**Системное программирование**

Современные проблемы

Мартынов Семён

*Санкт-Петербургский государственный Политехнический университет*

2 апреля 2015 г.

**Аннотация**

В системном программировании существуют два вектора раз- вития - обеспечение работы с аппаратурой и предоставление опре- делённых условий работы прикладному ПО. Я рассмотрю те слож- ности, которые возникли перед системными программистами за последние 7 - 10 лет. Некоторые из этих проблем актуальны и для программистов прикладных, но данной работе акцент сделан имен- но на разработке системного ПО.

*Ключевые слова:* системное программирование, сложность, параллель- ное исполнение, верификация

# Введение

Перед тем, как говорить о современных проблемах системного програм- мирования, нужно определить, что это такое. Традиционно, системное программирование связывают с разработкой программного обеспечения

(ПО), управляющего компонентами компьютерной системы. Это про- граммное обеспечение выступает своеобразным мостом, между аппара- турой и миром прикладного ПО. Если задачами прикладных программ является решение конкретных практических задач (того, что нужно поль- зователю), то системное ПО должно предоставлять прикладному опре- делённый уровень гарантий (того, что нужно программе пользователя), абстрагируя физические устройства. Со временем, объёмы и потребности прикладного ПО возросли, а вместе с тем возросли и требования к тому контексту, который обеспечиваются системным ПО. Другими словами, если раньше системное ПО было нацелено вниз, на уровень железа, опе- рационных систем и драйверов, то сейчас системное ПО получило допол- нительный вектор развития вверх, в сторону прикладного ПО, которому нужно обеспечить надёжность, переносимость и даже безопасность. За- дачей системного ПО стала работа с прикладными программами как с данными, т.е. анализ, выполнение, улучшение и проверка того, что за- пускает пользователь.

# Рост сложности кода

При традиционном рассмотрении вопроса, в первую очередь называются операционные системы. На моём ноутбуке установлена ОС Arch Linux, и ядро 3.18.3. Если запустить небольшой скрипт, который подсчитает ко- личество строк кода (в действительности этот скрипт не различает как таковой код, заголовочные файлы или строки из справочного файла, он считает все вместе) то получится 18 998 896. Это достаточно большое число само по себе, такой объём кода невозможно держать в голове, при этом необходимо этот код поддерживать (тестировать, находить и исправлять уязвимости, обеспечивать совместимость со старыми верси- ями) и развивать, добавляя новый функционал. Если посмотреть приме- чания к выпуску, то выяснится, что по сравнению с предыдущей версией

3.17 (представленной в августе прошлого года) данная версия ядра име- ет 11200 исправлений от 1300 разработчиков, а размер патча составляет 38 Мб! Изменения затронули 9307 файлов, добавлено 485719 строк кода, удалено 355945 строк, и это только за период с августа по декабрь!

Сайт openhub.net занимается сбором статистики по популярным откры-

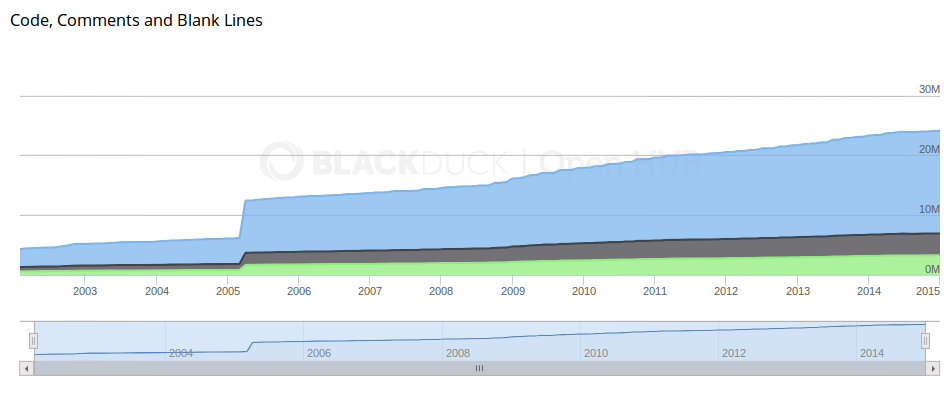


Рис. 1: Рост количества строк кода в ядре Linux (размер шага по оси абсцисс 1 год, по оси ординат 10 миллионов строк).

тым проектам, и на рисунок 1 видно, как изменялось количество строк кода ядра с течением времени (синим цветом показаны строчки кода, серым - строки с комментариями, зелёным - строки, используемы для оформления кода). Явно наблюдается восходящий тренд, объём кода бу- дет продолжать расти, а вместе с ним и сложность, которая ложится на плечи разработчиков.

Разработчиков тоже становится всё больше! На рисунке 2 мы видим, что количество разработчиков, вносящих свои изменения, приблизилось к 1 000 в месяц, а количество коммитов, которые они все вместе делают каж- дый месяц уже превысило отметку в 5 000 (см. рис 3). Важно помнить, что каждый такой коммит должен быть получен, изучен и принят (либо не принят) ментейнером, ответственным за тот или иной участок ядра.

# Экспоненциальный рост количества компьютерной техники

Помимо роста сложности кода, растёт и количество устройств, на кото- рых этот кода работает. Объем рынка мобильных телефонов в России увеличился за 2014 год на 18% до 254 млрд рублей, свидетельствую дан- ные группы "Связной". Было продано 26 млн устройств, 60% из них

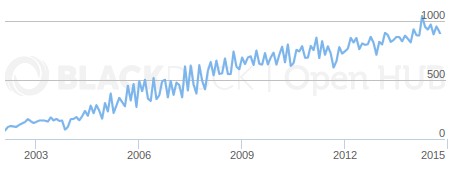


Рис. 2: Рост количества разработчиков ядра Linux.

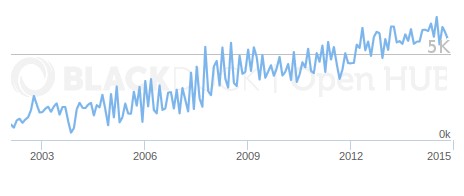


Рис. 3: Рост количества изменений, предлагаемых разработчиками.

пришлось на смартфоны. Помимо смартфонов, нас окружают ноутбуки, планшеты, умные телевизоры (и приставки к ним), домашние роутеры и даже системы "умный дом". А есть ещё компьютеры в автомобилях, банкоматах, турникетах метро... Всё это создаёт сложную мульти ком- понентную среду, состоящую из различных административных доменов. Всё это создаёт целый ряд проблем для системного программиста, среди которых выделяются сложность взаимодействия устройств (протоколы, физическая среда), обеспечение безопасности (безопасность пользовате- ля, безопасность данных), энергоэффективность (которая является ак- туальной проблемой сама по себе) и пр. Отдельной большой проблемой является взаимодействие человека и компьютера, но этот вопрос слиш- ком объёмный и изучается не только в области системного программи- рования.

Говоря о встраиваемых решениях, нужно упомянуть традиционные слож- ности, связанные с сильным ограничением на доступные ресурсы (ча- стоту процессора, объём оперативной и долговременной памяти, различ- ные интерфейсы связи) но это нельзя назвать новой проблемой. Всё это системное программирование проходило за последние 20 лет на архи- тектура x32/x64, а сейчас повторяет свой путь на различных версиях ARM/MIPS. Аппаратное обеспечение становится мощнее, а отладка тех- процесса позволяет существенно снизить его стоимость.

# Виртуальные машины

Производительность традиционных десктопных и серверных систем на данный момент достигла такого уровня, что позволяет запускать несколь- ко виртуальных машин одновременно. Изоляция машин может быть как полной (при этом эмулируется поведение процессора любой архитекту- ры) так и контейнерная (все экземпляры операционной системы исполь- зуют ресурсы одного ядра, при этом прикладной уровень изолируется различными пространствами имён). За управление ресурсами в этом слу- чае отвечает гипервизор, а системному программисту нужно решить как разделить какой-то ресурс (допустим, сетевую карту) так, чтобы ни одна виртуальная машина не испытав проблем. Работы по обеспечению вирту- ализации и изоляции уходят своими корнями в 80-е годы, но сейчас вир- туализация перешла с мейнфреймов на обычные пользовательские ком- пьютеры. В той или иной степени она используется даже антивирусами (для помещения подозрительного объекта в карантин) и веб-браузерами (когда требуется повышенный уровень безопасности).

Код Java так же выполняется в виртуальной машине. Подобный под- ход к построению программных систем имеет большую популярность, т.к. разработчики этой системы смогли успешно решить проблему про- граммной эмуляции абстрактной машины, которая обеспечивает перено- симость программ, полную изоляцию и даже верификацию кода в про- цессе исполнения. Самым удивительным является тот факт, что в неко- торых случаях JIT компиляция горячих участков обеспечивает среднюю производительность программы не уступающую нативному исполнению! На рисунке 4 показан график сравнения скорости выполнения одного и

того же алгоритма умножения матриц, реализованного на C и на Java. Но и в случае с Java есть ряд проблем, которые разработчикам виртуаль- ной машины ещё предстоит решить, к примеру работа с графикой. Так же, наивное использование Java не убережет пользователя от проблем с многопоточностью.

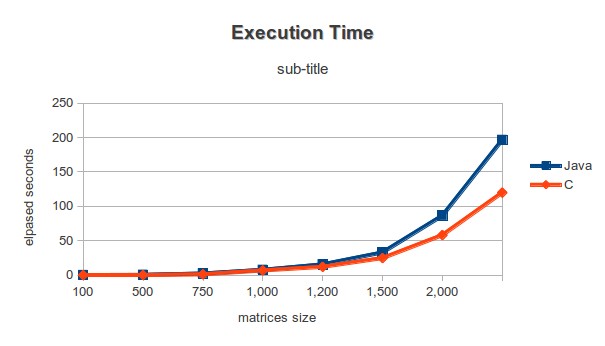


Рис. 4: Сравнение производительности виртуальной машины Java и на- тивно исполняемого кода (язык C) на примере задачи перемножения двух матриц.

# Многопоточное программирование

Параллельное и многопоточное исполнение кода стало трендом послед- них лет. Во многом это связано с реализацией аппаратной поддержки, получили широкое распространение многоядерные процессоры. Много- ядерные процессоры сейчас ставятся не только в сервера и десктопы, но даже в телефоны. Параллельная обработка многих задач (к примеру, обработка изображений) даёт значительно больший выигрыш в произво- дительности, чем банальное увеличение частоты центрального процессо- ра. Это подтверждает рисунок 5 (источник указан под изображением) -

начиная с 2005 года частота и потребляемая энергия замерли, но количе- ство транзисторов и производительность продолжили рост. Вместе с тем, это создаёт большое количество проблем, таких как гонки по данным, взаимные блокировки (дедлок и лайвлок), ресурсное голодание. Допол- нительно это сопровождается проблемами с аппаратным (когерентность кешей, общая память) и алгоритмическим уклонами (не все алгорит- мы хорошо поддаются распараллеливанию, а производительность парал- лельных проблем лимитируется законом Амдала).

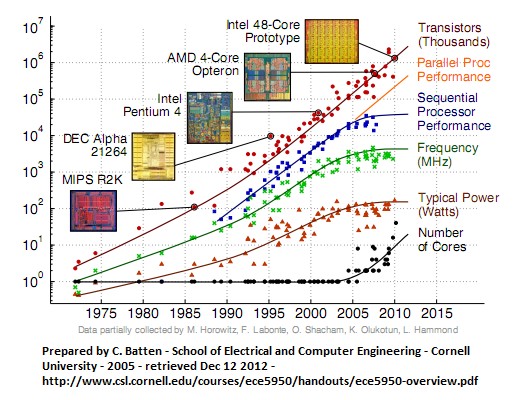


Рис. 5: Рост производительности микропроцессоров за счёт использова- ния параллельности.

Параллельные программы значительно сложнее поддаются тестирова- нию и отладке, т.к. они лишены такого важного качества как детерми- низм. Для демонстрации сложности можно изучить изображение 6. На нём показан граф вызовов, полученный в результате трассировки ма- ленькой программы hello world (его код приводится в листинге 1), ис- пользующей стандартную библиотеку MPI (код выполнялся на компи- ляторе gcc).

Listing 1: Пример простой многопоточной программы hello world

1 **void** main ( **int** argc , **char** *\** argv [ ] ) 2 {

3 **int** myrank , s i z e ; 4

1. MPI\_Init(& argc , &argv ) ;
2. MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myrank ) ;
3. MPI\_Comm\_size(MPI\_COMM\_WORLD, &s i z e ) ;
4. p r i n t f ( " P r oc e s s or %d o f %d : Hello World ! \ n" , myrank , s i z e ) ;
5. MPI\_Finalize ( ) ; 10 }

Очевидно, что те принципы обеспечения качества (основанные на стати- ческом анализе и ручном тестировании) и верификации кода, которые применялись раньше не подходят для параллельного программирования, и именно это мне кажется наиболее актуальной из современных проблем системного программирования.

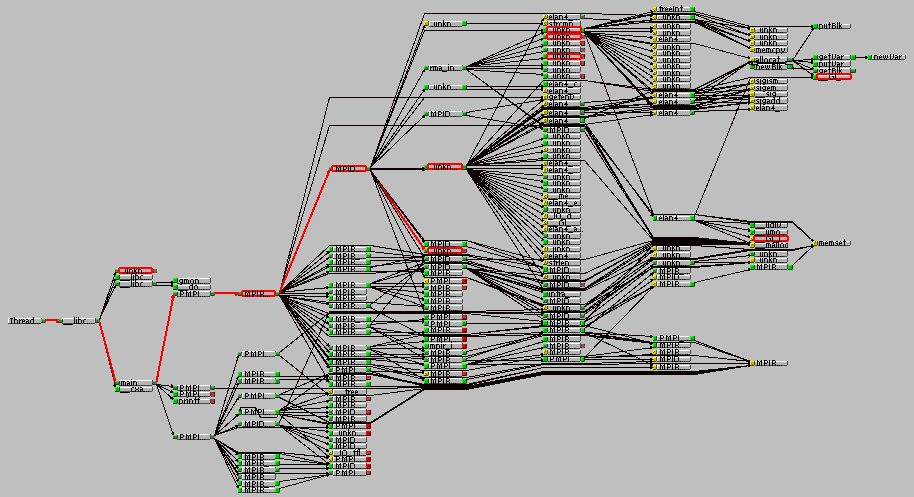


Рис. 6: Граф вызовов библиотек для простой параллельной программы hello world.

# Заключение

За последние 10 лет общее количество задач в Системном программи- ровании только увеличивалось. В то же время, для многих задач были сформулированы решения (такие как применение абстракции на всех уровнях), приятые всем сообществом.

В данной работе были упомянуты следующие проблемы:

* 1. увеличение объёмов кода
  2. рост количества "умных устройств"вокруг нас
  3. взаимодействие различных типов устройств
  4. разнообразие платформ
  5. обеспечение безопасности
  6. энергоэффективность
  7. взаимодействие человека и компьютера
  8. виртуализация и построение гипервизоров
  9. виртуальная абстрактная машина для выполнения кода
  10. эффективная и безопасная работа параллельных программ

На мой взгляд самой острой проблемой в данный момент является реше- ние последней проблемы, т.к. параллельное программирование применя- ется уже достаточно широко, при этом многие разработчики указывают именно эту часть своей работы как основной источник ошибок.