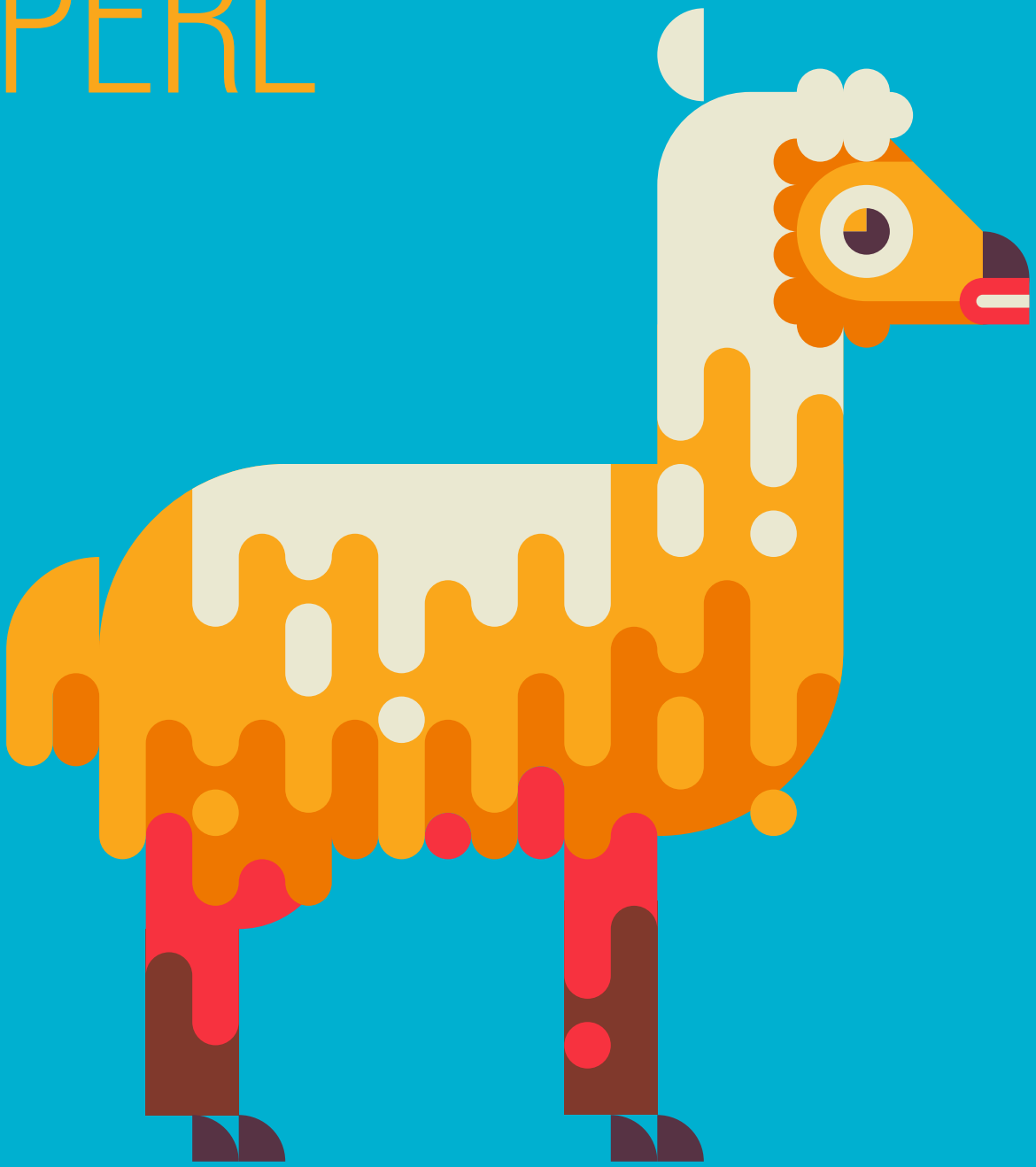
PRAGMATIC PERL

18



08/2014

pragmaticperl.com

Pragmatic Perl 18

pragmaticperl.com

Выпуск 18. Август 2014

Другие выпуски и форматы журнала всегда можно загрузить с pragmaticperl.com. С вопросами и предложениями пишите на почту editor@pragmaticperl.com.

Комментарии к каждой статье есть в html-версии. Подписаться на новые выпуски можно по ссылке pragmaticperl.com/subscribe.

Авторы статей: Андрей Шитов, Иван Бессарабов

Обложка: Марко Иваныk

Корректор: Андрей Шитов

Выпускающий редактор: Вячеслав Тихановский

Ревизия: 2014-11-29 16:23

Оглавление

1. От редактора

Лето, все в отпусках, но мы продолжаем выпускать журнал небольшим количеством авторов.

Приближается конференция YAPC::Europe 2014 в Софии. Она пройдет 22-24 августа. Если вы можете поехать, желаем отлично провести время! Если вы не можете поехать, следите за сообщениями на сайте мероприятия и в социальных сетях, обычно появляются либо видео, либо текстовые отчеты прямо во время конференции.

Мы продолжаем искать авторов для следующих номеров. Если у вас есть идеи или желание помочь, пожалуйста, свяжитесь с нами.

Приятного чтения.

■ *Вячеслав Тихановский*

2. Отчет о хакатоне 19-20 июля

Отчет о киевском Perl-хакатоне с приглашенным гостем

19-20 июля в Киеве прошел хакатон с участием Марка Леманна, автора таких модулей как JSON::XS, AnyEvent/EV, Coro и других. Событие было бесплатным, а спонсорами выступили две компании, занимающиеся разработкой на Perl: JB и WebbyLab.

С самого начала участники разбились на несколько групп: AnyEvent HTTP server, AnyEvent SMPP server, SockJS и доработка сайта киевской PM-группы http://kiev.pm.org.

У кого были вопросы во время разработки, имели уникальную возможность пообщаться с Марком Леманном, который как никто другой знает, как работает его софт. Кроме того, еще до начала активной фазы хакатона была проведена небольшая секция вопросов-ответов, где Марк высказал свою точку зрения на развитие Perl. В основном он опасается за сохранение обратной совместимости, что выгодно отличает Perl от других языков своей области.

В конце второго дня Марк выступил с небольшим докладом о своей новой разработке AnyEvent::Fork. Видео его выступления можно посмотреть на YouTube. Использование fork с AnyEvent крайне нежелательно, так как имеет множество побочных эффектов. Модуль Марка позволяет безопасно вызывать fork внутри AnyEvent-программ, а также просто обмениваться данными между процессами.

Два дня для меня как организатора пролетели незаметно. Надеюсь, что и участники получили удовольствие от общения и организованного хакинга. А Марк хочет приехать еще раз, так как не успел посмотреть на самую глубокую станцию метро в мире.

Следите за новостями и будущими мероприятиями!

■ *Вячеслав Тихановский*

3. Однострочники в Perl

Самый стандартный и часто используемый вариант написания кода на Perl — это создание текстового файла с кодом программы. Например, можно создать текстовый файл `hello.pl` вот с таким содержанием:

```
#!/usr/bin/perl

print "Hello\n";
```

И можно выполнить код этой программы в консоли:

```
$ perl hello.pl
Hello
```

Но иногда бывает очень удобно не писать отдельную программу в текстовом файле, а сразу написать в консоли в одну строчку весь код, который мы хотим выполнить. Такие команды называются однострончки (по английски — oneliner). Например:

```
$ perl -e 'print "Hello\n"'
Hello
```

Ключ -e и -E

Собственно говоря, ключ `-e` — это самый важный ключ для написания однострочников. Все что угодно можно сделать только с помощью этого ключа. Все остальные ключи были добавлены исключительно для упрощения написания однострочников.

Ключ `-e` — это выполнить код, который находится сразу после этого ключа. Например, сложить два числа:

```
$ perl -e 'print 2+5 . "\n"'
7
```

Чтобы не писать каждый раз символ для перевода на новую строку `\n` очень удобно использовать вместо оператора `print` оператор `say` — оператор `say` сам делает переход на новую строку, а еще `say` удобнее

писать, так как это слово короче чем слово `print`. Но для того, чтобы оператор `say` можно было использовать, нужно сказать:

```
$ perl -e 'use feature "say"; say 2+5'
7
```

Если не указать `use feature "say";`, то мы получим ошибку:

```
$ perl -e 'say 2+5'
Number found where operator expected at -e line 1, near "
    say 2"
        (Do you need to predeclare say?)
syntax error at -e line 1, near "say 2"
Execution of -e aborted due to compilation errors.
```

Но писать каждый раз `use feature "say";` слишком утомительно, поэтому появился новый ключ `-E`. Его можно использовать вместо `-e`, и он включит все новые штуки, которые есть в Perl:

```
$ perl -E 'say 2+5'
7
```

Ключ -M

Perl — замечательный язык, очень много в нем есть из коробки, но Perl очень силен тем, что у него есть куча библиотек. Например, вот однострочник, который скачивает веб-страницу, считает сколько на этой странице символов и выводит это число:

```
$ perl -E 'use LWP::Simple; say length get("http://ivan.
    bessarabov.ru")'
6096
```

Ключ `-M` позволяет упростить подключение библиотеки к однострочнику. Вот как выглядит эта же команда с ключом `-M`:

```
$ perl -MLWP::Simple -E 'say length get("http://ivan.
    bessarabov.ru")'
6096
```

Вот еще один пример. Хочу вывести десятый элемент последовательности Фибоначчи. Вот как выглядит однострочник без ключа `-M` (модулю `Math::Fibonacci` нужно ясно указывать, какие функции

нужно добавить в пространство имен):

```
$ perl -E 'use Math::Fibonacci qw(term); say term 10'
55
```

А вот более простой вариант этого однострочника с ключом `-M`:

```
$ perl -MMath::Fibonacci=term -E 'say term 10'
55
```

Ключ -n

Иногда хочется не просто что-то посчитать и вывести, а как-то обработать вывод других программ. И для этих целей однострочники великлепно подходят.

Например, у нас есть какая-то програма, которая создает некий вывод:

```
$ echo -e '1\n2\n3'
1
2
3
```

Мы хотим преобразовать этот вывод — возвести каждое число в квадрат. Для того, чтобы прочитать STDIN, в Perl используется бриллиантовый оператор (diamond operator) `<>`:

```
$ echo -e '1\n2\n3'| perl -E 'while (<>) { say $_ ** 2 }'
1
4
9
```

Для того, чтобы упростить написание подобных однострочников, был сделан ключ `-n`. При использовании этого ключа код, переданный в `-E` обрамляется:

```
LINE:
while (<>) {
    # your program goes here
}
```

И при использовании ключа `-n` однострочник выглядит так:

```
$ echo -e '1\n2\n3'| perl -nE 'say $_ ** 2'
1
4
9
```

butterfly operator

С ключом -n иногда используется специальный хакерский секретный оператор “бабочка” }{

Чуть изменим задачу из прошлого раздела. У нас есть скрипт, который создает вывод:

```
$ echo -e '1\n2\n3'
1
2
3
```

Нужно просумировать все числа из вывода этого скрипта.

Если бы мы решали эту задачу с помощью Perl-скрипта, то мы бы написали что-то вроде:

```
while (<>) {
    $sum += $_;
}

say $sum;
```

Мы хотим решить эту задачу с помощью однострочника. Ключ -n создает вот такой код:

```
LINE:
while (<>) {
    # your program goes here
}
```

Но нам нужно выполнить команду после того, как цикл отработает. Для этого и используется “butterfly operator”. Вообще, это никакой не оператор, а просто пара фигурных скобок. Первая фигурная скобка закрывает цикл, а вторая фигурная сбобка открывет новый блок, чтобы сохранить корректный синтаксис. Код получается таким:

```
LINE:
while (<>) {
    $sum += $_
}{
    say $sum
}
```

А однострочник выглядит вот так:

```
$ echo –e '1\n2\n3'| perl –nE '$sum += $_ }{ say $sum'
6
```

Ключ -a

Другая задача. Нужно обработать вывод команды, которая печатает на экран данные в виде таблицы:

```
$ echo –e 'a\t1\nb\t2\nc\t3'
a       1
b       2
c       3
```

Нужно отобразить все буквы из первого столбца, у которых цисло во втором столбце нечетное. То есть, должно быть:

```
$ echo –e 'a\t1\nb\t2\nc\t3' | perl –E ...
a
c
```

Вот длинный однострочник, который решает эту задачу. Разрезаем строку по пробелам и выводим первый элемент массива, если второй элемент массива — нечетное число:

```
$ echo –e 'a\t1\nb\t2\nc\t3' | perl –nE 'my @e = split /\
    s+/, $_; say $e[0] if $e[1] % 2'
a
c
```

Так писать, конечно, совсем не весело. Поэтому появился ключ -a, при использовании которого строка из дефолтной переменной разрезается на отдельные элементы, которые попадают в специальный массив @F. Вот как можно решить задачу с помощью ключа –a:

```
$ echo -e 'a\t1\nb\t2\nc\t3' | perl -naE 'say $F[0] if $F
    [1] % 2'
a
c
```

По дефолту `-a` разрезает строку по пробелам. Но с помощью ключа `-F` можно указать паттерн, по которому будет проводиться разрезание. Например. В файле `/etc/passwd` разделителем ялвяется доветочие:

```
$ cat /etc/passwd | head -n 2
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
```

Вот однострочник, который находит все User ID (колонка 3 в файле `/etc/passwd`), в которых есть цифра 1. Скрипт выводит логин (это колонка 1) и User ID.

```
$ cat /etc/passwd | perl -F: -naE 'say "$F[0] $F[2]" if
    $F[2] =~ /1/'
daemon 1
uucp 10
proxy 13
gnats 41
libuuid 100
syslog 101
messagebus 102
ntp 103
sshd 104
vagrant 1000
statd 105
bessarabov 1001
mysql 106
redis 107
elasticsearch 108
```

Ну и конечно паттерн, который передается в `-F`, может быть регулярным выражением. Например, есть скрипт который разделяет буквы всякой ерундой, нужно убрать ерунду и отобразить только буквы через пробел:

```
$ echo -e 'a---b\nc__d\ne=====f'
a---b
c__d
e=====f
```

Вот простое и понятное и очень аккуратное решение:

```
$ echo -e 'a--b\nc__d\ne=====f' | perl -F'/[=_-]+/' -naE
    'print "$F[0] $F[1]"'
a b
c d
e f
```

Резюме

Однострочники в Perl — это удобный способ очень быстро решить небольшую задачу. Для работы в консоли часто используют команды sed и awk. Но если человек умеет работать с Perl, то стоит использовать именно его.

В этой статье описаны самые базовые способы использования perl в консоли. Полное описание всех возможных способов запуска perl есть в документации perlrun.

■ *Иван Бессарабов*

4. Использование портов GPIO в Raspberry Pi. Часть 2

Во второй части статьи рассказано о новой модели B+ и о том, как при работе с Raspberry Pi добиться точности отсчета интервалов порядка одной микросекунды.

Новая модель Raspberry Pi B+

С момента выхода первой части этой статьи разработчики Rasbperry Pi анонсировали и начали продавать обновленную модель своего дивайса, Raspberry B+. В анонсе говорится, что это не новая, а именно обновленная модель. Несмотря на это, изменения очень приятные:

- вместо двух USB-портов теперь четыре;
- слот для SD-карты заменен слотом микро-SD (они называют это USD, видимо выбрав традиционное ASCII-написание греческой буквы mu);
- разъем видеовыхода совмещен со звуковым (кому вообще может потребоваться видеовыход при наличии HDMI?);
- всесто 26-контактного разъема с портами GPIO P1 теперь установлен 40-контактный;
- вместо двух отверстий для крепления платы теперь четыре :-)

В рамках темы этой статьи самое интересное изменение для нас — увеличение числа выводов GPIO. В новом 40-контактном разъеме J8, который теперь стоит на месте бывшего P1, доступны 26 портов GPIO вместо прежних 17. При этом с платы исчезли контакты для подключения разъема P5, хотя даже с учетом этой потери общее число доступных выводов все равно увеличилось.

Верхняя часть разъема полностью совпадает с разъемом P1 модели B, что позволяет без проблем заменить старое устройство и поставить на его место плату новой модели B+. Разводка нового разъема

J8 опубликована на сайте производителя, а новые доступные порты GPIO соответствуют выводам следующим образом:

1. GPIO05 — 29
2. GPIO06 — 31
3. GPIO12 — 32
4. GPIO13 — 33
5. GPIO16 — 36
6. GPIO19 — 35
7. GPIO20 — 38
8. GPIO21 — 40
9. GPIO26 — 37

Все перечисленные GPIO абсолютно новые, а выводы GPIO28—GPIO31, которые были выведены на исчезнувший разъем P5, теперь недоступны.

B+ и libbcm2835

Библиотека libbcm2835 и модуль Device::BCM2835, о которых рассказано в предыдущей части статьи, пользуются константами типа `RPI_V2_GPIO_P1_12`, содержащих в своих именах номер разъема (P1 или P5) и номер физического контакта (последние две цифры) соответствующего разъема. При программировании для Raspberry Pi B+ можно по-прежнему применять эти константы, но это неудобно.

Во-первых, они не позволяют обратиться к новым выводам. Констант типа `RPI_V2_GPIO_P1_37` не существует, а библиотека по-прежнему не обновлена, и на сегодня доступна версия 1.36, а на официальном форуме не видно никаких обсуждений по поводу того, будет ли это сделано.

Во-вторых, константы для раъема P5 более неактуальны.

Такое положение дел — не причина не использовать новые выводы, а наоборот хороший повод разобраться, что делать. Все константы

определены в заголовочном файле bcm2835.h следующим образом:

```
typedef enum
{
    . . .
    // RPi Version 2
    RPI_V2_GPIO_P1_03     =  2,  ///< Version 2, Pin P1
       −03
    RPI_V2_GPIO_P1_05     =  3,  ///< Version 2, Pin P1
       −05
    RPI_V2_GPIO_P1_07     =  4,  ///< Version 2, Pin P1
       −07
    . . .
} RPiGPIOPin;
```

Числовое значение констант — это не что иное как номер GPIO, поэтому вместо громоздких имен можно безболезненно подставлять числовые значения. Это не только облагородит код, но и немного уменьшит необходимость во время программирования бесконечного пересчета номеров GPIO в номера физических выводов на плате (хотя от этого не скрыться, когда дойдет дело до подключения реальных устройств).

Например, вместо строк (из которых непонятно, с каким GPIO ведется работа)

```
Device::BCM2835::gpio_set(RPI_V2_GPIO_P1_12); # Установка
    в 1
Device::BCM2835::gpio_clr(RPI_V2_GPIO_P1_12); # Сброс в 0
```

полностью безопасно и корректно писать так (где номер GPIO указан явно):

```
Device::BCM2835::gpio_set(18); # Установка в 1
Device::BCM2835::gpio_clr(18); # Сброс в 0
```

При таком подходе появляется возможность программно добраться до всех 26 GPIO, доступных в модели B+. Однако надо проявлять повышенную осторожность, поскольку теперь компилятор (что C, что Perl) не сможет проконтролировать вас, сверяя имена констант с предопределенным списком, и если указать номер несуществующего GPIO (или порта, который имеется в процессоре BCM2835, но не выведен на разъем Raspberry Pi) могут произойти непредвиденные побочные эффекты. Функции библиотеки libbcm2835 не делают

никакой проверки переданных аргументов. В моем случаи после попытки записи в порты с неверными номерами Raspberry Pi просто отключалась, к счастью, без физических повреждений.

Работа в реальном времени

В проекте, где я использую Raspberry Pi, мне потребовалось плавно изменять яркость лампы накаливания, причем такой регулятор должен управляться полностью программно и быть многоканальным. В интернетах такого толком ничего не описано: есть много ссылок о том, как регулировать яркость светодиода, парочка видео, где разработчик меняет яркость, передвигая ползунок на экране, но нет никакого описания, наконец, есть одна схема, работающая скорее с Arduino, чем с Raspberry, и одно смешное решение, где компьютер управляет моторчиком, который вращает регулятор яркости.

Кратко регуляторы яркости ламп накаливания (питающихся от сетевого переменного напряжения 220 В) работают так. Одна часть устройства следит за сетевым напряжением и формирует импульс в тот момент, когда оно переходит через нуль. Начиная с этого момента необходимо отсчитать определенное число миллисекунд и подать другой импульс, который включит симистор, который в свою очередь подаст на лампу напряжение. Как только напряжение вновь приблизится к нулю, симистор закроется (сам по себе), и лампа обесточится до следующего программного импульса. Все это происходит дважды за период сетевого напряжения. Его частота 50 Гц, так что управляющий цикл повторяется 100 раз в секунду, а его длина составляет 10 мс. Например, если задержку сделать в 3 мс, то лампа будет светиться примерно на две трети, а если 7 мс, то на четверть.

Казалось бы, при тактовой частоте Raspberry Pi 700 МГц проблем с программным формированием в 700 000 раз более низкочастотных интервалов быть не должно. На практике, однако, все не так. Регулятор яркости работает, но раз в одну-две секунды, а иногда и чаще, лампа помаргивает. Не помогает даже оверклокинг, максимально возможный до 1 ГГц. Я пробовал писать программу и на перле, и на C, но результат остался тем же.

Проблема заключается в том, что Raspberry Pi — это все-таки компьютер с операционной системой, где разрешены и используются прерывания. Работа с сетью, ввод и вывод и кто знает что еще, но в итоге прерывания задерживают работу цикла основной программы, и поэтому часть импульсов, которые должны включить лампу, пропускаются, а другая часть формируется в неправильное время.

Дальше можно пойти тремя путями. Во-первых, попробовать поставить иную операционную систему. Например, среди рекомендованных для Raspberry есть интересная система RISC OS, которая изначально проектировалась с учетом ограниченных возможностей оборудования, но она совсем не Unix-подобная, и как с ней работать, мне осталось неясным. Во-вторых, никто не мешает перейти на Arduino (хоть это и не спортивно) или на UDOO (это более интересно, и есть смысл попробовать). В-третьих, остается разобраться с тем, как избавиться от прерываний.

Регистры контроля прерываний

Контроль за работой прерываний в Raspberry Pi, а точнее, в процессоре BCM2835 описано в седьмой главе «Interrupts» мануала по периферии. Если не вдаваться в подробности, то имеется десяток регистров, доступных по адресам, начиная с 0x2000B000, со следующими смещениями:

1. 0x200 — IRQ basic pending
2. 0x204 — IRQ pending 1
3. 0x208 — IRQ pending 2
4. 0x20C — FIQ control
5. 0x210 — Enable IRQs 1
6. 0x214 — Enable IRQs 2
7. 0x218 — Enable Basic IRQs
8. 0x21C — Disable IRQs 1
9. 0x220 — Disable IRQs 2
10. 0x224 — Disable Basic IRQs

На странице Accurate timing for real time control показан пример кода на C, который отключив прерывания, формирует на одном из GPIO сигнал частотой 50 кГц. Этот обнадеживающий код я взял за основу и написал небольшую библиотеку libraspio, которая позволяет программно запрещать и разрешать прерывания. Как только прерывания перестали мешать, удается формировать интервалы времени с точностью до единиц микросекунд, что для моей исходной задачи более чем достаточно.

Вот так выглядит фрагмент функции `main()` моей итоговой программы, управляющей светом:

```
#ifndef DEBUG
    disable_interrupts();
#endif

    dimmer_loop();

#ifndef DEBUG
    enable_interrupts();
#endif
```

Да, это не перл, но если воспользоваться XS или Inline::C или просто обернуть библиотеку по типу того, как сделан модуль Device::BCM2835, все получится и на перле.

Отдельной строкой замечу, что позже я узнал и про то, что существует пара функций `local_irq_disable()` и `local_irq_disable()`, доступных практически из коробки, и наверняка есть смысл в следующий раз поэкспериментировать именно с ними.

Важно понимать, что при отключенных прерываниях выводы GPIO остаются полностью управляемыми и работоспособными. Поэтому их можно использовать не только, чтобы выводить сигналы на физические устройства, но и для того, чтобы считывать состояния любых кнопок или датчиков, подключенных к ним. Или даже подключить несколько разрядов GPIO ко второй плате Raspberry, на которой не отключены прерывания, и использовать шину GPIO для передачи команд на ведомую плату с отключенными прерываниями, работающей в режиме реального времени.

Поскольку прерывания более не доступны, временные задержки придется отсчитывать самостоятельно. Полагаться на функции типа `sleep` или `Device::BCM2835::delay` более нельзя: если их вызвать, то обратно не вернуться, и программа зависнет.

Как работать с регистрами на перле

Процессор BCM2835 содержит внутренний таймер, работающий на частоте 1 МГц, который доступен через регистры, отображаемые на память. В мануале его работа описана в главе 12 «System Timer». В отличие от регистров контроля прерываний, здесь все проще, поэтому я остановлюсь на этом подробнее.

Системное время (время, отсчитываемое с момента подачи питания), хранится в двух 32-разрядных регистрах CLO и CHI, видных, соответственно, по адресам 0x20003004 и 0x20003008. Два беззнаковых регистра обеспечивают 64 бита, что дает возможность отсчитывать время до 2^64 микросекунд, то есть на пятьсот с лишним тысяч лет (32 бит не хватит и на полтора часа).

Прочитать значения системных регистров относительно просто. Для этого в перле удобно воспользоваться функциями модуля Sys::Mmap (он, очевидно, использует POSIX-фукнцию mmap(2)):

```
1 my $devmemfh;
2 my $timer_CLO_reg;
3 my $timer_CHI_reg;
4
5 sysopen($devmemfh, "/dev/mem", O_RDWR|O_SYNC, 0666);
6 mmap($timer_CLO_reg, 4, PROT_READ, MAP_SHARED, $devmemfh,
      0x20003004);
7 mmap($timer_CHI_reg, 4, PROT_READ, MAP_SHARED, $devmemfh,
      0x20003008);
8 close($devmemfh);
```

Теперь переменные `$timer_CLO_reg` и `$timer_CHI_reg` всегда содержат значения из соответствующих регистров системного таймера (эти значения, изменяются без вмешательства программиста, поэтому последовательные чтения могут вернуть разные результаты — поведение, описываемое ключевым словом `volatile` в C, C# и Java).

Несмотря на кажущуюся избыточность, я использую все 64 бита, чтобы не получить непрятных сюрпризов в вычислениях после того, как заполнится весь младший регистр. Распаковать число типа `unsigned long`, сдвинуть и сложить, на перле это делается так:

```
my $timer_lo = unpack 'L', $timer_CLO_reg;
my $timer_hi = unpack 'L', $timer_CHI_reg;

my $timer = $timer_lo + ($timer_hi << 32);
```

Описанная функция реализована в модуле Device::BCM2835::Timer, его использование сводится к вызову функции `timer()`:

```
use Device::BCM2835::Timer;
say Device::BCM2835::Timer::timer();
```

Другие возможности

Когда не мешают ни прерывания, ни запущенная графическая оболочка, Raspberry Pi становится вполне быстрым и точным устройством, и работать с ним весьма приятно. Помимо описанного выше, GPIO предоставляют и другие интересные возможности для работы в реальном времени, например:

1. Формирование широтно-модулированного сигнала (PWM, pulse-width modulation) на выводе GPIO18. Но, к сожалению, такой выход только один, и сделать многоканальное устройство не получится.

2. Использование прямого доступа к памяти (DMA, direct memory access), что позволяет с высокой точностью пересылать на выход (опять же, один-единственный) данные, предварительно записанные в некоторую область памяти. В частности, так можно реализовать проигрывание звукового файла в обход процессора.

3. Самый необычный проект, использующий DMA, — pifm. Это FM-передатчик на Raspberry Pi. Программным путем генерируется сигнал на частоте до 200 МГц, который принимается обычным УКВ-приемником:

```
sudo ./pifm sound.wav 100.0
```

Одно из последних обновлений этого проекта (сайт у них давно сломан) позволяло выводить в эфир стереозвук (сам по себе аудиовыход у Raspberry Pi монофонический).

■ *Андрей Шитов*

5. Обзор CPAN за июль 2014 г.

Рубрика с обзором интересных новинок CPAN за прошедший месяц.

Статистика

- Новых дистрибутивов — 219
- Новых выпусков — 867

Новые модули

- Hash::Ordered

Модуль позволяет работать с Perl-хешами как отсортированными структурами.

- Form::Diva

Модуль позволяет программно генерировать HTML5-формы. В отличие от других генераторов форм он позволяет только отдельно генерировать поля, таким образом модуль удобно использовать в шаблонах с помощью хелперов, сохраняя гибкость ручной верстки.

- Type::Tiny::XS

XS-версия модуля `Type::Tiny`, которая ожидаемо работает существенно быстрее.

- Web::Compare

Модуль позволяет сравнивать web-страницы и генерировать отчет, в качестве модуля сравнения можно подставить различные модули из семейства `diff`.

- App::plackbench

Как можно понять из названия, модуль позволяет тестировать производительность Plack-приложений, просто указав на `.psgi`-файл, без запуска http-сервера.

- Pod::Markdown::Github

Преобразование POD-файла в Markdown, принимаемый сервисом GitHub.

- Code::DRY

Модуль позволяет обнаруживать участки копипасты в Perl-коде. Полезен для проектов, где применяется DRY.

- Getopt::Long

Автор утверждает, что если заменить `Getopt::Long` в программе на `Getopt::Long::Complete`, то она сразу же будет поддерживать автодополнение опций. Нужно не забыть выполнить еще некоторые манипуляции в bash-консоли.

Обновлённые модули

- URI 1.64

Несколько обновлений после двухлетнего перерыва. Закрыты старые RT-баги, некоторые из них связаны с UTF-8.

- LWP 6.08

Исправления, связанные с поддержкой IPv6.

- Storable 2.51

После года перерыва несколько исправлений, связанных с утечкой памяти при клонировании, а также некорректным поведением при уничтожении объектов.

- Proc::Pidfile 1.06

Переход на `Dist::Zilla`, прекращение использования `Proc::ProcessTable` кроме некоторых платформ.

- Rex 0.51.1

Множество небольших изменений, поддержка FreeBSD 10 и Cent OS 7.

- HTTP::Tiny 0.047

Поддержка пустых заголовков, обновление `Mozilla::CA`, мелкие исправления в тестах.

- DBIx::Class 0.082700_04

Релиз для разработчиков. Множество исправлений.

■ *Вячеслав Тихановский*

6. Интервью с Дмитрием Карасиком

Дмитрий Карасик (karasik) — Perl-программист, автор IO::Lambda и Prima.

Как и когда научился программировать?

В 1988-м году на Электронике МК-54. Виноват в этом журнал “Наука и жизнь”, который публиковал статьи о программируемых калькуляторах, который, огромное спасибо родителям, мне подарили. Когда журнал стал публиковать программы на Basic для PC, мне намекнули, что семейный бюджет такого вложения не выдержит, но по счастью в университете был доступ к PC-совместимым ЕС-1841, а там уже был DOS, Turbo Pascal, и пошло-поехало.

Какой редактор используешь?

vim

Как и когда познакомился с Perl?

В 1997-м Антон Березин пригласил меня в проект на базе лаборатории, занимающейся молекулярной биологией в Копенгагенском университете. Он искал подходящий язык для макропрограммирования и остановил выбор на перле. Рабочие станции были на OS/2, Windows 95 и FreeBSD, и нам нужно было сначала сделать графическую обвязку для перла - отсюда вырос GUI-тулкит Prima. На нем мы делали клиентские программы для исследовательских нужд, что включало в себя в основном программирование собственно GUI, но с приятной экзотикой вроде обработки изображений, автоматического управления видеокамерой, алгоритмов распознавания и тому подобное. Только гораздо позже я открыл для себя перл как инструмент для веба.

С какими другими языками интересно работать?

Если именно работать, то вот как-то ни с какими, если честно. Профессионально я сильно завязан на перл, C и javascript, а так конечно держусь в курсе того что происходит в мире. По мелочи писал или

просто пробовал силы на c++, java, php, ruby, python, erlang, haskell, lisp, go … ну как сказать, языки и языки. Особо сильного энтузиазма не испытываю ;)

Что, по-твоему, является самым большим преимуществом Perl?

Качественные библиотеки (т.е. CPAN), которые не нужно постоянно мониторить на предмет багов. Community — доброжелательные люди, с которыми можно внятно общаться. То же, что и раньше в общем. Мне лично импонирует некая доля юмора в перловой тусовке; вот только вчера с коллегами вспоминали про Acme::Code::Police который стирает файл с модулем, не испльзующий strict. И Acme::Code::FreedomFighter который стирает Acme::Code::Police))))) Ах да, юникод же еще. Отличная поддержка юникода.

Что, по-твоему, является самой важной особенностью языков будущего?

Опенсорс, я так думаю; причем не только открытый исходный код в режиме read-only, но и принадлежность самой культуре — с форками, патчами, интеграцией с существующими библиотеками. Качественная абстракция над сложными вещами вроде многопоточности, корутин (как в go), интроспекции самого языка. Ненавязчивость (ОО — привет Java, пробелы — привет python). Здоровый минимализм (например js).

Что думаешь о будущем Perl?

Думаю, что оно есть и никуда не денется, но поуменьшится в общей доле вследствие разных причин. Молодым профессионалам нужно строить свое портфолио, показывать миру свои идеи и библиотеки, а в перле все “низко висящие фрукты” в основном собраны. Я думаю, в будущем мы будем видеть все меньше новых громких проектов на перле, но не потому, что их число будет уменьшаться, а потому что их “громкость” не будет приоритетна. Примерно как, ну скажем, водопроводные трубы — вещь нужная, но ни разу не хипстерская)

Чем IO::Lambda отличается от других event loop? Как IO::Lambda уживается с AnyEvent?

IO::Lambda это конечно проект амбициозный но он, увы, не смог заработать широкой поддержки. Его отличие наблюдается только на достаточно высоком уровне, когда нужно описывать много, не одну-две, а именно много машин состояний, и возникает необходимость, чтобы их код был собран локально, а не размазан по отдельным функциям. Возможно именно из-за узкой специализации.

Эту же задачу решают корутины (и многопоточность в какой-то мере), но в перле с корутинами туговато, я считаю главным образом из-за того, что автор Coro не смог наладить эффективное общение с группой разработчиков перла, так чтобы Coro нормально дружил с языком. С AnyEvent модуль уживается абсолютно нормально и может его использовать как back-end, кроме многопоточных программ, где сам AnyEvent не работает.

Расскажи про Prima. Насколько успешно можно применять Perl для графических программ?

В двух словах — это GUI специально для перла. Вполне успешно можно применять на windows и unix, выглядит на обоих практически идентично. Там даже есть visual builder чтобы рисовать простые интерфейсы.

Этому проекту я и сам удивляюсь — первый код был написан в 97-м году, я до сих пор изредка в него коммичу, в основном по запросам извне. Наверное это больше говорит обо мне, чем о проекте ;) На сегодня с ним связаны несколько побочных проектов, которые тоже веду я, в основном для интеграции с другими тулкитами — с ImageMagick, PDL, Cairo, OpenGL. PDL-девелоперы рассматривают возможность перейти с gnuplot на Prima для визуализации графиков — посмотрим, будет ли желание этим заняться у кого-нибудь. Toby Inkster вот сделал год назад Data::Dumper::GUI.

Проект живет себе, эгосёрфинг иногда приносит приятные вещи вроде того, что вот, люди его используют и даже отвечают на вопросы на perlmonks. Как-то я прочитал комментарий на CPANTESTs, что мол не верится, что у гуи-тулкита есть автоматические тесты, и что он их всех проходит на всех тестовых платформах без ошибок :) но это старый проект, и многие вещи в нем хорошо отполированы.

В приме есть маленькая пасхалка — название проекта означает “первый”, но в visual builder’е до сих пор используется иконка, похожая на папиросы “Прима” — кто помнит, да ;) Вот думаю, убирать или не убирать, серьезный проект все-таки :)

Как натыкаешься на нестандартные применения стандартных особенностей языка, как, например, модуль Image::Match?

С Image::Match была история — нам нужно было сделать автоматизацию для клиента на Win32::GuiTest, когда программа сама нажимает кнопки и пишет текст и тому подобное. Проблема возникла, когда автоматизировать пришлось удаленного клиента через remote desktop и использовать поиск окон и кнопок и прочего не было возможности через WINAPI, единственная возможность была анализировать графический вывод. Т.е. находить скриншот на экране и посылать туда сообщение от мыши. И вот я сижу думаю, какой у нас алгоритм поиска скриншота? Берем изображение из файла, на котором например скриншот нужной кнопки. Читаем оттуда первую скан-линию, т.е. строку байтов, представляющих набор горизонтально расположенных подряд пикселей, и ищем эту строку в скриншоте всего экрана. Если нашли, значит вторая скан-линия должна находиться на сколько-там байтов дальше, на ширину экрана минус ширина кнопки… потом третья и так далее. А если третья строка не совпадает? Тогда откатываемся к самой первой линии и ищем дальше — это backtracking. Который отлично работает в регулярных выражениях! А значит поиск это всего лишь:

```
1 $screenshot =~ /$scanlines[0].{$width}$scanlines[1].{
    $width} ... /
```

и т.д. до конца…

Первый баг там был интересный — модуль на одной машине заработал, а на другой нет. Выяснилось, что на одной иконке был пискель со значением `0x0A`, и регексп, в который я забыл поставить `/s`, рассматривал его как перевод строки и не работал :)

Где сейчас работаешь? Сколько времени проводишь за написанием Perl-кода?

Сейчас я работаю в research & development в биохимической ком-

пании Novozymes. Мы поддерживаем исследовательский отдел на связке linux, postgres, apache, perl, catalyst — несколько внутренних сайтов, базы, исследовательские инструменты на базе Bio::Perl. Моя работа это в основном back-end на перле и поменьше front-end на js/html. Ну и связанные вещи — баг-фиксинг, базы, апдейты, по мелочи. Кстати, прямо сейчас набираем девелоперов (дедлайн 24 августа).

Стоит ли советовать молодым программистам учить сейчас Perl?

Конечно стоит, кто ж новые проекты будет на перле тогда делать? :D

Вопросы от читателей

Не надоел Perl?

Ну разве самую малость. И то, не сколько перл, сколько программирование и компьютеры в целом.

Посещаешь ли Perl-мероприятия? Каковы, по-твоему, основные отличия конференций за рубежом от СНГ?

Да, стараюсь попадать на YAPC::EU и Nordic Workshop. Или по возможности на русскоязычную конференцию.

Отличия? Теперь уже и нет почти этих отличий, не считая языка, мне так кажется. Доклады на ::EU интересней, так как охватывают больше людей, да вот и только.

Как найти работу за рубежом?

Как и везде — на job-сайтах. Вот расскажу про датские реалии например, которые мне хорошо известны:

1) гуглите denmark job sites, попадаете например на jobindex.dk, там вбиваете в поиск perl и читаете анонсы потенциальных работодателей.

2) выбираете то что интересно, посылаете cv с коротким cover-letter.

3) ждете ответа, если вами заинтересуются — пригласят. Если нет, то хорошие компании пришлют отказ, плохие — нет… да и то, в плохих компаниях работать вредно для здоровья.

4) PROFIT‼!

Тонкий момент — качественное составление cv и cover-letter. Это важный навык, если не умеете — проконсультируйтесь у специалиста, серьезно. Потому что когда нужно прошерстить сотню заявлений (чем я вот прямо сейчас занимаюсь), плохо (слишком длинно, слишком коротко, без фото, .doc, не на английском, трудно читать, непонятно, сколько опыта на перле) оформленная заявка это не шанс на победу, а совсем наоборот.

Ну и английский нужно знать, конечно.

Это активный путь, есть еще и пассивный — распространить инфу всем знакомым за рубежом, что вы ищете работу по такой-то специальности. Есть небольшой шанс, что вас порекомендуют.

Почему не развивается Укроп?

Да ну, это шутка была ;) Я в то лето прочитал слишком много рассказов Борхеса и раздухарился на подражание с уклоном в компьютеры. А тут как раз воркшоп в Киеве, ну грех было не сделать лайтнинг перла на суржике :)

■ *Вячеслав Тихановский*