## Введение

В этой статье рассматривается алгоритм синхронизации времени внутри секунды, его назначение, а так же приводится описание существующего в ядре Linux алгоритма и сложности, возникающие при его реализации.

Рассмотрим следующую систему: имеется машина с установленной Linux-based операционной системой и различными подключенными внешними устройствами. На машине запущены одна или несколько программ, которым для работы требуется, чтобы системные часы шли согласно некоторому эталонному источнику времени. Например, такой программой может быть драйвер сетевого адаптера, который должен регистрировать приходящие в систему извне пакеты данных и ставить в соответствие каждому из них некоторую метку времени, а источником могут быть часы другой системы, или одно из устройств, подключенных к рассматриваемой системе.

То есть, таким программам требуется, чтобы системные часы были *синхронизированы* с эталонным источником. Это означает, что в каждый момент времени значения часов источника и того, кто с ним синхронизируется, должны как можно меньше отличатся. Чем меньше разница, тем выше точность синхронизации.

Источников, как и программ, нуждающихся в синхронизации, может быть несколько. Для синхронизации значений часов системы может быть использован NTP (Network Time Protocol), дающий относительно неплохую точность синхронизации, порядка миллисекунд. Но иногда такой точности недостаточно, и тогда помогают алгоритмы, точность которых достигает порядка наносекунд.

Рассматриваемый далее в статье алгоритм позволяет с помощью некоторого высокоточного генератора PPS (Pulse per second) сигналов узнавать о начале каждой новой секунды и сообщать об этом системе, в которой, в свою очередь, регистрируются приложения - PPS-потребители, которым требуется реагировать на эти сигналы. Каждый PPS-сигнал вызывает в системе рассылку текущего значения системных часов всем зарегистрированным потребителям.

То есть для получения высокой точности можно использовать какой-нибудь относительно грубый алгоритм, вроде NTP, в совокупности с PPS. И тогда если время будет синхронизироваться с помощью основного алгоритма до хотя бы секунд, то уже внутри секунды синхронизацию обеспечит PPS.

Рассмотрим далее более конкретно, что такое метки времени, откуда берется PPS-сигнал, и что это такое, какие сложности возникают при реализации алгоритмов, работающих с PPS, а так же конкретный пример - алгоритм, использующийся в ядре Linux для синхронизации системного времени с PPS-источником.

## Время в ядре Linux

В компьютере время представлено как значение некоторого постоянно возрастающего аппаратного счетчика, значение которого увеличивается через примерно равные промежутки времени. Счетчик не может инкрементироваться через просто равные промежутки времени, так как частота его инкрементации связана с частотой некоторого генератора, а его частота колеблется в относительно небольшом диапазоне, зависящем от условий, в которых находится генератор, например, от давления и температуры, от энергопотребления и других причин.

Значение этого счетчика в каждый момент времени - целое число, которое используется системой для перевода его в человеческие единицы измерения времени. Говоря о человеческих единицах времени будем здесь и далее иметь ввиду наносекунды. Для перевода значения счетчика в наносекунды требуется знать, с примерно какой частотой счетчик инкрементируется - для каждой системы устройств эта частота известна заранее.

Значение этого счетчика - это так называемое *raw monotonic time -* "сырое" значение времени. Так же в системе есть понятие *real time -* это время, получаемое из raw monotonic с помощью различных коэффициентов пересчета. Применение алгоритмов синхронизации времени - это способ улучшать эти коэффициенты для получения более точного времени. real time - это то время, которое доступно в пользовательской области операционной системы для приложений вне ядра, и которое обычно называется *системными часами*.

Когда говорят о коррекции времени, то имеют ввиду не прямое изменение значения аппаратного счетчика, а способ его преобразования в real time из raw monotonic time. То есть алгоритмы синхронизации служат для построения правил, по которым вычисляются коэффициенты для пересчета raw monotonic time.

Как инкрементируется значение счетчика времени? Для этого существует несколько механизмов, вот одни из них:

1. Аппаратные прерывания
2. Time stamp counter (TSC)
3. High precision event timer (HPET)

Прерывание - это сигнал, сообщающий процессору о наступлении какого-либо события, управление при возникновении прерывания передается его обработчику. Прерывание может быть как аппаратным, в случае счетчика времени, так и программным, если оно инициировано специальной инструкцией в коде. То есть через примерно равные промежутки времени счетчик инкрементируется с помощью аппаратного прерывания. Но этот способ устарел, поскольку обработка прерывания - это дополнительная нагрузка на систему.

Time stamp counter - специальный регистр процессора, который считает количество тактов, прошедшее с запуска компьютера. Его значение можно получить специальной инструкцией в коде.

HPET - аппаратный таймер, используемый в персональных компьютерах. Фактически, это отдельное устройство, подключенное к процессору. Метку времени можно читать с его регистров. Так же это устройство можно программировать на прерывания.

Выше мы рассматривали систему, часы которой надо синхронизировать, и устройство, являющееся источником PPS сигналов. У системы есть свой счетчик времени, значение которого менять нельзя, но можно менять коэффициенты и правила, по которым считается real time. Вычислением таких коэффициентов и занимается алгоритм синхронизации, использующий PPS.

## Pulse per second

Pulse per second (PPS) - сигнал, генерируемый некоторым высокоточным источником (PPS-source) с частотой максимально близкой к одной секунде. Для рассматриваемого в статье алгоритма важно уметь "ловить" моменты, когда возникает PPS-сигнал, то есть получать метки времени точно в эти моменты.

PPS-источник подключается к вычислительной системе посредством контакта интерфейса последовательной линии (modem-control pin on a serial-line interface) или параллельной . Система узнает о возникновении сигнала, получая прерывание, и как можно скорее запоминает текущую метку времени. Далее эта метка становится доступна с относительно небольшой задержкой после ее получения для ПО, работающего со временем - PPS-потребителей.

PPS-потребители - это приложения, заинтересованные в обработке возникающих PPS-сигналов, которые ранее зарегистрировались в системе, как получатели PPS-сигналов. С помощью метки, полученной с PPS-сигналом, PPS-потребитель может, например, получить расхождение между источником точного времени и системными часами.

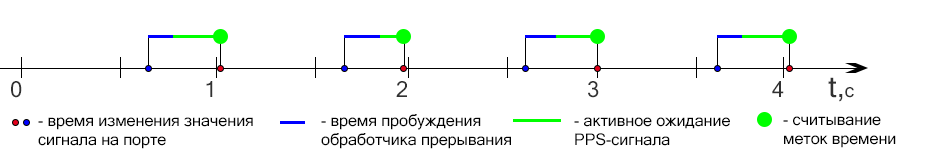
К одной системе может быть подключено несколько PPS-источников, каждый из которых генерирует собственные PPS-сигналы независимо от других. Когда очередной PPS-потребитель регистрируется на прием сигналов, он может указать, какой PPS-источник ему нужно слушать, причем слушать можно несколько источников.

Кроме пользовательских программ в ядре есть еще один потребитель сигналов: специальная процедура ядра - *hardpps*. Эта процедура на основании получаемых сигналов пересчитывает коэффициенты, по которым monotonic time пересчитывается в real time, в результате чего изменяется системное время. Причем эта процедура может слушать единовременно только один PPS-источник. Как раз с ним рассматриваемая система и будет синхронизировать свои часы.

Рассмотрим, какие возникают сложности при реализации алгоритмов синхронизации, основанных на PPS.

1. Как правильно учесть время, которое пройдет с момента появления сигнала и возникновения прерывания до вызова его обработчика и запоминания текущего времени для его передачи PPS-потребителям?
2. Как использовать получаемые метки для коррекции системного времени в процедуре hardpps, описанной выше?

Рассмотрим, в чем суть первой задачи, с помощью следующего рисунка:



На рисунке изображена шкала времени, разделенная по секундам. На нем схематически изображена схема получения меток времени по PPS-сигналу.

В обработке PPS-сигналов важны не сами PPS-сигналы, а моменты времени, в которые они появляются - каждому сигналу надо как можно точнее поставить в соответствие метку времени.

Если бы прерывание возникало непосредственно в момент времени возникновения PPS-сигнала, то на пробуждение обработчика прерывания и само считывание меток уходило бы слишком много времени, причем эти временные затраты были бы непредсказуемы на каждом новом сигнале, то есть время пробуждения обработчика и считывания было бы каждый раз разным, что зависело бы от некоторых факторов, например:

* уровень энергопотребления процессора (процессор постоянно пытается его снизить). Если энергопотребление снижено, то необходимо время, чтобы "пробудить" процессор.
* если в данный момент в ядре запрещены прерывания.
* могут возникать более приоритетные прерывания.

В качестве решения используется следующий алгоритм: источник PPS-сигнала изменяет значение электрического сигнала, посылаемого на порт, немного раньше, до возникновения самого PPS-сигнала. Система, которая должна получить сигнал, вызывает прерывание в ответ на это изменение. На рисунке эти моменты обозначены красными точками на шкале. Затем обработчик прерывания просыпается - синие линии на рисунке - и переходит в режим активного ожидания PPS-сигнала - зеленые линии на рисунке. За PPS-сигнал принимается обратное изменение значения электрического сигнала на порте, - красные точки на рисунке - но в этот момент времени обработчик прерывания находился в активном ожидании этого события и может сразу считать метки времени - зеленые точки на рисунке.

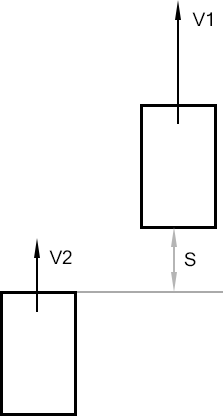
В итоге, если подготовится к сигналу пораньше, то почти всегда метка будет считана достаточно точно. Далее рассмотрим вторую из задач, описанных выше, решение которой и является рассматриваемым в статье алгоритмом.

## Алгоритм коррекции времени

Для разбора алгоритма коррекции времени необходимо знать об алгоритме NTP, как он работает, и какие проблемы призван решить.

Network Time Protocol (NTP) - алгоритм, используемый для синхронизации часов системы с одним или несколькими удаленными серверами посредством сети. Реализация NTP - это процесс-демон, работающий в пользовательском пространстве операционной системы, не в ядре. Поскольку синхронизация времени требует вмешательства ядра для коррекции коэффициентов пересчета raw monotonic time в real time, то для этого используется механизм системных вызовов.

Какие проблемы могут возникнуть с часами, которые решает алгоритм NTP? - проблема расхождения фазы часов, и частоты хода часов. Чтобы лучше понять, в чем суть этих двух расхождений, можно рассмотреть следующий пример:

Пусть есть дорога, по которой движутся два автомобиля. Пусть они так же движутся с разными скоростями. Тогда расстояние между ними - S - будет постепенно увеличиваться. Если провести аналогию с часами, то автомобили - это разные часы, их скорости - это частоты хода часов, а S - это постоянно возрастающая разница показаний этих часов.

В данном случае просто скорректировать S недостаточно - оно все равно будет снова возрастать из-за разных частот. Поэтому здесь требуется коррекция именно частот хода часов.

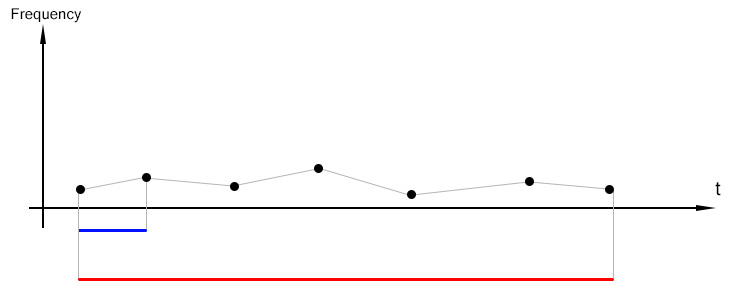
Что значит ошибка фазы? - пусть снова есть два автомобиля, которые движутся с одинаковыми скоростями рядом друг с другом. Пусть теперь один из автомобилей, сохраняя прежнюю скорость вращения колес, пробуксовал на льду, немного отстав от второго автомобиля. Скорости автомобилей остались прежними, но дистанция между ними стала S. Если провести аналогию с часами, то здесь произошла ошибка фазы S. Исправлять такие ошибки фазы можно за один раз, до следующего сбоя ("пробуксовки").

Таким образом, алгоритм NTP исправляет сразу две ошибки: ошибку фазы, и ошибку частоты. Для этого используется множество коэффициентов, которые обновляются по мере работы алгоритма.

Часть этих коэффициентов обновляется специальной процедурой - *hardpps*. Эта процедура является частью ядра, а не демона NTP, работающего в пользовательском пространстве, и вызывается при каждом возникновении PPS-сигнала, чтобы обновить параметры алгоритма NTP, которые используются для вычисления коэффициентов пересчета raw monotonic time в real time.

Алгоритм коррекции времени состоит из двух самостоятельных частей: первая корректирует фазу, а вторая частоту, и называются они соответственно Phase-Locked-Loop и Frequency-Locked-Loop.

В то время, как фаза корректируется каждую секунду, частота корректируется реже, раз в секунд. - это один из параметров алгоритма, который называется *длина интервал частоты*, и он меняется по мере работы алгоритма в некотором диапазоне. Зачем для частоты нужен именно относительно длинный интервал, а не просто две соседних секунды, как с фазой?



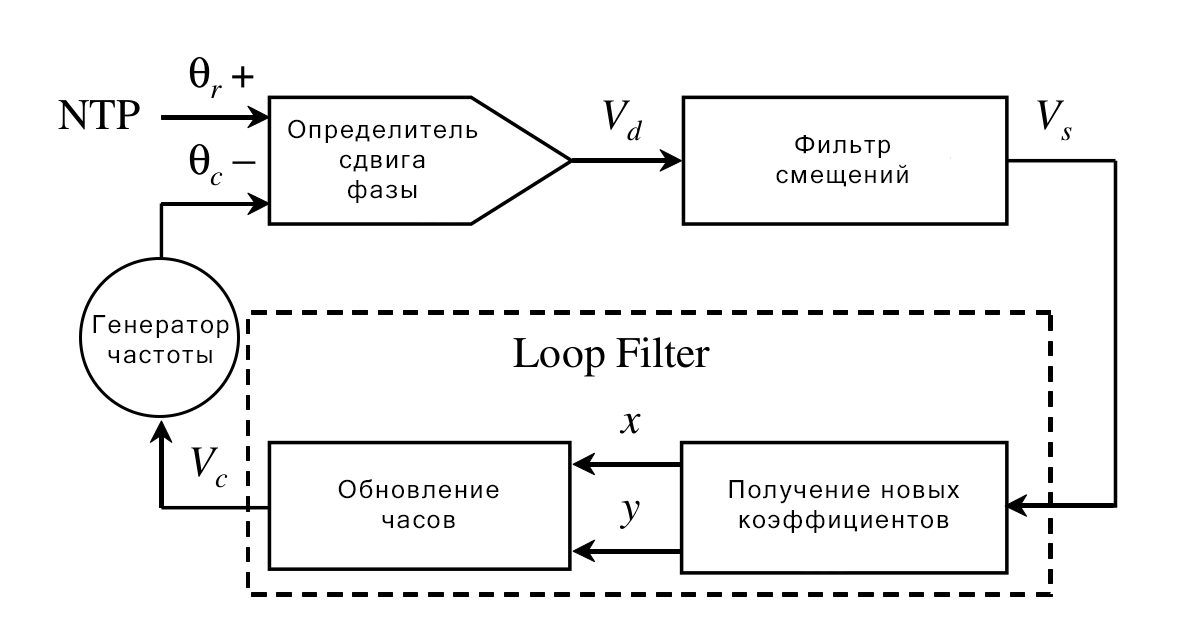
Что вообще такое частота хода часов в терминах алгоритма NTP? Это насколько единиц увеличивается real time за одну секунду. Поскольку частота хода часов зависит от частоты некоторого аппаратного генератора, то она постоянно колеблется, и вычислять ее по двум соседним секундам - на рисунке это синяя полоса - может быть ненадежно. То есть для получения более среднего значения частоты используют более длинные интервалы - красная линия на рисунке выше.

Когда частота становится менее стабильной, то интервал частоты уменьшается, так как частота нуждается в более частой корректировке, а когда частота стабильна, то обновлять ее можно относительно редко.

Вопрос - что может служить причиной ошибок фазы и ошибок частоты? Ошибки фазы возникают обычно из-за некоторых программных причин, например, это могут быть различные задержки в операционной системе. А ошибки частоты возникают из-за колебаний температуры генератора частоты, перепадов напряжения. NTP умеет определять, какие ошибки в системе преобладают, и использует исправления частоты и фазы с разными весами.

На выходе такого комбинированного алгоритма получаются коэффициенты, которые используются для коррекции времени.

Алгоритм работает, как замкнутая система управления *(feedback control system)* следующего вида:

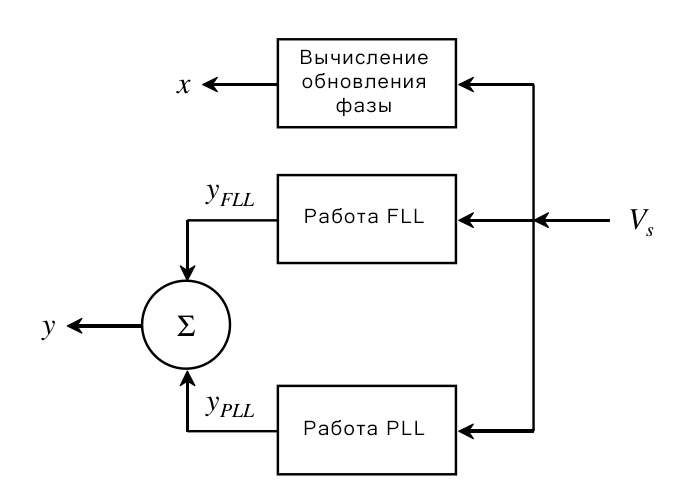


представляет эталонное время, - системное время. Сдвиг определяется как разница этих значений .

Фильтр смещений отсеивает неправдоподобные значения . Так же в фильтре NTP использует различные алгоритмы выбора, кластеризации и комбинирования данных через несколько фильтров, чтобы получить наилучшее смещение . В случае использования hardpps вместо внешних серверов с эталонным временем этот фильтр просто отбрасывает заранее невозможные смещения, например, слишком большие.

Далее алгоритм вычисляет смещения частоты и фазы - *y* и *x,* которые уже используются для генерации сигнала , который каждую секунду корректирует сдвиг фазы системы, используя вычисленные коэффициенты.

Как считаются *x* и *y*?



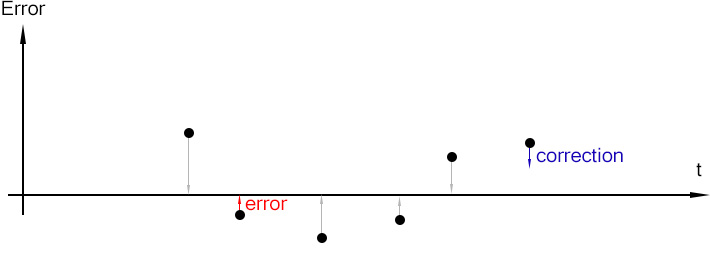
Здесь - функции над и *x*, они будут рассмотрены подробнее далее. *x* - называется остаточной ошибкой фазы, и используется для ежесекундной коррекции фазы.

Обобщая работу NTP - цель состоит в том, чтобы как можно сильнее уменьшить к следующему шагу работы алгоритма.

Рассмотрим роль процедуры hardpps в этом алгоритме. *hardpps* - так называется специальная процедура ядра, служащая для синхронизации системных часов с некоторым PPS-источником, и реализована она как С функция в коде ядра. При этом hardpps является частью реализации алгоритма NTP, но находящейся именно в ядре. hardpps вызывается при каждом возникновении сигнала и работает с двумя аргументами - метками времени real time и raw monotonic time, запоминаемыми в момент получения сигнала.

На каждом вызове hardpps корректирует параметр алгоритма NTP - смещение фазы времени. Для этого вычисляется количество наносекунд в текущей секунде метки real time - обозначим как *nsecs.* Алгоритм во время работы использует и обновляет *кольцо* последних значений nsecs. На каждой итерации из кольца выбирается некоторое наилучшее значение - *correction*.

Алгоритм выбора correction может быть реализован по разному, например, в оригинальном ядре используется выбор просто последнего положенного в кольцо значения. Так же есть более сложные алгоритмы: например, выбор медианы из значений кольца, или среднего значения, но у медианного способа сходимость слишком медленна, а среднее значение неустойчиво к сильным отклонениям. В качестве альтернативы есть так называемый "смарт фильтр".



Пусть черные точки - это последние отклонения, записанные в кольцо. Смарт фильтр для вычисления correction использует минимальное из абсолютных значений кольца - здесь это error, а в качестве знака - противоположный от последнего значения кольца, и в итоге получается correction. Смарт фильтр обеспечивает хорошую сходимость, и не чувствителен к разовым сильным отклонениям - выбросам.

Далее используется фильтр, который проверяет, насколько ошибка nsecs изменилась с прошлой итерации. Если ошибка стала слишком велика, то корректировка времени не происходит. Иначе время обновляется на значение correction, но не сразу - это исправление "распределяется" по секунде, чтобы не было резкого скачка. В оригинальном ядре, например, в течение секунды тысячу раз применяется по одной тысячной от correction.

Коррекция параметров NTP, отвечающих за частоту, происходит следующим образом:

1. Вычисляется скорость хода часов, как отношение изменения raw time с последнего обновления параметров к длине этого интервала обновления.
2. Полученное отношение выставляется как обновление частоты

Как конкретно вычисляются *x, y,* и , и как выглядит и меняется ?

- интервал частоты, рассмотренный ранее, имеет вид , где уже меняется в некотором интервале. Соответственно, когда частота становится нестабильна, то уменьшается на единицу вплоть до некоторой нижней границы, а когда частота стабильна, то увеличивается.

Более сложно выглядят формулы других параметров алгоритма:

Стоит отметить, что на каждом обновлении считается в предположении, что на следующем обновлении будет нулем. А если это не так, то надо уменьшить на эту величину, чтобы исправить только влияние частоты.

Когда заканчивается очередной интервал частоты, то .

Каждую секунду, то есть в течение очередного интервала частоты, процесс коррекции времени вычисляет обновление для фазы и новый . Величина используется ядром для коррекции фазы. Так продолжается до следующего обновления, когда и будут пересчитаны заново.

## Заключение

## Алгоритм хоть и работает, но он несовершенен и может быть улучшен. Например, размер кольца для последних nsecs можно увеличить, и усовершенствовать алгоритм выбора из него значения correction - можно, например, брать среднее арифметическое от элементов кольца, отбросив самые большие и самые маленькие.

## Список литературы

1. Cochran R., Marinescu C., Riesch C. Synchronizing the Linux system time to a PTP hardware clock //Precision Clock Synchronization for Measurement Control and Communication (ISPCS), 2011 International IEEE Symposium on. – IEEE, 2011. – С. 87-92.
2. Mogul J. C. et al. Pulse-per-second api for unix-like operating systems, version 1.0 //Pulse. – 2000.
3. Gleixner T., Niehaus D. Hrtimers and beyond: Transforming the linux time subsystems //Proceedings of the Linux symposium. – 2006. – Т. 1. – С. 333-346.
4. Köker K., Hielscher K. S., German R. A Low-Cost High Precision Time Measurement Infrastructure for Embedded Mobile Systems //Robot Motion and Control 2007. – Springer London, 2007. – С. 445-452.