PRAGMATIC PERL

17

07/2014

pragmaticperl.com

Pragmatic Perl 17

pragmaticperl.com

Выпуск 17. Июль 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, Владимир Леттиев, Елена Большакова, Александр Ружников, Тимур Нозадзе

Обложка: Марко Иваныk

Корректор: Андрей Шитов

Выпускающий редактор: Вячеслав Тихановский

Ревизия: 2014-11-29 16:22

Оглавление

1 От редактора

19-20 июля в Киеве пройдет хакатон/встреча. В качестве приглашенного гостя — Марк Леманн (Marc Lehmann), автор таких модулей как `JSON::XS`, `AnyEvent/EV`, `Coro` и многих других. Участие бесплатное, однако требуется обязательная регистрация, так как число мест ограничено.

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 конференции YAPC::Russia 2014 от организатора

YAPC::Russia — ежегодная конференция для Perl-программистов, одна из семейства мероприятий YAPC, известных интересными участниками и замечательной неформальной атмосферой. Несмотря на название, конференция имеет некий элемент международности, и проводится исторически попеременно то в Москве, то в Киеве.

Такой же план был и в этом году — ещё в декабре на конференции Saint Perl Андрей Шитов объявил, что YAPC::Russia 2014 пройдёт в середине июня в Киеве. Однако, непростая политическая обстановка поставила такой вариант под сомнение, и было объявлено о переносе конференции в Москву. И тут был высказан вполне

резонный вопрос: «А что, кроме Киева и Москвы и городов больше нет? Почему, например, не Питер?» Тем более, середина июня — разгар белых ночей, и, пожалуй, лучшее время для посещения Северной столицы. Идея была всячески поддержана, и это случилось — конференция YAPC::Russia 2014 впервые была проведена в Санкт-Петербурге, 13-14 июня.

Генеральным спонсором (а также и всеми остальными спонсорами и партнёрами) конференции в этом году выступил Регистратор доменных имён REG.RU — одна из немногих компаний в России, продолжающих активно поддерживать Perl-сообщество. За что ей спасибо.

Мероприятие традиционно началось на день раньше официальной даты — вечером перед конференцией те участники, кто

уже прибыл и рвался в бой, встретились на препати-вечеринке, где смогли приятно провести время и перезнакомиться, кто ещё не знаком. (Согласно древней легенде, они вынуждены делать это каждый год заново, потому что утром всё равно никто никого уже не узнаёт.) Уютное общение за кружечкой пива или томатного сока продолжилось на афтепати после первого дня конференции. Однако, большинство фотографий и других свидетельств с этих мероприятий засекречено ввиду глубокой аполитичности.

Конференция проходила на территории площадки «Место Роста», где мы, к нашей большой удаче, смогли найти оптимально подходящий нам зал со всем нужным оборудованием, к тому же за очень скромную сумму (что для бесплатной конференции тоже немаловажно). На входе участники

сразу получали футболки с логотипом конференции и целый мешок подарков от спонсора.

Компания REG.RU в этом году не только выступила генеральным спонсором конференции, но и обеспечила существенную часть информационного наполнения. Программисты компании рассказали о задачах, с которыми они сталкиваются при работе над крупной программной платформой, обеспечивающей функционирование всего бизнеса и построенной практически полностью на Perl, таких как перевод большой кодовой базы на Unicode, механизм локализации и т. п., об инструментах, которые используются в компании (от поиска на Sphinx до платформы видеоконференций). Нашлось место и для более общетеоретических докладов (про статический анализ Perl-кода с помощью PPI, использование

Coro и целой лекции про асинхронное программирование). Остальные члены российского Perl-сообщества, честно говоря, в этом году несколько поленились, но тоже представили несколько интересных докладов. (И не думайте отсидеться в следующем году. За вами уже выехали.) Не обошлось и без традиционных сессий блиц-докладов с местами вполне эзотерическими темами и даже бесплатной раздачей алкоголя (финальный выстрел в голову тем, кто не приехал).

Большая часть выступлений была записана на видео. Записи нескольких докладов уже выложены на YouTube. В описании видео можно найти ссылки на слайды презентаций. Ссылки на презентации и видео есть и на сайте конференции. Видео и презентации остальных докладов мы надеемся подготовить и выложить в ближайшее время,

следите за новостями.

Участникам конференции было чем развлечься и помимо докладов. Например, постепенно становящийся уже традиционным конкурс Perl Golf дал всем желающим хороший повод пошевелить извилинами и припомнить пару магических трюков из арсенала Perl. Ходят слухи, что некоторые увлеклись решением настолько, что это пошло во вред количеству выпитого на афтепати пива. Но это, скорее всего, правдой быть не может. Задание на конкурс (а это, ни много ни мало, подсчёт небезызвестного числа Эрдёша) опубликовано на Гитхабе, так что все желающие могут попробовать свои силы. А в скором будущем в репозитории появятся и решения от участников. Победитель конкурса получил отличный набор для офисного мини-гольфа. К тому же, каждому участнику достались всякие

приятные штуки от спонсора.

На спонсорском стенде тоже было весело — ребята (и, конечно, девчата) из REG.RU провели викторину на знание Perl и его истории и соревнования по прицельному бомбометанию фрисби. Ничто так не расслабляет в кофе-брейк, как вкусные печеньки и возможность покидать что-нибудь по комнате, полной людей и цветочных горшков. Кроме того, была роздана тонна футболок, браслетов, наклеек и других симпатичных вещичек от REG.RU и — гвоздь программы! — большая порция перловой атрибутики, специально приехавшей из Нидерландов от Wendy & Liz: значки, наклейки и всеми горячо любимые «туиты» (по секрету: чуть-чуть туитов осталось, обращайтесь, договоримся).

В общем, мероприятие удалось. Собрались

отличные ребята и отлично провели время. Те, кто не был, могут оценить, насколько всё было здорово по рассказам очевидцев и фотокарточкам с места события и https://vk.com/album798351_198161004.

Ну и ждём всех на YAPC::Russia 2015!

■ *Тимур Нозадзе*

3 Еще один отчет о конференции YAPC::Russia 2014

С 13 по 14 июня 2014 в городе Санкт-Петербурге прошла очередная конференция «YAPC::Russia». Это был первый случай, когда в городе на Неве проводилось данное мероприятие, обычно конференция проводилась либо в Москве, либо в Киеве. Основную часть организации выполнил Тимур Нозадзе, основным спонсором выступили «РЕГ.РУ». Сразу хочется сказать, что конференция удалась на славу — было представлено в общей сложности 30 докладов на самые различные темы от юникода до установки перла на iOS.

В первый день было целых два доклада про асинхронность и параллелизм — доклад про `Coro` и большой рассказ про асин-

хронное программирование в перл. Елена Большакова рассказала, как в «Яндекс» деполятся перловые проекты, Полина Шубина из «РЕГ.РУ» рассказала про использование разметки Markdown, вскользь упомянув загадочный язык «г*внокод»:-). Также прозвучали доклады про избавление от циклических ссылок и написание GUI-интерфейса в виде локально работающего сайта.

Первый день конференции также запомнился тем, что в фойе расположилась группа пропагандистов здоровой еды и вегетарианства. Они каждому предлагали бесплатно попробовать экзотические фрукты и посетить их семинары йоги. Ребята немного не от мира сего, но добродушные и щедрые — нам подарили спелый плод манго.

Завершился первый день конференции посиделками в баре недалеко от места проведения. Там собралась бОльшая часть участников, самые стойкие сидели и обсуждали насущные вопросы до самого закрытия бара.

Второй день конференции открыл Андрей Нуггед рассказом о том, как запустить перловый интерпретатор на айфоне. Дальше был интересный доклад Виктора Ефимова про юникод в перле. Тема актуальная и мало кем из разработчиков глубоко изученная (сам недавно столкнулся с проблемой юникода в модуле `Net::LDAP`). Дальше шли доклады про использование `Vagrant`, мой рассказ про использование системы видеоконференции `BigBlueButton`, доклад про поиск в `OTRS` с помощью `Sphinx` и доклад про использование перла в работе с сетевым оборудованием по протоколу

SNMP. Виктор Ефимов вновь вышел на сцену и рассказал про модуль `PPI`, затем Игорь Миронов и Акжан Абдулин рассказали про то, как можно (а может и нужно) делать интернационализацию сайта. Первую половину доклада Игорь рассказывал на сцене, а Акжан сидел за кулисой и оттуда дополнял рассказ, со стороны выглядело слегка комично. Вторую половину доклада оба уже рассказывали стоя на сцене. Последним большим докладом второго дня был рассказ Антона Герасимова про опыт использования модуля `Dist::Zilla`.

Оба дня конференции по традиции завершались блиц-докладами по 5 минут каждый. Также во время конференции проходил перл-гольф — программисты выясняли, у кого короче код получается. Победители получили памятные призы.

Всё время проведения конференции в соседнем малом зале сотрудники и сотрудницы компании «РЕГ.РУ» рассказывали о компании, дарили подарки и проводили викторины на знание тонкостей перла и принципов его работы.

От себя лично хочу выразить благодарность всем участникам. Было очень интересно и занимательно. Надеюсь увидеться на YAPC::Russia 2015!

■ *Александр Ружников*

4 Яндекс.Директ: как мы деплоим наши Perl-web-приложения

Это примерная расшифровка моего доклада на YAPC::Russia 2014, аннотацию см. на сайте конференции.

Привет!

У меня будет «нестандартное» выступление: обычный доклад – это «какую крутую штуку мы сделали», или «какие крутые штуки можно сделать», или «какие крутые библиотеки можно использовать, чтобы делать еще более крутые штуки», а я собираюсь рассказать, что происходит у нас *после* того, как мы все это использовали и запрограммировали.

Далее описываются слайды доклада.

Слайды 3, 4, 5

Немного контекста: кто это – «мы»?

Это Яндекс.

На Яндексе есть реклама.

Чтобы реклама была, нужен интерфейс, в котором рекламодатель ее создаст.

Слайд 7

Может показаться, что это плевое дело – интерфейс для приема рекламных материалов. Однако даже принять у пользователя объявления не так легко, если их ОЧЕНЬ много. Кроме того, есть несколько способов рекламу заливать и управлять ею (API, web-

интерфейс, десктопное приложение, ...) И категорий пользователей тоже несколько: рекламодатели, рекламные агентства, менеджеры по продажам и другие, так что возникает сложная система ролей. Помимо собственно создания объявлений есть еще прогнозирование рекламных компаний, всяческие проверки, оптимизации, оплаты, отправка в показывающую часть, предоставление рекламодателям разнообразной статистики, внутренние отчеты и т.д. Так что данных у нас много, запросов много, общения со смежными сервисами тоже много, функциональность обширная, разработка активная.

Слайды 8-9

О технологиях. У нас более или менее LAMP: живем на Ubuntu; Apache исполь-

зуем, Nginx и Starman – тоже; данные храним в основном в mysql, есть немного mongodb и для тяжелой аналитики YT – Яндексовый map-reduce[1]; много кода на Perl, есть серверный javascript, немного Python. Для очередей задач у нас есть Gearman (не очень нравится), немного используем Ubic, у нас есть SOAP (не нравится, но приходится), xmlrpc, jsonrpc (нравится).

Слайд 10

Среды у нас делятся на разработческие, тестовые и продакшен. На разработческих — упрощенная инфраструктура (apache запускается прямо над рабочей копией), тестовые среды по возможности точно

[1]Про YT был доклад на YAC 2013: http://tech.yandex.ru/events/yac/2013/talks/1091

воспроизводят продакшен. В качестве разработческих и тестовых баз данных используем обезличенные бекапы продакшена. Это очень полезная практика, так как заодно регулярно проверяется пригодность бекапов к восстановлению. Кроме того, настоящая база – отличный источник готовых заковыристых сочетаний данных для тестирования.

Возможно, неочевидный факт: акции Яндекса размещаютя на бирже NASDAQ[2], поэтому мы обязаны выполнять все их требования, касающиеся публичных кампаний[3]. Например, доступы к разным средам разделены, продакшен обновляют и обслуживают только админы, не разработчики. На релизы есть подробный строгий регламент.

---

[2]http://www.nasdaq.com/symbol/yndx

[3]См. SOX, SOX 404 top–down risk assessment

Слайд 12

Для управления серверами (выкладок и прочего) есть собственная утилита, похожая на Rex[4]. Работает поверх ssh, на управляемых машинах никаких специальных агентов не требуется. То есть для того, чтобы выложить релиз, админу не надо заходить на все серверы, достаточно выполнить нужные команды с центральной машины.

Опенсорсить утилиту не собираемся, в ней слишком много проектной специфики.

---

[4] http://www.rexify.org

Слайд 13

Релизы у нас живут в таск-трекере. Релиз-менеджер записывает, что и как выкладываем, тестировщики пишут о найденных проблемах, там же потом собираются все нужные подтверждения – от QA, от заказчика, от релиз-менеджера.

Каждый релиз тестируется на тестовой среде. Проверяется новая функциональность и то, что старая не сломалась («регрессия»). Для changelog'а берем svn log от прошлого релиза до текущего head'а. Для транковых коммитов в коммит-сообщении обязательно есть ссылка на таск-трекер, на тикет, который этим коммитом решается. Это тоже очень хорошая практика: коммиты в транк связаны с перепиской в трекере, так что даже спустя несколько лет можно узнать, какие обсуждения предшествовали

внесению определенного изменения, какие были проблемы при тестировании и т.п.

Слайд 14

Код выкладываем deb-пакетами. Это работает, это универсально, позволяет работать с зависимостями, в том числе и с неперловыми. Много готовых инструментов: и для сборки пакетов, и для поддержания своего репозитория, и для чего угодно.

Есть неудобья: apt плохо приспособлен к тому, чтобы устанавливать не самую новую версию пакета, имеющуюся в репзитории. А если с выкладкой что-то пошло не так, и надо вернуть все «как было», то это нетривиальная задача.

Слайд 15

Транк у нас «зеленый», то есть туда попадает уже проверенный и проревьюенный код. Пакетируется для выкладки либо транк, либо релизный бранч, который отводится от транка и в который, если требуется, домерживаются отдельные хотфиксы.

Версии присваиваются так, что по ним однозначно понятно, из какой точки в истории репозитория пакеты были собраны.

Ведется лог: когда на какой сервер выкладывались какие версии каких пакетов. Доступен этот лог в виде небольшого веб-сервиса с фильтрациями, отбором, сортировками и т.п. В частности, такой лог помогает, если надо восстановить на сервере старое состояние пакетов.

Кстати, зависимости лучше указывать с ‘>=’, а не с точными версиями – так, по крайней мере, apt не будет ломаться при откате на предыдущую версию приложения.

Очень полезная практика — заранее проверять пригодность зависимостей к установке. Если этого не делать, получаются неприятные неожиданности в момент, когда релиз надо выкладывать на ТС. У нас работает проверка после каждого коммита в транк, и в случае чего разработчик может быстро все поправить.

Опасные и/или сложнооткатываемые обновления полезно делать отдельно от прочих релизов — чтобы проблемы, если они появятся, не аффектили бы другие задачи.

Слайды 16, 17, 18

Помимо выкладки на серверы нового кода в релизе могут требоваться другие действия: изменения структуры таблиц в БД (DDL-запросы: alter'ы, create'ы, drop'ы), конвертации данных (запуск скриптов), нестандартные ручные действия (специфические обновления, рестарты и т.п.).

Все такие действия, которые требуется выполнить в дополнение к доставке на серверы нового кода, мы называем миграциями.

Работа с миграциями у нас устроена похоже на `django`-вый `south`, на `dbdeploy`, да это и закономерно, я думаю.

Миграции описываются в файлах специального формата, которые коммитятся в репозиторий вместе с кодом. Все миграции

хранятся в каталоге /deploy, пишут их разработчики параллельно с написанием кода. Формат — перловые структуры, хеши и массивы. Это с одной стороны достаточно формально и пригодно к валидации, с другой — человекочитаемо.

В миграции указывается тип (выполнение sql-запроса, запуск скрипта или другие действия), когда надо ее применить (до выкладки кода, после или не имеет значения), оценка длительности выполнения и если требуется — комментарии об особенностях выполнения и отмены.

Применяются миграции полуавтоматически на ТС и вручную в продакшене. Пока не хватает автоматического определения примененных миграций и возможности запускать простые скрипты без участия человека.

Поскольку миграции пишутся в виде перловых хешей и массивов, парсить их можно простыми eval'ами. Для безопасности (чтобы в миграцию нельзя было вписать выполнение внешнего кода), это делается через `Safe::reval`, в котором запрещены все операции, кроме самых простых.

Миграции обязательно ревьюятся и апрувятся. Это важно, так как миграции – это потенциально очень опасные изменения, да к тому же и сложно поддаются классическому тестированию. Менять структуру БД — тяжело, гораздо тяжелее, чем рефакторить код, поэтому схема данных заслуживает особо тщательного проектирования.

Кроме того, автоматически проверяется синтаксис миграций, наличие всех

нужных полей и отсутствие ненужных. Синтаксис sql-запросов проверяется с помощью `DBIx::MyParsePP`, и это очень полезно.

Чтобы не запутаться в названиях полей, есть специальный скрипт-хелпер. Он задает серию вопросов (одно действие или несколько? Sql-запрос или скрипт? До кода или после?) и генерирует шаблон миграции, в который остается вписать свои фактические запросы и другие данные.

Слайд 21

Для удобного хранения истории изменения структуры БД и для документирования таблиц и полей мы используем следующую систему: в каталоге `/db_schema` для каждой таблицы хранится файлы

`table_name.schema.sql` и `table_name.text`. В первом хранится запрос create table, которым можно создать таблицу, а во втором — описание таблицы в целом и ее полей по отдельности.

Есть скрипт, который показывает расхождения между файлами `.schema.sql` и реальной базой, отчет о таких расхождениях в продакшене регулярно приходит на общую рассылку.

В итоге получается удобно: хочешь узнать, какой смысл у поля в таблице — читаешь описание. Хочешь узнать, когда поле в таблице появилось — смотришь `svn annotate` на файл `.schema.sql`.

Слайд 22

Регулярные запуски скриптов делаются через `cron`. Кронтабы генерируются при сборке пакетов из специальных pod-секций в самих скриптах, и с пакетами устанавливаются в `/etc/cron.d` одновременно с установкой основного кода.

Слайды 24, 25

Кроме тестов, проверяющих поведние кода, есть еще много разных проверок, которые полезно регулярно запускать в проекте. На слайдах выписаны самые полезные на мой взгляд — те, что срабатывают чаще всего или ловят ошибки, которые в другом случае могли бы долго и незаметно портить работу продакшена.

Слайд 26

Вот и всё. Если прочитали что-то новое и полезное — пожалуйста, пользуйтесь. Если умеете что-то делать удобнее и надежнее — расскажите.

■ *Елена Большакова*

5 Асинхронный ввод/вывод с IO::AIO

В UNIX-системах операции чтения и записи файлов, как правило, являются синхронными, т.е. вызовы `read()` *и* `write()` *не возвращаются, пока актуальные данные не поступят в/из буфера ядра. Все дисковые операции на несколько порядков медленнее операций с памятью, поэтому однопоточное приложение, проводящее операции с файлами, может значительное время проводить в ожидании, что может негативно сказаться на обработке асинхронных событий (сигналы, новые данные в сокетах/каналах). Модуль* `IO::AIO` *предоставляет набор функций, которые позволяют асинхронно выполнять операции ввода/вывода, не блокируя основную нить выполнения процесса.*

Реализации асинхронного ввода/вывода

Существует несколько различных реализаций асинхронного I/O. Стандарт POSIX.1b определяет набор асинхронных функций и их интерфейс, но не регламентирует способ реализации. В библиотеке GNU libc эти функции реализованы на пользовательском уровне, когда запрос на ввод/вывод обрабатывается в выделенном треде, а информация о результатах может доставляться отправкой сигнала или инициацией нового треда.

В ядре Linux, начиная с версии 2.6, появился набор системных вызовов, которые можно использовать в программах непосредственно или через обёртку в виде библиотеки libaio. Реализация aio-функций

в ядре Linux имеет ограничения: требуется открытие файлов с флагом `O_DIRECT`, что должно поддерживаться файловой системой, а также полностью отключается дисковый кэш.

Модуль `IO::AIO` был создан *Марком Лемманном* на основе C-библиотеки libeio, которая также, как и glibc, использует пул тредов для выполнения обычных синхронных операций ввода/вывода, не блокируя основной поток исполнения программы. Такой подход обеспечивает высокую переносимость библиотеки, вплоть до возможности использования на платформе Windows.

API `IO::AIO`

API модуля `IO::AIO` предоставляет множество функций для работы с файлами и директориями, функции для интеграции с модулем обработки событий, а также контролем исполнения. Некоторые функции могут успешно работать и с сокетами, и с каналами, но по эффективности они будут уступать аналогичным функциям из модуля обработки событий (например, EV).

Прежде чем начать описание функций и разбор примеров, необходимо оговориться о способе интеграции `IO::AIO` с модулем обработки событий. Для служебных целей `IO::AIO` создаёт канал, который используется для получения данных из тредов. Как только данный канал становится доступным для чтения, т.е. пришли данные какого-либо запроса, необходимо выпол-

нить функцию обработки этих данных. Дескриптор канала возвращает функция `IO::AIO::poll_fileno()`, а функция обработки это `IO::AIO::poll_cb()`. Таким образом, например, при работе совместно EV требуется установить IO-страж:

```
my $aio = EV::io IO::AIO::
    poll_fileno, EV::READ, \&IO::
    AIO::poll_cb;
```

В общем случае рекомендуется использовать модуль `AnyEvent`, в таком случае интеграция выполняется тривиально, просто загрузив модуль `AnyEvent::AIO`:

```
use AnyEvent;
use IO::AIO;

# Интеграция IO::AIO с AnyEvent
use AnyEvent::AIO;
```

Все дальнейшие примеры будут использо-

вать такую шапку и для простоты она будет опускаться.

Открытие, чтение и запись файла

Для открытия файла предназначена функция `aio_open`, которая принимает в качестве аргументов путь к файлу, битовую маску из флагов (список возможных флагов можно подсмотреть в `Fcntl`), права для создаваемого файла (меняется в соответствии с umask текущего процесса).

Чтение файла производится функцией `aio_read`, которая читает по заданному файловому дескриптору и смещению указанное количество данных в буфер по заданному смещению.

```
1 my $cv = AE::cv;
```

```
2
3  # открытие только на чтение
4  aio_open "/etc/passwd", IO::AIO::
       O_RDONLY, 0, sub {
5
6      # полученный файловый
           дескриптор
7      my $fh = shift or die $!;
8
9      # размер файла
10     my $size = -s $fh;
11
12     my $contents;
13
14     # чтение
15     aio_read $fh, 0, $size,
           $contents, 0, sub {
16
17         # прочитанный размер
               данных
18         my $readed = shift;
19
20         # -1 - признак ошибки
```

```
21          die "ошибка чтения: $!"
               if $readed == -1;
22
23          die "Не удалось прочитать
                 весь файл"
              if $readed != $size;
24          close $fh;
25
26          # печать содержимого
27          print $contents;
28          $cv->send;
29      }
30  };
31
32  $cv->recv;
33
```

Запись файла выполняется функцией `aio_write`, с аналогичным набором аргументов, как и в `aio_read`.

```
1  my $cv = AE::cv;
2
3  # открытие на запись и при
```

```
    необходимости
# создание нового файла с правами
    0666
aio_open "hello", IO::AIO::
    O_CREAT | IO::AIO::O_WRONLY,
    0666, sub {

    # полученный файловый
       дескриптор
    my $fh = shift or die $!;

    my $contents = "hello, world!
       " ;

    # запись из буфера $contents
    aio_write $fh, 0, length(
       $contents), $contents, 0,
       sub {

        # записанный размер
           данных
        my $len = shift;

```

```
18              # –1 – признак ошибки
19              die "ошибка записи: $!"
                   if $len == –1;
20
21              die "Не удалось записать
                   весь файл"
22                 if $len != length(
                      $contents);
23              close $fh;
24
25              $cv–>send;
26          }
27  };
28
29  $cv–>recv;
```

Чтение каталогов

Довольно часто возникает задача получения списка файлов в каталоге, для этих

целей существует несколько асинхронных функций.

`aio_readdir` выполняет сразу три операции: открывает каталог, читает содержимое и закрывает каталог. Ссылка на список полученных элементов передаётся в функцию обратного вызова. Этот список **не содержит** специальных каталогов `.` и `..`.

```
my $cv = AE::cv;

# чтение каталога
aio_readdir "/etc/init.d",  sub {

    # полученный список файлов
    my $list = shift or die $!;

    print join "\n", @$list;
    $cv->send;
};

```

```
$cv->recv;
```

Функция aio_scandir по аналогии с aio_readdir выполняет чтение содержимого каталога, но при этом также пытается определить тип файла. Вызываемая функция обратного вызова получает в случае успеха два списка: список директорий и список остальных файлов.

```
my $cv = AE::cv;

# чтение каталога
aio_scandir "/etc", 0,  sub {
    my ($dirs, $nondirs) = @_ or
       die $!;
    print "каталоги: @$dirs\n";
    print "ёвс остальное:
       @$nondirs\n";
    $cv->send;
};

$cv->recv;
```

В данном примере `aio_scandir` также имеет второй параметр, который определяет, как много выделенных тредов будет занято сканированием (выполнять `stat` на файлы). Значение `0` или меньше означает, что будет использоваться количество тредов по умолчанию: 4.

`aio_scandir` позволяет, например, реализовать поиск файла.

```
use File::Spec::Functions;

sub scan {
    my $cb = pop;
    my $cv = pop;
    my @dirs = @_;

    my $scan;

    # Функция-замыкание, которая
      вызывает саму себя до тех
      пор,
```

```
11      # пока не останется
           непросканированных
           каталогов
12      ( $scan = sub {
13          my $dir = shift @dirs;
14          aio_scandir $dir, 0,  sub
                {
15              my ($dirs, $nondirs)
                   = @_;
16
17              # добавим новые
                   каталоги для
                   сканирования
18              push @dirs, map {
                   catfile $dir, $_ }
                    @$dirs if $dirs;
19
20              # вызываем
                   пользовательскую
                   функцию обратного
                   вызова
21              $cb->($dir, $nondirs)
                   ;
```

```
22
23                  # продолжаем
                        сканирование, если
                         есть что
24                  if (@dirs) {
25                      $scan->()
26                  } else {
27                      $cv->()
28                  }
29              };
30      } )->();
31  };
32
33  my $cv = AE::cv;
34
35  # Регулярное выражение для поиска
        файла
36  my $search = qr/foo|bar/;
37
38  # Поиск в каталоге /etc
39  scan "/etc", $cv, sub {
40      my ($dir, $files) = @_;
41      for my $file (@$files) {
```

```
42              printf "Found match: %s\n
                    ", catfile $dir, $file
43                   if $file =~ $search
44      }
45  };
46
47  $cv->recv;
```

В данном примере функция `scan` выполняет рекурсивный поиск по списку каталогов. Предпоследний параметр — функция, которая вызывается при завершении сканирования (в данном случае — это условная переменная). Последний параметр — это функция обратного вызова, которая вызывается после сканирования каждого каталога. Ей передаются два параметра: `$dir` — текущий каталог, `$files` — ссылка на массив с файлами в текущем каталоге. В данном примере функция ищет имена файлов по регулярному выражению

`/foo|bar/`.

Операции с двумя дескрипторами

Иногда возникает потребность в пересылке данных из одного дескриптора в другой. Например, веб-сервер должен отправить локальный файл через сетевой сокет клиенту. В этом случае удобно использовать функцию `aio_sendfile`, которая при возможности использует системный вызов `sendfile`, позволяя избежать копирования данных из ядра в пользовательские буфер и обратно.

```
1 use AnyEvent::Socket;
2
3 my $cv = AE::cv;
4
5 # ёСоздам tcp–сервер, который
    слушает порт 8080
```

```
# и отправляет всем
    подключившимся клиентам
# файл /etc/passwd
tcp_server undef, 8080, sub {
    my $socket = shift or die $!;

    # откроем /etc/passwd на
       чтение
    aio_open "/etc/passwd", IO::
       AIO::O_RDONLY, 0, sub {
        my $fh = shift or die $!;
        my $size = -s $fh;

        my $sendfile;
        my $offset = 0;

        # Функция-замыкание для
           отправки данных
        ( $sendfile = sub {

            # Копирование из
               файла в сокет
```

```
23          aio_sendfile $socket,
                $fh, $offset,
               $size, sub {
24
25              # Произошла
                    ошибка
26              if ( $_[0] == -1
                    ) {
27
28                  # Не
                        смертельно
                        , можно
                        повторить
29                  if ( $!{
                        EAGAIN} ||
                         $!{EINTR}
                         ) {
30                       $sendfile
                             ->();
31                  }
32
33                  # ёвс-таки
                        что-то не
```

```
            так
        else {
            die $!;
        }
    }

    # Отправлено меньше чем надо, повторяем
    elsif ( $_[0] < $size ) {
        $offset += $_[0];
        $size   -= $_[0];
        $sendfile->();
    }

    # Файл успешно отправлен, конец сеанса
```

```
47                      else {
48                          close $socket
                                ;
49                          close $fh;
50                      }
51                  };
52              } )->()
53      };
54  };
55
56  $cv->recv;
```

В данном примере дополнительно рассмотрено, как обрабатывать ошибки: если вызов `aio_sendfile` вернул `-1`, то необходимо проверить код ошибки, если, например, это `EAGAIN`, то значит сокет ещё не готов принять данные и надо повторить вызов позже.

Операции копирования, перемещения и удаления

Представлен набор асинхронных функций, который позволяет производить типичные операции по манипуляциям с файлами и каталогами:

- `aio_copy` — производит копирование файла (директории не поддерживаются) из пути источника в путь назначения. Функция обратного вызова получает 0 в случае успеха и -1 в случае ошибки.
- `aio_move` — перемещение файла. Это композитная функция, которая пытается выполнить вызов `rename` и в случае, если это невозможно (путь назначения на другом разделе) выполняет копирование `aio_copy`.

- `aio_unlink` — удаление файла по заданному пути.
- `aio_rmtree` — удаление дерева каталогов по заданному пути.

```
aio_copy "/mnt/backup/etc/passwd", "/etc/passwd", sub {
    $_[0] and die $!;
    aio_copy "/mnt/backup/etc/shadow", "/etc/shadow", sub {
        $_[0] and die $!;
        aio_rmtree "/mnt/backup/etc";
    };
};
```

Файловые пути и текущий каталог

Многие aio-функции оперируют с файловыми путями. Модуль `IO::AIO` требует, чтобы все используемые пути передавались как байтовые строки, т.е. если путь передан в программу как строка символов, необходимо принять меры для её приведения к байтовой строке, иначе вызов aio-функции вызовет исключение.

```
# строки кода — набор символов в кодировке UTF-8
use utf8;
use Encode;

my $path = "/путь/юникод/в/кодировке/UTF-8";

# файловый путь кодируется в байтовую строку
```

```
8 aio_stat encode_utf8($path), sub
      {
9     ...
10 };
```

IO::AIO рекомендует использовать абсолютные пути во всех функциях. Это связано с тем, что при смене текущего каталога вполне возможна ситуация, что какой-то запущенный тред работает со старым рабочим каталогом, что может привести впоследствии к ошибке. Разумеется, если вы уверены, что в течении работы программы текущий каталог не меняется, то использование относительных путей вполне легально.

Если использование относительных путей востребованно и требуется контроль текущего каталога при выполнении aio-функций, то в IO::AIO для подобных

целей существует функция aio_wd, формирующая объект класса IO::AIO::WD, который можно использовать в aio-функциях для обозначения текущего каталога.

```
aio_wd "/etc", sub {
    my $etcdir = shift;

    aio_stat [$etcdir, "passwd"],
        sub {
        ...
    }
};
```

Переменная $etcdir сохраняет информацию о каталоге “/etc”, на поддерживающих системах происходит открытие файлового дескриптора каталога, что ускоряет разрешение имён относительного данного каталога. aio_stat в качестве параметра пути получает ссылку на массив, где первый элемент это объект текущего ката-

лога, а второй элемент — относительный путь. Все функции `IO::AIO`, которые работают с путями, поддерживают подобный формат вызова. Таким образом, можно всегда быть уверенным, что вызов функции с относительным путём будет проходить в контексте выбранного текущего каталога, даже если текущий каталог самого приложения изменился.

Запросы

Каждая aio-функция возвращает объект запроса `IO::AIO::REQ`, если вызывается не в пустом контексте. Данный объект ассоциируется с выполняемой операцией. Каждая операция проходит пять состояний:

1. Готовность — сразу после создания за-

проса, ожидание выполнения тредом.

2. Выполнение — запрос принят тредом, и в данный момент тред выполняет его.

3. Ожидание — запрос выполнен, и тред ожидает, когда будет обработан результат.

4. Результат — результат запроса обрабатывается в `poll_cb`, который вызывает пользовательскую функцию обратного вызова.

5. Выполнено — запрос выполнен, все связанные структуры данных освобождены.

До того как запрос перейдёт в стадию выполнению, существует возможность его отменить с помощью вызова метода `cancel` объекта запроса:

```
# создать запрос на удаление
# какого-нибудь ненужного
    каталога
my $req = aio_rmtree "/", sub {
    print "all your base are
        belong to us\n";
};

# отменить запрос, если не поздно
$req->cancel;
```

Также можно задать или переопределить пользовательскую функцию обратного вызова запроса, до того как начнётся обработка результата запроса:

```
# вызов stat на файле
my $req = aio_stat "/etc/passwd";

# установка колбека
$req->cb( sub {
    $_[0] and die "ошибка stat: $
        !";
```

```
    printf "размер %i\n", -s _;
});
```

Группы запросов

Несколько запросов можно объединить в группы, чтобы иметь возможность отследить завершение всех запросов. Для этих целей создана функция `aio_group`, которая принимает один параметр — функцию обратного вызова, которая вызывается, когда завершены все запросы группы. Возвращается объект класса `IO::AIO::GRP`, в который можно добавлять запросы с помощью метода `add`:

```
my $cv = AE::cv;

# ёСоздам группу
my $grp = aio_group sub {
```

```
5       print "выполнены все запросы\
            n";
6       $cv->send
7   };
8
9   # Добавляем первый запрос
10  add $grp aio_stat "/etc/passwd",
        sub {
11      printf "размер passwd %i\n",
            -s _ unless $_[0];
12  };
13
14  # Добавляем второй запрос
15  add $grp aio_stat "/etc/group",
        sub {
16      printf "размер group %i\n", -
            s _ unless $_[0];
17  };
18
19  $cv->recv;
```

Группы, как и сами запросы, можно отменять с помощью вызова `cancel`. В этом случае отменяются все запросы, входящие в группу. Группы также можно добавлять в другие группы.

В `IO::AIO` реализована возможность динамического добавления запросов в группу. Каждой группе можно назначить так называемый генератор или фидер (*feeder*) — функцию, которая вызывается, как только количество запросов в группе снизится ниже определённого уровня. Функция может добавить новый запрос в группу, таким образом “питая” группу.

Например, описанную выше задачу поиска файла по заданному регулярному выражению можно реализовать с помощью группы и генератора:

```
1 use File::Spec::Functions;
```

```
2
3  sub scan {
4      my $cb = pop;
5      my $cv = pop;
6      my @dirs = @_;
7
8      # Группа запросов, в качестве
           функции обратного вызова
9      # ёпередатся условная
           переменная $cv
10     my $grp = aio_group $cv;
11
12     # Установим лимит на
           минимальное число запросов
           в группе,
13     # после которого вызывается
           фидер
14     limit $grp 1;
15
16     # Генератор/фидер запросов
17     feed $grp sub {
18         my $dir = shift @dirs //
               return;
```

```
19
20          # Добавляем запрос в
                группу
21          add $grp aio_scandir $dir
                , 0,  sub {
22              my ($dirs, $nondirs)
                    = @_;
23              push @dirs, map {
                    catfile $dir, $_ }
                     @$dirs if $dirs;
24
25              # Вызываем функцию
                    пользователя
26              $cb->($dir, $nondirs)
                    ;
27          };
28      };
29  };
30
31  my $cv = AE::cv;
32
33  # Регулярное выражение для поиска
        файла
```

```
34 my $search = qr/foo|bar/;
35
36 # Поиск в каталоге /etc
37 scan "/etc", $cv, sub {
38     my ($dir, $files) = @_;
39     for my $file (@$files) {
40         printf "Found match: %s\n
              ", catfile $dir, $file
41              if $file =~ $search
42     }
43 };
44
45 $cv->recv;
```

В данном примере мы модифицировали функцию `scan`, создав группу, которую снабжает новыми запросами функция-генератор. Также обратите внимание, что используется специальная функция `limit`, которая устанавливает минимальный порог запросов, после которого должен вызываться генератор/фидер. В данном

примере, это 1, таким образом, одномоментно работает только один aio-запрос.

Служебные функции и функции тонкой подстройки `IO::AIO`

`IO::AIO` имеет несколько функций, которые относятся непосредственно к работе самого модуля `IO::AIO`, позволяя подстроить различные аспекты его функционирования.

- `IO::AIO::min_parallel` — минимальное число одновременно запущенных тредов для обработки запросов, по умолчанию 8.
- `IO::AIO::max_parallel` — максимальное число одновременно

запущенных тредов для обработки запросов, по умолчанию 32.

- `IO::AIO::max_idle` — максимальное число простаивающих тредов
- `IO::AIO::nreqs` — возвращает количество готовых, выполняющихся и ожидающих обработку тредов.

Помимо служебных функций `IO::AIO::poll_fileno` и `IO::AIO::poll_cb`, описанных в начале статьи, есть ещё несколько функций, связанных с обработкой запросов.

- `IO::AIO::poll_wait` — блокирующее ожидание, пока один из запросов не перейдёт в состояние готовности для чтения.

- `IO::AIO::flush` — блокирующее ожидание, пока все запросы не будут обработаны, аналогичен выполнению кода:

```
IO::AIO::poll_wait, IO::AIO
    ::poll_cb
  while IO::AIO::nreqs;
```

Данные функции могут быть полезны при использовании `IO::AIO` без модуля обработки событий.

Заключение

`IO::AIO` однозначно является самой мощной системой для работы с дисковой подсистемой в Perl. Интеграция с различными модулями обработки событий,

огромное число функций для всевозможных нужд, высокая переносимость модуля, тонкая настройка, адаптация к высокой нагрузке являются наиболее заметными преимуществами `IO::AIO`. Если вы разрабатываете систему, которая генерирует высокую нагрузку на дисковую подсистему и при этом должна оставаться крайне отзывчивой к внешним воздействиям, то лучше `IO::AIO` вы вряд ли сможете что-то найти.

В статье описаны лишь немногие востребованные функции, исчерпывающий их список с подробным описанием доступен в POD-документации модуля.

■ *Владимир Леттиев*

6 Использование портов GPIO в Raspberry Pi. Часть 1

В этой статье рассказано о том, как устроены универсальные порты ввода-вывода (GPIO) и о том, как работать с ними на перле.

Одноплатный компьютер Raspberry Pi примечателен не только своими размером и ценой, но и наличием порта ввода-вывода общего назначения (General Purpose Input-Output, GPIO). Эдакая современная версия параллельного LPT-порта, которая вместе с компьютером занимает меньше места, чем плата PCI с контроллером LPT.

Базовая информация о портах

В базовом комплекте второй версии Raspberry Pi установлен один 26-контактный разъем (он обозначен на плате как P1), в котором доступно 17 портов ввода-вывода (тех самых GPIO). Остальные восемь контактов подключены к земле или к питанию +3,3 В и +5 В. Выводы, отданные под GPIO, могут быть программно переконфигурированы для работы в качестве последовательных портов, вывода с широтно-импульсной модуляцией и еще как-то (вот это все называется альтернативными режимами). Но по умолчанию после включения питания все контакты работают в режиме «один контакт — один бит». Каждый из них может быть либо входом, либо выходом (по умолчанию включен режим ввода).

Кроме упомянутого разъема P1 на плате есть отверстия для установки восьми-контактного разъема P5, который дает возможность получить еще четыре порта GPIO. Итого в распоряжении программиста оказывается 21 бинарный порт.

Наличие GPIO позволяет относительно легко связывать компьютер с устройствами из реального мира и управлять ими программно. Разумеется, здесь потребуется знание не только программирования, но и хотя бы основ электроники (детям рекоменую книгу В. Т. Полякова «Посвящение в радиоэлектронику», взрослым — книгу П. Хоровица и У. Хилла «Искусство схемотехники»).

При работе с GPIO важно учитывать пару моментов:

1. Рабочее напряжение всех выводов — 3,3 В. Случайная подача на вход GPIO большего напряжения (даже 5 В с соседнего штырька разъема) приводит к выходу из строя не только этого вывода, но и вообще всего Raspberry Pi (подтверждаю экспериментально).
2. Контакты разъема P1 и нумерация портов GPIO не совпадает, поэтому при программировании надо всегда помнить, какая из нумераций используется. Еще более они не совпадают в первой версии Raspberry (надеюсь, сейчас, если не прилагать дополнительных усилий, купить удастся только новую модель).
3. Дополнительным пунктом надо отметить, что нумерация самих GPIO в Raspberry Pi идет с пропусками.
4. Raspberry Pi построен на ARM-процессоре BCM2835, поэтому иногда

полезнее гуглить *BCM2835*, а не *Raspberry GPIO* (то же самое действительно для поиска на CPAN).

Хорошее практическое описание опубликовано на странице elinux.org/RPi_Low-level_peripherals.

Для справки, вот так разведены порты GPIO на контакты разъемов P1 и P5 (контакты традиционно обозначаются в формате PX-NN, где X — номер раъема, а NN — двузначный номер контакта):

1. GPIO02 — P1-03
2. GPIO03 — P1-05
3. GPIO04 — P1-07
4. GPIO07 — P1-26
5. GPIO08 — P1-24
6. GPIO09 — P1-21

7. GPIO10 — P1-19
8. GPIO11 — P1-23
9. GPIO14 — P1-08
10. GPIO15 — P1-10
11. GPIO17 — P1-11
12. GPIO18 — P1-12
13. GPIO22 — P1-15
14. GPIO23 — P1-16
15. GPIO24 — P1-18
16. GPIO25 — P1-22
17. GPIO27 — P1-13
18. GPIO28 — P5-03
19. GPIO29 — P5-04
20. GPIO30 — P5-05
21. GPIO31 — P5-06

Каждый битовый порт способен работать в режиме ввода или вывода. Кроме того, в режиме ввода может быть дополнительно включен подтягивающий резистор, что

поможет максимально упростить способ подключения выключателей — их достаточно подключить между соответствующим выводом GPIO и общим проводом (если включен резистор pull up) или между GPIO и источником питания +3,3 В (если выход сконфигурирован в режиме pull down).

За более детальными продробностями о внутреннем устройстве портов GPIO я отсылаю читателя к шестой главе «General Purpose I/O (GPIO)» мануала по процессору BCM2835.

Регистры для работы с GPIO

Режим, в котором работает каждый отдельный разряд порта GPIO, управляется полностью программным способом. В этой статье

рассмотрен только режим ввода-вывода, который включается по умолчанию после подачи на устройство питания.

Процессор BCM2835 имеет 41 32-разрядный регистр, которые полностью определяют режим и состояние портов GPIO. В частности, для установки единичного значения на выводе, запрограммированном на работу как выход, необходимо записать единичный бит в соответствующий разряд одного из двух регистров установки битов GPIO Pin Output Set Registers (GPSETn). Чтобы установить выход в ноль, следует выставить единичный бит в регистрах сброса битов GPIO Pin Output Clear Registers (GPCLRn). Такая на первый взгляд странная схема позволяет независимо устанавливать и сбрасывать любой бит GPIO без необходимости чтения текущего состояния выводов.

Аналогично, когда разряды GPIO работают на чтение, то узнать уровень входного сигнала можно, прочитав значение одного из двух портов GPIO Pin Level Registers (GPLEVn), каждый бит которого отображает текущее состояние входного разряда.

Программирование портов ввода-вывода

Регистры, отвечающие за работу с GPIO, расположены по адресам 0x7E200000—0x7E2000B0, которые отображаются на физическую память с адресами, начинающимися с 0x20200000. В принципе, в этом месте уже можно было бы начать управлять портами, программируя чтение и запись нужных битов в эти регистры (на перле это вполне возможно, если восполь-

зоваться мапингом переменных на области памяти, используя модуль Sys::Mmap).

Но более практично еще немного усложнить систему, чтобы программировать стало легче.

Модуль Device::BCM2835

Для управления портами ввода-вывода на перле удобно воспользоваться модулем Device::BCM2835. Он является Perl-оберткой над C-библиотекой того же автора bcm2835 и один в один повторяет все ее функции (поэтому документацию может оказаться полезнее почитать в оригинале).

Установка библиотеки bcm2835 не вызывает сложностей:

```
./configure
make
sudo make install
```

Равно как и модуль со CPAN:

```
perl Makefile.PL
make
sudo make install
```

Библиотека (и модуль на перле) определяет огромное число констант, позволяющих выбирать нужные выводы разъемов P1 и P5 и устанавливать режимы их работы, и довольно большое число функций для доступа к отдельным битам GPIO.

Все дальнейшие действия необходимо выполнять от имени суперпользователя.

Перед началом работы следует вызвать функцию `init()`:

```
use Device::BCM2835;
Device::BCM2835::init() || die "
    Could not init library";
```

(При инициализации происходит вызов функций `mmap` и создание переменных, которые отображаются на нужные регистры процессора — ровно то, где прикладному программисту проще воспользоваться готовой библиотекой.)

Вызов теоретически может завершиться неудачей, хотя наиболее вероятная причина отказа — запуск скрипта не от рута:

```
bcm2835_init: Unable to open /dev
    /mem: Permission denied
```

Примечание ко всем примерам кода из документации. Модуль Device::BCM2835 разрешает экспортировать используемые

константы, но при этом не хочет экспортировать имена функций (несмотря на то, что многие из них начинаются с префикса gpio_). Поэтому если подключить модуль с экспортом констант, то бесконечные повторы получится значительно, но все же не полностью, сократить и вместо:

```
use Device::BCM2835;
. . .
Device::BCM2835::gpio_fsel(
    &Device::BCM2835::
       RPI_V2_GPIO_P1_05,
    &Device::BCM2835::
       BCM2835_GPIO_FSEL_OUTP
);
```

записывать:

```
use Device::BCM2835 ':all';
. . .
Device::BCM2835::gpio_fsel(
   RPI_V2_GPIO_P1_05,
```

```
    BCM2835_GPIO_FSEL_OUTP);
```

Вывод Чтобы переключить один из разрядов порта GPIO для работы в режиме вывода, надо вызвать функцию установки режима `gpio_fsel($pin, $mode)` и передать ей номер вывода (константу, соответствующую номеру физического вывода нужного разъема) и режим (другую определенную константу). Например, чтобы перевести выход P1-12 в режим вывода, следует выполнить следующее:

```
Device::BCM2835::gpio_fsel(
    RPI_V2_GPIO_P1_12,
    BCM2835_GPIO_FSEL_OUTP);
```

Запись нуля или единицы выполняют, либо вызывая функцию `gpio_write($pin, $value)`:

```
Device::BCM2835::gpio_write(
    RPI_V2_GPIO_P1_12, 1);
Device::BCM2835::gpio_write(
    RPI_V2_GPIO_P1_12, 0);
```

Либо используя пару функций `gpio_set($pin)` и `gpio_clr($pin)`, которым достаточно передать номер вывода:

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

Обратите внимание, что в обоих примерах физический вывод P1-12 (`RPI_V2_GPIO_P1_12`) является логическим выходом GPIO18. А еще обратите внимание на наличие `V2` в именах констант. Для первой версии Raspberry Pi следует использовать кон-

станты типа `RPI_GPIO_P1_12`, которые за парой исключений совпадают с вариантами с `V2`. Но следует иметь в виду, что порт P5 доступен только во второй версии, например: `RPI_V2_GPIO_P5_04` для GPIO29.

Ввод Чтение данных с портов ввода-вывода также прост, как и запись в них. Прежде всего, необходимо перевести соответствующие разряды GPIO в режим чтения (BCM2835_GPIO_FSEL_INPT):

```
Device::BCM2835::gpio_fsel(
    RPI_V2_GPIO_P1_10,
    BCM2835_GPIO_FSEL_INPT);
```

По желанию и необходимости можно подключить один из подтягивающих резисторов:

```
# Подтягиваем к питанию
```

```
Device::BCM2835::gpio_set_pud(
    RPI_V2_GPIO_P1_10,
    BCM2835_GPIO_PUD_UP);

# Подтягиваем к земле
Device::BCM2835::gpio_set_pud(
    RPI_V2_GPIO_P1_10,
    BCM2835_GPIO_PUD_DOWN);
```

В случае, если к выводу порта подключена кнопка, замыкающая вывод на землю, то будет полезен резистор, подключенный к источнику питания +3,3 В (BCM2835_GPIO_PUD_UP), что обеспечит на входе уровень логической единицы, когда кнопка не нажата. При нажатой кнопке на входе окажется логический нуль.

Чтобы прочитать значение со входа, достаточно вызвать функцию `gpio_lev($pin)`, передав ей номер нужного входа:

```
say Device::BCM2835::gpio_lev(
    RPI_V2_GPIO_P1_10);
```

Функция возвращает либо нуль, либо единицу.

Примеры Как видно из предыдущих разделов, работать с портами ввода-вывода в бинарном режиме очень просто. Для закрепления материала — два небольших примера, которые будет полезно выполнить, если вы соберетесь программировать порты GPIO.

Код из первого примера раз в секунду включает и выключает светодиод, подключенный (через резистор сопротивлением 300...1000 Ом) к выводу GPIO03 (контакт P1-05).

```
use v5.12;
```

```
2 use Device::BCM2835;
3
4 Device::BCM2835::init() || die "
    Could not init library";
5
6 Device::BCM2835::gpio_fsel(
    RPI_V2_GPIO_P1_05,
    BCM2835_GPIO_FSEL_OUTP);
7
8 while (1) {
9     Device::BCM2835::gpio_set(
        RPI_V2_GPIO_P1_05);
10    Device::BCM2835::delay(500);
11
12    Device::BCM2835::gpio_clr(
        RPI_V2_GPIO_P1_05);
13    Device::BCM2835::delay(500);
14 }
```

Обратите внимание на использование функции `Device::BCM2835::delay($n)` из того же модуля, которая вы-

полняет задержку на `$n` миллисекунд. Доступна и функция `Device::BCM8235::delayMicroseconds($n)` для задержки в микросекундах. Обе, однако, не гарантируют точности отсчитанного времени. На практике следует быть осторожным, если требуется получить более или менее точные задержки меньше 20-50 миллисекунд.

Второй пример включает тот же светодиод при нажатии на кнопку, подключенную ко входу GPIO15 (P1-10).

```
1 use v5.12;
2 use Device::BCM2835;
3
4 Device::BCM2835::init() || die "
    Could not init library";
5
6 Device::BCM2835::gpio_fsel(
    RPI_V2_GPIO_P1_05,
    BCM2835_GPIO_FSEL_OUTP);
7
```

```
8  Device::BCM2835::gpio_fsel(
       RPI_V2_GPIO_P1_10,
       BCM2835_GPIO_FSEL_INPT);
9  Device::BCM2835::gpio_set_pud(
       RPI_V2_GPIO_P1_10,
       BCM2835_GPIO_PUD_UP);
10
11 while (1) {
12     Device::BCM2835::gpio_write(
13         RPI_V2_GPIO_P1_05,
14         !Device::BCM2835::
               gpio_lev(
               RPI_V2_GPIO_P1_10)
15     );
16     Device::BCM2835::delay(50);
17 }
```

Здесь интересно отметить, что в отличие от предыдущего примера удобнее воспользоваться функцией `gpio_write`, а не парой `gpio_set` и `gpio_clr`.

■ *Андрей Шитов*

7 Обзор CPAN за июнь 2014 г.

Рубрика с обзором интересных новинок CPAN за прошедший месяц.

Статистика

- Новых дистрибутивов — 247
- Новых выпусков — 768

Новые модули

- HTTP::Tiny::Paranoid

`HTTP::Tiny::Paranoid` является надстройкой над `HTTP::Tiny`, которая перед

подключением выполняет разрешение имени хоста в Net::DNS::Paranoid, что позволяет исключить подключение к ip-адресам из диапазонов приватных сетей, а также выполнять проверку по чёрным/белым спискам хостов.

- Net::Netconf

Компания Juniper выпустила свой первый официальный модуль на CPAN для управления конфигурацией сетевого оборудования своего производства. Не совсем понятна, какова лицензия продукта, но код открыто доступен.

- BusyBird

`BusyBird` — это веб-приложение для просмотра ленты статусов/сообщений. Это своеобразный симбиоз twitter'а и RSS-агрегатора. BusyBird сам не выполняет сбор сообщений, но имеет API для их получения. Таким образом, пользователь должен быть немного программистом, чтобы написать модули, которые бы собирали данные, например, твиты, почтовые сообщения, RSS-ленты и передавали в BusyBird, задача которого уже отображать эти данные.

- Protocol::OTR

`Protocol::OTR` — это обвязка к библиотеке libotr v4, реализующая криптографический протокол Off-the-Record для систем мгновенных сообщений, позволяющая безопасно обмениваться сообщениями, аутентифицировать собеседников, защищать

историю переписки при компрометации ключей.

- Etcd

Etcd является perl-клиентом для высокодоступного сервиса хранения пар ключей-значений для разделяемых конфигураций etcd.

- Cookieville

Cookieville — это веб-приложение на базе Mojolicious, реализующее REST-интерфейс к базе данных. Для работы должен быть создан класс схемы `DBIx::Class`.

- Devel::Trace::Syscall

`Devel::Trace::Syscall` позволяет выводить трассировку системных вызовов, которые вызываются в вашей Perl-программе. Модуль использует вызов `ptrace` и работает только на Linux, также требуется Perl 5.18.0 или старше.

```
$ perl -d:Trace::Syscall=
   open my-script.pl
open("somefile", 0x0, 0666)
   = -2 at my-script.pl line
    4
```

- Protocol::Gearman

`Protocol::Gearman` — это ещё одна реализация протокола, а также клиента и рабочего процесса для сервера задач Gearman.

- Async::Chain

Модуль `Async::Chain` позволяет организовать группу вложенных функций обратного вызова в плоскую структуру, более наглядную и удобную для сопровождения. Вместо написания

```
async_func1 cb => sub {
    async_func2 cb => sub {
        ...
    }
}
```

Указывается ключевое слово `chain` и далее следует последовательность функций:

```
chain
    sub {
        async_func1 cb =>
            shift;
    },
    'optional name of sub' =>
        sub {
```

```
6               async_func2 cb =>
                    shift;
7           },
8           ...
9       ;
```

- warnings::pedantic

Прагма `warnings::pedantic` добавляет шесть новых предупреждений для сомнительных конструкций: grep, close, print в пустом контексте, некорректный прототип для sort-функции, присвоение массиву ссылки на массив, использование констант с правой стороны «жирной запятой» или в качестве ключа хеша.

Обновлённые модули

- Coro 6.39

В новой версии реализации сопрограмм `Coro` улучшена совместимость с Perl 5.20.

- Sereal 3.001

Новый мажорный релиз модуля (де)сериализа `Sereal` включает поддержку сжатия zlib и новый магический заголовок данных для лучшего детектирования закодированных в UTF-8 данных.

- Mango 1.01

Первый мажорный релиз драйвера MongoDB `Mango` официально больше не считается экспериментальным.

- IO::AIO 4.31

Обновлён модуль асинхронного ввода/вывода `IO::AIO`. В новом релизе сделаны исправления для работы с Perl 5.20.

- App::perlbrew 0.69

Новый релиз утилиты perlbrew для инсталляции и управления версиями Perl в домашнем каталоге исправляет проблему с установкой perl 5.21, а также проблемы при работе со сломанной версией `local::lib`.

- AnyEvent::HTTP 2.21

В новом релизе исправлена ошибка в формировании заголовка `Connection`. Добавлена поддержка кода перенаправления 308, а также всех допустимых идемпотентных методов HTTP/1.1 в запросе (их на сегодняшний день 34).

- B::C 1.47

Новый релиз компилятора Perl `B::C` содержит множество исправлений, включая исправления байткода для Perl 5.18, лексических подпрограмм для Perl с поддержкой тредов, работы модуля Coro. Существенно уменьшен генерируемый код за счёт исключения ненужных зависимостей.

- Email::Address 1.905

Новый релиз модуля `Email::Address` для проверки email-адреса в соответствии с RFC 2822 исправляет ошибку в безопасности CVE-2014-0477, которая позволяла организовать DoS-атаку на сервисы, использующий данный модуль. Функция `Email::Address::parse` затрачивала существенное время при разборе, если встречалась последовательность из двух двойных кавычек "".

- perl 5.21.1

В новый релиз perl для разработчиков 5.21.1 включена поддержка Unicode 7.0, а также реализовано API для вывода трассировки вызовов внутренних функций perl при предупреждениях, аналогичный выводу gdb:

```
$ env
   PERL_C_BACKTRACE_ON_WARN
   =10 perl —we '$x=1'
0    496712:0012
   Perl_dump_c_backtrace
   util.c:6138    perl
1    4967b7:0047
   Perl_mess_sv    util.c
   :1382    perl
2    497786:0006
   Perl_vwarn      util.c
   :1810    perl
3    497c87:0087
   Perl_warner     util.c
   :1902    perl
4    44a8a5:0215
   Perl_gv_check   gv.c:2304
       perl
5    4487fc:14fc
   S_parse_body    perl.c
   :2314    perl
6    42b630:0080    main
              perlmain.c:113
```

```
        perl
7       7f031f593ad5:00f5    —
         —    /lib64/libc.so.6
8       42b6ed          _start
                start.S:126
        perl
Name "main::x" used only once
   : possible typo at —e line
    1.
```

- Readonly 1.61

Для обновлённого `Readonly` больше не требуется `Readonly::XS`. Также исправлено множество ошибок в коде и неточностей в документации.

■ *Владимир Леттиев*

8 Интервью с Питэром Рэббитсоном (Peter Rabbitson)

Питэр Рэббитсон (ribasushi) — Perl-программист, наиболее известный как текущий разработчик DBIx::Class.

Как и когда научился программировать?

Сначала стандартная история — лет в 9 появился доступ к компьютеру — сначала на работе отца и у друзей, а потом в 11 и дома. Собственно программировать начал в 1992-м на Бейсике на шикарной ИЗОТ 1036С / ЕС1832 http://bulgariancomputers.freeservers.com/isot1036c/isot1036c_eng.html. Очень нравилось возиться с довольно простыми, но внушающими визуализациями на основе тригонометрии. Пытался делать что-то на Pascal и на Turbo-C, но продрать-

ся через управление памятью мне самому не удалось. Параллельно с этим получил первые навыки системного администрирования, вплоть до сооружения локальной сети с соседом по лестничной клетке путем сделанного вручную(!) 30-метрового провода для последовательной (COM) и параллельной (LPT) передачи файлов.

А потом мне стукнуло 13, и я забил на все что-либо связанное с программированием — интерес круто перешел на копание в сетевом оборудовании, знакомство с интернетом, попыткам освоить линукс (вначале довольно неуспешным) и не в последнюю очередь компьютерные игры.

Мой развод с программированием продолжался довольно долго — вплоть до 2002 года. В то время я работал заведующим на складе компьютерной техники. Соб-

ственник склада пытался выйти на рынок онлайн-продаж, пользуясь для отчета наличности двсякой ребеденью сомнительного качества. Почти каждый день что-то не стыковалось, постоянно или обнаруживались недостатки, или товар появлялся из воздуха. Я кипятился и пересчитывал все заново, пока мне это все не надоело и я не стал копаться как там все все-таки устроено. Системы общались чем-то CSV и еще XML. Знакомый админ посоветовал мне попробовать Perl, так как я уже знал немного Bash на уровне написания эквивалентов `.bat`-файлов... Остальное, как говорится, уже история. В 2004-м задал первый вопрос в рассылке perl.beginners и через месяц попал на IRC.

С какими другими языками интересно работать?

Получилось так, что на сегодняшний день я хорошо знаю только Perl и Bash. Да, я могу написать что-то элементарное на Python, на JavaScript или даже на C, но я не знаком с особенностями и «изюминками» этих языков. В будущем хочу по-настоящему (на уровне Perl) изучить C, но пока обхожусь одним языком.

Какой редактор используешь?

В последние 11 лет (с тех пор как окончательно съехал с Windows) я пользуюсь исключительно mcedit: форк cooledit в поставке с MidnightCommander. Да, я так же пользуюсь самим MidnightCommander. Кроме очевидного плюса работы в знакомом не меняющемся окружении, неестественно синий экран привлекает ничего не подозревающих зевак на конференциях. Когда разговор быстро доходит до «И

ты на этом пишешь!?», я уведомляю что 70% кода `DBIx::Class` на сегодняшний день сверстано в этом же синем окне и делаю фотографию в стиле «реакция зрителя на девушек со стаканом». Спешу заметить, что я не исключение, например, релиз-инжинеринг Perl 5.21.1 прошел исключительно с помощью nano (бежит за фотоаппаратом).

Я пробовал работать почти во всех редакторах, включая Vi(m), nano, notepad, notepad++, Kate, Eclipse, Komodo Edit и Komodo IDE, Padre... Вроде как только на Emacs-е не доводилось. Но мне гораздо удобнее пользоваться привычными инструментами, что я кстати советую любому программисту. По профессии нам и так приходится принимать чрезмерное количество непростых решений, можно обойтись без вечного вопроса о редакторе

:)

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

Мне, наверное, влетит за то, что я не продвигаю «линию партии», но что правда, то правда: по-моему сильная сторона Perl не в CPANe, не в сети CPAN Testers, и не в CPANTS. Не в скорости разработки и не в свободе решить задачу несколькими способами. И даже не в сообществе, которое мне лично довольно дорого. Сильная сторона Perl — в его стабильности и в его повсеместности. Perl можно найти почти на любой платформе, начиная с динозавров z/OS (если не слыхали о EBCDIC — поинтересуйтесь) и заканчивая миниатюрным линуксом, впихнутым на флеш-карточку SD http://haxit.blogspot.de/2013/08/hacking-transcend-wifi-sd-cards.html. Причем гра-

мотно написанное приложение будет работать совершенно одинаково на любом железе и без особых усилий на любой версии perl, включая версии более чем десятилетней давности. Если сделать шаг назад и присмотреться, Perl удалось сделать то, чего не удалось Java (несмотря на несоизмеримое наличие средств и кадров): священный Грааль «напиши раз — запускай где угодно» (write once — run anywhere). И по-моему не важно, что эта особенность Perl не особенно привлекательна для новичков и дает повод закатывать глаза поклонникам Modern Perl. Преимущество здесь, оно работает, и малое, но достаточное количество ключевых разработчиков готово за нее постоять. Так что на это можно и нужно рассчитывать.

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

Встроенная закулисная многопоточность. Не асинхронность, не корутины и не треды, а именно то, чего пытается добиться Perl 6 (хотя к сожалению об этом мало говорят). Я считаю что многопоточность — последний бастион концепций, не укладывающихся в головы большинства программистов. Наша профессия расправились с управлением памяти, абстрагировала ввод и вывод вплоть до потери разницы между локальным файлом и данными, блуждающими в сети, понаделала кучу абстракций как для элементарных 2D-приложений, так и для сногсшибательно реалистичных 3D-эффектов. Но на сегодняшний день все еще нет возможности построить цепь передачи данных между функциями типа «unix shell piping» и, взмахнув волшебным оператором, заказать, чтобы все это прогналось через свободные CPU и GPU ядра в системе. Мне кажется, что движок Perl 6

(точнее то, что лежит между грамматикой и VM) наиболее близок к постижению такого колдовства. Вот только если бы не умопомрачительный синтаксис Perl 6... но это уже совсем другая история :)

Как получилось, что стал развивать DBIx::Class? Как вообще управляются проекты такого масштаба?

В начале 2008 года я начал пользоваться `DBIx::Class` и довольно быстро наткнулся на проблему с дополнительной выборкой (prefetch). Несмотря на то, что мне было совсем непонятно, откуда ножки растут, я составил довольно грамотный тест «на потом» (TODO). Кроме этого было несколько патчей, чтобы поправить несколько надежно падающих тестов. Кода было не так уж и много, но Мэтт Траут (Matt «mst» Trout) на это все посмотрел и

дал мне права на репозиторий. Я потрогал в двух-трех местах, и забросил все на лето. В конце августа потыкал снова тут и там, и... пошло поехало: с тех пор так и тыкаю. Со временем легких проблем становилось все меньше, а армия пользователей становилась все больше и больше. Это привело с собой новые требования к стабильности и осторожности внедрения новых фич, требования, сравнимые вроде как только с самим интерпретатором perl. Кстати о сложности проблем — на то, чтобы вернуться и полностью расколоть недоработку, которая собственно и привела меня к DBIC, у меня ушло (подергивая глазом) аккурат 5 лет — фича «свободная дополнительная выборка» (arbitrary prefetch) ушла на CPAN в апреле 2013-го года.

Насчет управления проекта — я и сам не знаю :) Я наверное что-то сделал не

совсем так, ибо состав разработчиков на сегодняшний день: полтора человека. С другой стороны, пользователи вроде как довольны. Прошлым летом меня довольно сильно заели сомнения по этому поводу, и я написал статью «крауд-соурсинг самоуверенности» где я просто поставил вопрос «А нужен нам такой DBIC?» Получил сногсшибательное количество позитивных отзывов. Так что пока я не сдаюсь и даже пытаюсь вывести разработку на новый уровень. Что из этого получится — пока не уверен, но новости должны появится к концу этого месяца, или на худой конец до начала YAPC::EU 2014.

С технической точки зрения пока много куда развиваться: увеличение производительности, добавка чертовски нехватающих функций API, качественно новый способ дополнительной выборки, асин-

хронность и много чего еще. Вопрос о том, как найти время, чтобы всем этим заняться и одновременно не снизить качество проекта, я пока не решил :)

Чем занимаешься в p5p?

В плане разработки практически ничем. Я был назначен релиз-менеджером для Perl 5.19.5, но со временем сложилось так, что мне пришлось передать задачу Стиву Хею. Так что по сей день реального кода в Perl 5 у меня нет. Зато иногда я встреваю в «политические игры», проводящиеся в рассылке perl5-porters, когда мне кажется что все с разумными доводами вдруг сговорились и ушли дружно в отпуск. Хотя я делаю это довольно редко, по-моему, эффект какой-то есть. Как минимум, я могу приписать в свои заслуги предотвращение поломки поведения $@ во время DESTROY

в 5.14 (там история сложнее чем кажется на первый взгляд), введения экспериментального флага для smartmatch (~~) в 5.18, и отказ принять первый вариант сигнатур функций (то что приняли под конец в 5.20 в разы более элегантно и продумано).

Расскажи про свою кампанию на Crowdtilt в 2013 году. И как в этом году удалось поехать на YAPC::NA?

Если смотрел фильм «Adaptation» — все случилось по той же идее. Я просто сел одним вечером и начал писать, как я решаю купить себе краудсорсный билет. Закончил, прочитал, проникся, переделал в правильную форму (прочитав несколько «конкурентных кампаний») и… поехали. Подробнее я описал, как все произошло в блоге, но все сводится к заключению: «Короче, Perl привлекает клевых людей.

Не просто людей, с которыми приятно выпить кружку пива, а тех которые хотят выпить кружку именно с тобой». Если есть интерес, к следующему выпуску Pragmatic Perl могу перевести статью целиком на русский язык ;)

В этом году я никак не был богаче, но раскручивать кампания заново как-то… не дело. Я решил довольно давно, еще в феврале, что пора уступить место другим. Причем в последний момент Леон Тимерманнс (известный в основном своим участием в разработке Module::Build(::Tiny) и поддержкой системы PerlIO) действительно успешно опубликовал и закончил такую же кампанию.

Так как же мне удалось съездить еще раз. В апреле я разговаривал с Хенри Ван Штейном, создателем шикарной системы

http://www.rapidapp.info/. Когда речь зашла о YAPC, он не только предложил лично оплатить всю поездку (размером кстати в $1500 - деньги немаленькие). Нет, он не отстал от меня, пока я не согласился приехать, несмотря на довольно неловкую для меня ситуацию, за что я ему бесконечно благодарен. В общем получается поддержка крупного проекта не такая уж бесперспективная штука ;)

Пересекаются ли глобальное и xCCCP Perl-сообщества? Все дело в языке?

В основном дело, к сожалению, действительно в языке, причем я сам этого не подозревал, пока не приехал на YAPC::RU в 2012 году. Тем более досадно, что у xCCCP-сообщества довольно много свежих идей, о которых в глобальном сообществе думают, но не говорят. В основном вы-

деляется упор на производительность и низкое количество зависимостей. Кстати, та же проблематика в основном закрыто обсуждается и среди японского сообщества, и опять же вся проблема в языке. В общем всем бывшим соотечественникам могу сказать только одно — учите английский и читайте/пишите больше. Шума хронически не хватает.

Чем тебя привлекают Perl-конференции? Доклады, социализация или что-нибудь еще?

Ха! Ты отлично знаешь, что это вопрос с подвохом, наверное потому и задал ;) Как и большинство, я начал посещать конференции ради докладов и чтоб вживую познакомится со «звездами Perl», но это быстро переросло в «чтоб встретится и провести несколько вечеров с офигенными

друзьями». К сожалению, я не могу ответить на вопрос точнее — надо приехать и попробовать самому. И потом, первое правило клуба Perl ... :)

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

На сегодняшний день пишу исключительно на Perl и довольно часто, но довольно редко что-либо за это получаю. В этом году у меня что то вроде полуотпуска, так как у меня случился довольно жестокий burnout, и заниматься программированием за деньги стало... непросто. Подрабатываю изредка для ShadowCat и вяло занимаюсь поддержкой в той же конторе, где научился по настоящему писать. Остальное время уходит на свободное ПО: в основном продолжаю медленно заниматься усовершенствованием `DBIx::Class` и

подумываю над некоторыми пока сверхсекретными проектами... Кстати, если у кого валяются лишние биткоины — `1riba1Z6o3man18rASVyiG6NeFAhvf7rU` ;)

Стоит ли советовать молодым программистам учить сейчас Perl?

Здесь два вопроса. Я честно не знаю, что посоветовать людям, которые идут в программисты с идеей хорошо заработать – у меня этого пока точно не получилось, да я особенно и не пытался. Могу сказать что Perl-а на так называемом DarkPan-е (закрытый перл-код на разных предприятиях) ужасно много, и вряд ли он скоро куда-нибудь денется. Так что заработать в будущем будет все легче и легче, было бы желание. На сегодняшний день C++ уже созрел таким же образом — достаточно взглянуть на зарпла-

ты толкового сишника.

Одержимым молодым людям, которым немыслимо заниматься чем-то другим кроме программирования, у меня единственный совет – выбирайте, что учить следующим образом:

1. Выберите языки по наличию ментора. У вас наверняка есть близкие знакомые специалисты, у которых можно спросить что угодно, когда угодно о том, что решено изучать. У каждого из нас есть такие знакомые, причем он или она будут в абсолютном восторге, что их попросили общаться на эту тему – просто узнайте чем они интересуются. Если знакомых все же нет – зайдите на meetup.com и заявитесь на любой вечер.

2. Отсеяв несколько вариантов, посмотрите, что больше нравится на чисто эстетическом уровне. Здесь не важна перспектива, не важен хайп. Работать с языком вам, и если не суждено – всегда можно будет переключится на что-нибудь другое.

Так что если у вас есть крутые знакомые перловики – смело изучайте Perl. Если хаскелиты – ради бога учите Haskell. Если brainfuck или x86 assembler… наверное пора найти новых знакомых.

Вопросы от читателей

Сколько раз выступал с докладом «Замеры производительности это непросто?» и сколько еще это будет продолжаться?

Гы-гы-гы. С докладом выступал не так

уж много – три раза в программе (GPW12, YAPC::RU12 и YAPC::EU13) и один раз на FOSDEM14, заменив больного докладчика. Кстати из за технических неполадок в последний раз получилось зачетное немое кино ;).

В нынешнем формате доклад вряд ли будет продолжаться. Может, если забреду на какую нибудь `.pm`-встречу. Когда дойдут руки до написания обертки, делающей все, о чем я говорил автоматически, может еще раз расскажу, но уже как блиц-доклад.

Помогает ли знание нескольких (человеческих) языков программированию?

Довольно трудно говорить о себе, так как сравнивать больше не с чем. Но в целом... это вообще отдельный длинный и сложный разговор. С одной стороны

мне повезло в детстве – я вырос сразу на двух языках и в добавку довольно рано «расколол» английский. Но с другой стороны, для половины западной Европы такое детство ничем не удивляет. И чтобы все запутать — я лично не нахожу очевидной разницы между американскими/российскими (подавляюще единственный язык) и европейскими (как правило, 2+ языка) коллегами.

Знание языков на беглом уровне наверное меняет мировоззрение и неизбежно процесс мышления. Как минимум, с малых лет понимаешь что слова лишь идентификаторы, никаким образом не определяющие сущность того, к чему их прицепили. Помогает ли это программированию в частности... вряд ли. Зато почти уверен: что помогает писать злобным сарказмом ;)

У кого, где и за сколько так поставил свой неповторимый акцент?

В университете жизни, в кредит :) Английский я выучил в основном сам, долго до того, как мне пришлось на нем говорить (я также выучил, как держать и перелистывать одной рукой, другая была занята мышью, передвигая Роджера Уилко, Индиану Джоунса, Гайбраш Трепвуда, или на худой конец короля Грехема). Потом семья переехала в Америку и буквально выходя из самолета я заговорил на почти том же странном языке, схожим с английским. Первые лет пять мне совершенно не приходило в голову чего нибудь исправлять в произношении... и с тех пор так все и осталось. Сегодня мне говорят, что у меня шикарный итальянский акцент, хотя скорее я просто правильно размахиваю руками во время разговора :)

■ *Вячеслав Тихановский*