или Ctrl-Shift-T.

Живут они в отдельном вольере /dev/pts (=pseudo-terminal slave) и представляют собой файлы под номерами 0, 1, 2 и т.д. Выполняем ps в текущем терминале и видим – мы на /dev/pts/1. Нажимаем Alt+5 – перемещаемся на наш четвертый по порядку открытия терминал, файл виртуального устройства которого /dev/pts/20. На любом терминале нас встречает bash, с каждым терминалом связано своё множество процессов. Пока никаких сюрпризов.



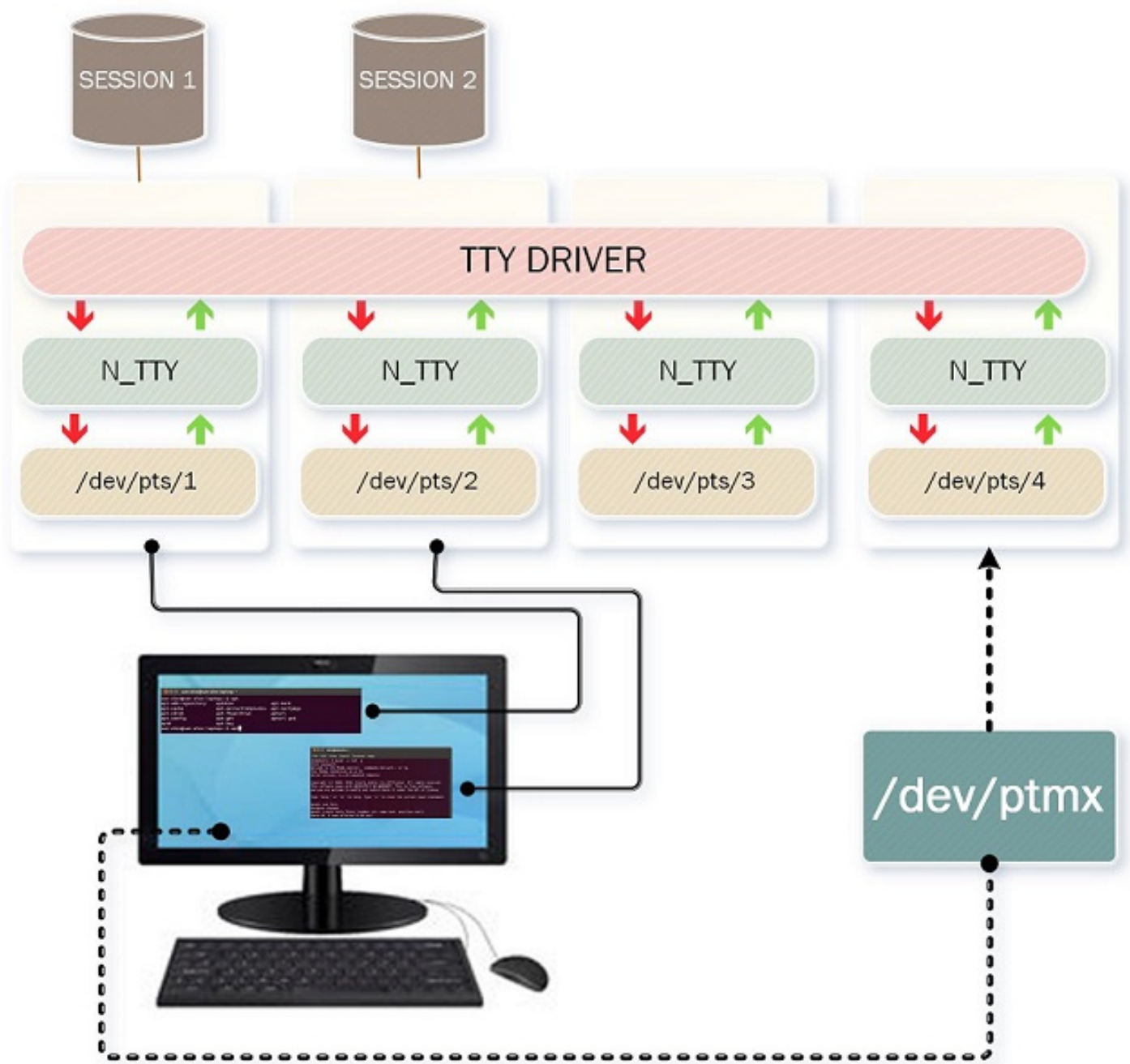


Но заметим: /dev/pts/X создаются динамически, запускаются на одной консоли /dev/tty7, не требуют запуска login и самое интересное – здесь нет правила «кресла»: мы можем открыть несколько виртуальных терминалов и одновременно наблюдать, как происходит работа на каждом из них. В очередной раз у %username% может появиться повод для сомнений: сохраняется ли и здесь принцип tty-абстракции и как в этот принцип вписывается устройство — slave, которым, вероятно, управляет некое устройство — master?

Новый объект не заставляет себя ждать: в единственном экземпляре файл ptmx лежит в той же директории /dev (На самом деле он может быть и не один: если запущено более одной консоли с графической оболочкой). По ману, при открытии /dev/ptmx создается подчиненная часть псевдотерминала /dev/pts/X, связанного со своей ведущей частью «ptm» (обращение происходит через дескриптор файла, но реальный файл не создается). Затем «ptm» передается в функции grantpt и unlockpt, и после всего этого можно открывать непосредственно /dev/pts/X, который будет вести себя точно так же, как виртуальная консоль (за исключением описанных выше особенностей).

Для нас в ключе идеи tty-абстракции это означает следующее: когда пользователь хочет запустить **эмулятор терминала**, XServer обращается к /dev/ptmx с просьбой создать виртуальное устройство /dev/pts/X. Могущественный «мультиплексер» /dev/ptmx любезно делает это, закрепляет файл устройства за экземпляром терминала и … /dev/pts/X занимает место /dev/ttyX, ему

назначается драйвер слоя TTY_LINE_DISCIPLINE, его ласково принимает в свои объятия TTY_DRIVER. Стек над /dev/pts/X принимает уже привычный вид. Задача изучения механизма **эмулятора терминала** плавно сводится к предыдущей истории с **виртуальной консолью**, однако его подробное изучение требует отдельной статьи (которая входит в планы на будущее!).



4. LET'S PLAY WITH TTY LINE DISCIPLINE

На секунду вспомним tty-«палеозой»: было время, когда слой TTY_LINE_DISCIPLINE и отдельным слоем-то не был и полноценной современной функциональностью не обладал. Попробуем оценить вес перемен, произошедших с тех пор.

Для начала убедимся, что мы действительно имеем дело с N_TTY:

```
carrangi@ubuntu:~$ sudo cat /proc/tty/ldiscs
[sudo] password for carrangi:
n_tty       0
```

Всё познается в сравнении, поэтому действуем кардинально и с помощью stty отключаем все полезные фичи N_TTY:

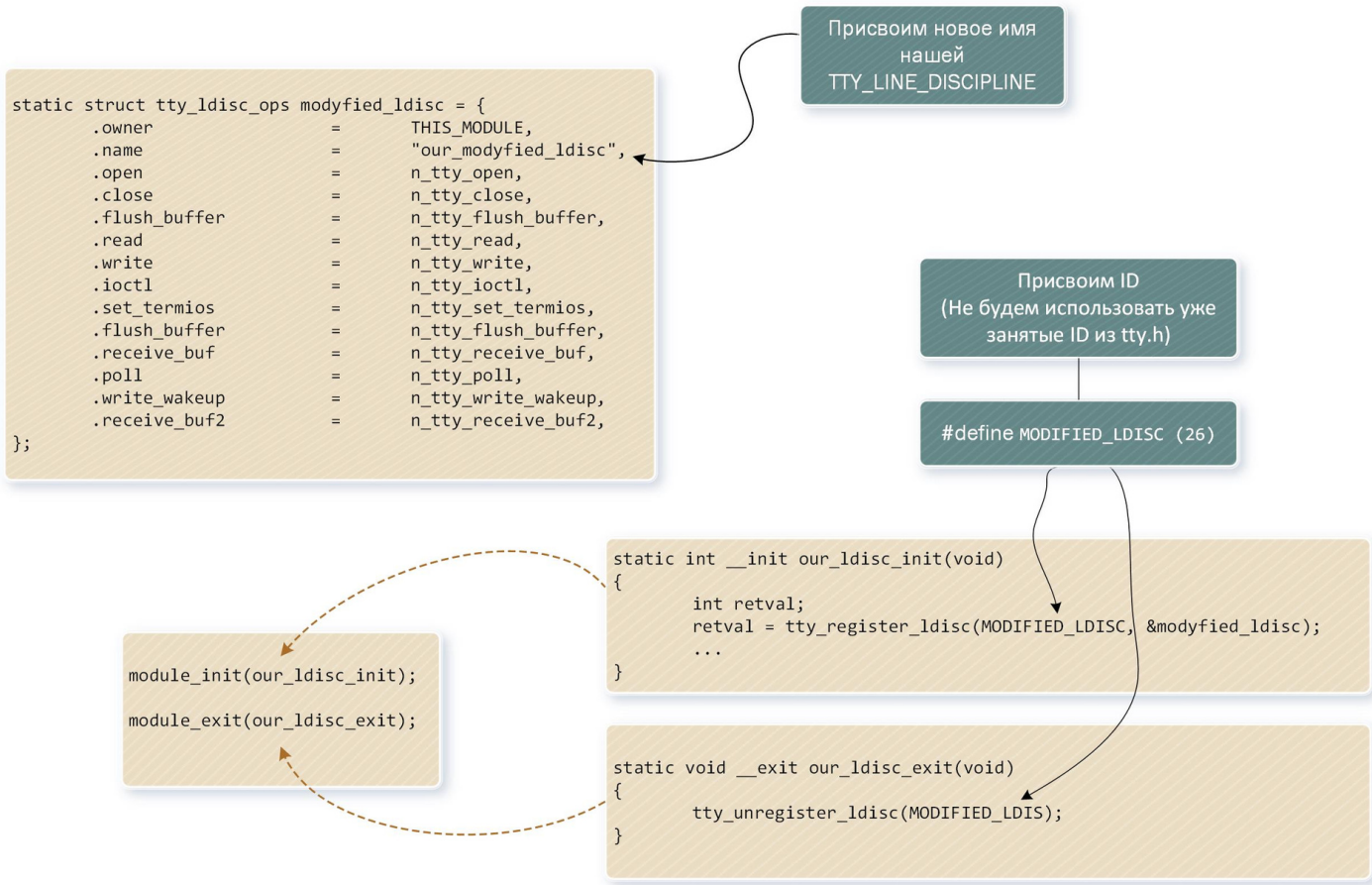
```
carrangi@ubuntu:~$ stty raw -echo
carrangi@ubuntu:~$ we have not any ldisc
                                    carrangi@ubuntu:~$ Desktop    Downloads
        Music     Public     Videos
                                 Documents  examples.desktop  Pictures  Templa
tes
```

Результат наглядно демонстрирует область ответственности N_TTY, без которой вывод не форматируется, ввод не отображается. Причем, если мы откроем новый терминал, то убедимся в целостности и невредимости его LINE_DISCIPLINE. Полученный эффект наталкивает на мысль: мы точно знаем, какой компонент обрабатывает весь наш ввод, мы можем модифицировать его для каждого виртуального терминала в отдельности и, помнится, мы слышали, что этот компонент можно заменить, загрузив свой модуль.

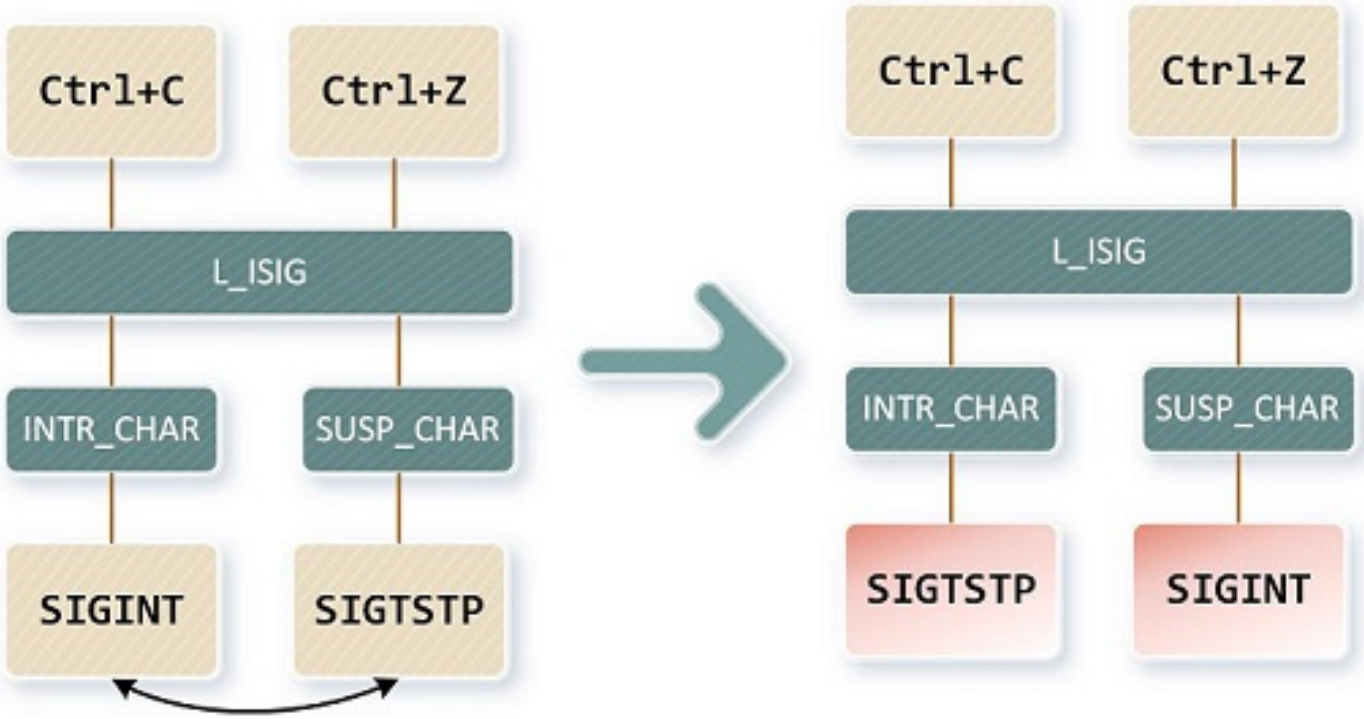
К сожалению, N_TTY сама по себе является частью ядра. Поэтому

за основу возьмем другие драйверы слоя LINE_DISCIPLINE, предусмотренные в Linux и загружаемые в виде модулей. По их образу и подобию модифицируем файл исходного кода n_tty.c:

1. Добавим функцию отгрузки модуля, в которой вызывается функция tty_register_ldisc, осуществляющая «знакомство» ядра с нашей персональной линией дисциплины. В эту функцию первым параметром передадим ее уникальный идентификатор, а вторым – указатель на структуру с методами драйвера.
2. Добавим функцию «выгрузки» модуля, в которой вызывается, соответственно, функция tty_unregister_ldisc.
3. В самой структуре «tty_ldisc_ops» зададим новое имя драйвера.
4. Позаботимся о том, чтобы наш модуль «узнал» нужные ему функции из файла tty_io.c (он не радует нас макросами «EXPORT SYMBOL», что заставляет либо дописывать все требуемые функции вручную, либо линковать вместе с tty_io.c).



Теперь добавим какой-нибудь функционал, отличающий нашу линию дисциплины от оригинальной. Помним, что именно TTY_LINE_DISCIPLINE обрабатывает служебные последовательности, поэтому грех не поколдовать на этом поприще. Для этого открываем функцию «n_tty_receive_char_special», в которой TTY_LINE_DISCIPLINE проверяет, не является ли введенные символы специальными и при нахождении оных посылает соответствующий сигнал. Для примера поменяем местами сигналы, генерирующиеся для Ctrl+Z и Ctrl+C:



После этого получим из нашего модифицированного файла непосредственно модуль ядра our_ldisc.ko. Загрузим его, убедимся, что загрузка произошла успешно. Проверим, что «our_modyfied_ldisc» действительно зарегистрировалась как TTY_LINE_DISCIPLINE. Откроем терминал и посмотрим номер pts. После этого назначим наш драйвер ответственным за слой TTY_LINE_DISCIPLINE у /dev/pts/X:

```
kseniia@ubuntu:~/Documents/test$ sudo insmod my_ldisc.ko
kseniia@ubuntu:~/Documents/test$ lsmod | grep my_ldisc
my_ldisc               32768  0
kseniia@ubuntu:~/Documents/test$ sudo cat /proc/tty/ldiscs
n_tty       0
our_modyfied_ldisc 26
kseniia@ubuntu:~/Documents/test$ ps
    PID TTY          TIME CMD
   5161 pts/18   00:00:00 bash
   5234 pts/18   00:00:00 ps
kseniia@ubuntu:~/Documents/test$ sudo ldattach 26 /dev/pts/18
```

Настроим новую линию дисциплины с помощью команды «stty echo cooked» — теперь терминал работает в привычном для нас режиме. Запустим тестовую программу с вечным циклом и

сравним эффект Ctrl+Z и Ctrl+C:

Мы добились желаемого: генерация сигналов переопределена на **уровне драйвера слоя** TTY_LINE_DISCIPLINE в индивидуальном порядке для одного эмулятора терминала! Есть поле для работы фантазии: от фокусов с обработкой служебных последовательностей до кастомизированного фильтра команд.

В ЗАКЛЮЧЕНИЕ

Теперь для тебя, %username%, тайны виртуальных консолей и эмуляторов терминала – больше **не** тайны, беспорядочная магия – **не** магия, а технология, прошедшая немалый путь, чтоб создать гибкую подсистему tty, а телетайп – **не** артефакт древности, а изобретение (кстати говоря, наше, отечественное), без потомков которого современный компьютер представлять как-то не хочется.

Мы любим рассказывать увлекательные истории. Хочешь послушать их вживую? Приходи на «Очную ставку» NeoQUEST-

2017, там тебя ждёт множество интересных докладов: от «железа» до криптографии! Вход свободный при регистрации на сайте.

Похожие публикации

Вам Telegramma: SPARQL-инъекции и CSRF через Telegram-сообщения в задании NeoQUEST-2016 2

NWOcs • 10 июня 2016 в 11:07

Полиция против мафии или занимательная статистика online-этапа NeoQUEST-2016 3

NWOcs • 29 марта 2016 в 13:09

Открыта регистрация на NeoQUEST-2016 17

NWOcs • 19 февраля 2016 в 13:08

Популярное за сутки

Яндекс открывает Алису для всех разработчиков. Платформа Яндекс.Диалоги (бета) 69

BarakAdama • вчера в 10:52

Почему следует игнорировать истории основателей успешных стартапов 20

ПЕРЕВОД

m1rko • вчера в 10:44

Как получить телефон (почти) любой красотки в Москве, или интересная особенность MT_FREE 24

ИЗ ПЕСОЧНИЦЫ

cab404 • вчера в 20:27

Java и Project Reactor 10

zealot_and_frenzy • вчера в 10:56

Пользовательские агрегатные и оконные функции в PostgreSQL и Oracle 6

erogov • вчера в 12:46

Лучшее на Geektimes

Как фермеры Дикого Запада организовали телефонную сеть на колючей проволоке

31

NAGru • вчера в 10:10

Энтузиаст сделал новую материнскую плату для ThinkPad X200s

49

alizar • вчера в 15:32

Кто-то посылает секс-игрушки с Amazon незнакомцам. Amazon не знает, как их остановить

85

Pochtoycom • вчера в 13:06

Илон Маск продолжает убеждать в необходимости создания колонии людей на Марсе

140

marks • вчера в 14:19

Дела шпионские (часть 1)

16

TashaFridrih • вчера в 13:16