ИСПОЛЬЗОВАНИЕ ПОТОКОВ И СУБПРОЦЕССОВ В RUBY

Петух И.И.

Институт информационных технологий БГУИР,
г. Минск, Республика Беларусь

Образцова О.Н. – и.о. зав. кафедрой ИСиТ, к.т.н., доцент

Рассматриваются результаты временных тестов для случаев использования потоков и подпроцессов в языке Ruby

О языке Ruby часто говорят две вещи: Ruby медленный и имеет GIL. Однако лишь немногие знают, как лучше писать параллельные алгоритмы на Ruby.

Мнение большинства состоит в том, что нет смысла использовать параллелизм, поскольку все потоки выполняются один за другим (благодаря GIL). Однако это не так и это можно продемонстрировать двумя способами:

1. Собственные потоки ОС, потоки с GIL, где переключение происходит в соответствии с событиями ввода-вывода
2. Форки, которые создают отдельный подпроцесс

Оба метода похожи по реализации: входящий файл делится на несколько потоков по размеру; каждая нить содержит смещение, которое используется для процесса поиска.

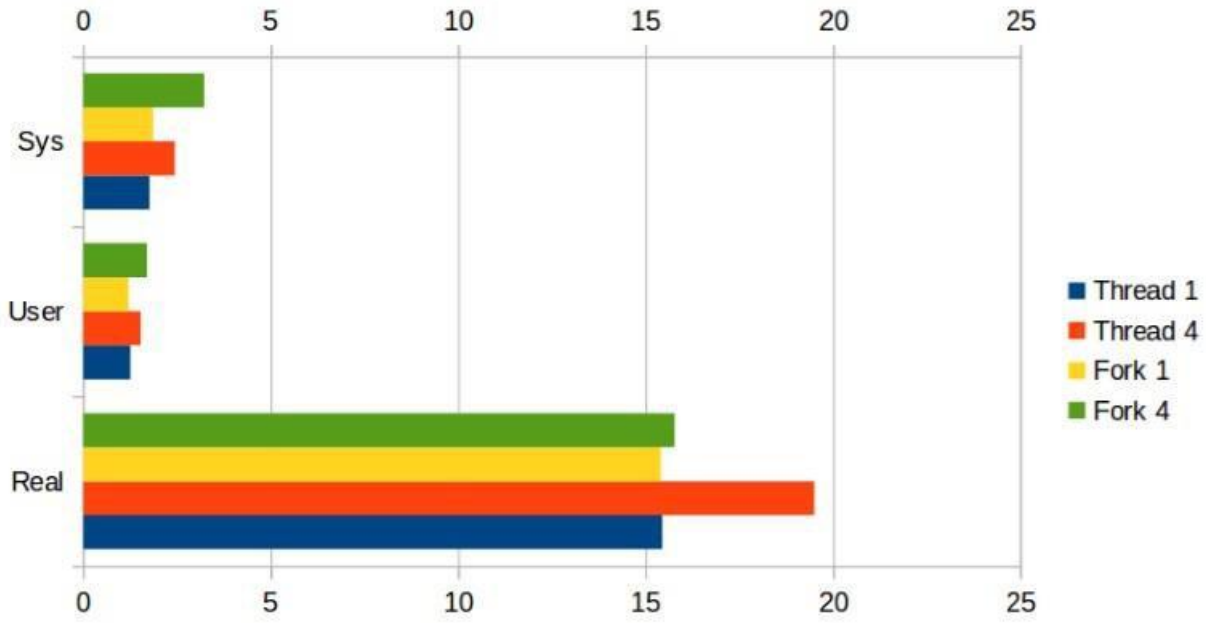
Форк создает подпроцесс, поэтому синхронизация становится действительно проблематичной.

Для тестирования будем использовать файл со случайными данными размером 1 ГБ.

Конфигурация:

1. Файл хранится на жестком диске
2. Данные извлекаются в SDD
3. Процессор: Intel (R) Core (TM) i3-2130 CPU @ 3,40 ГГц, 2 ядра + Hyper-Threading
4. Размер фрагмента для чтения: 5Mb

Результаты теста:

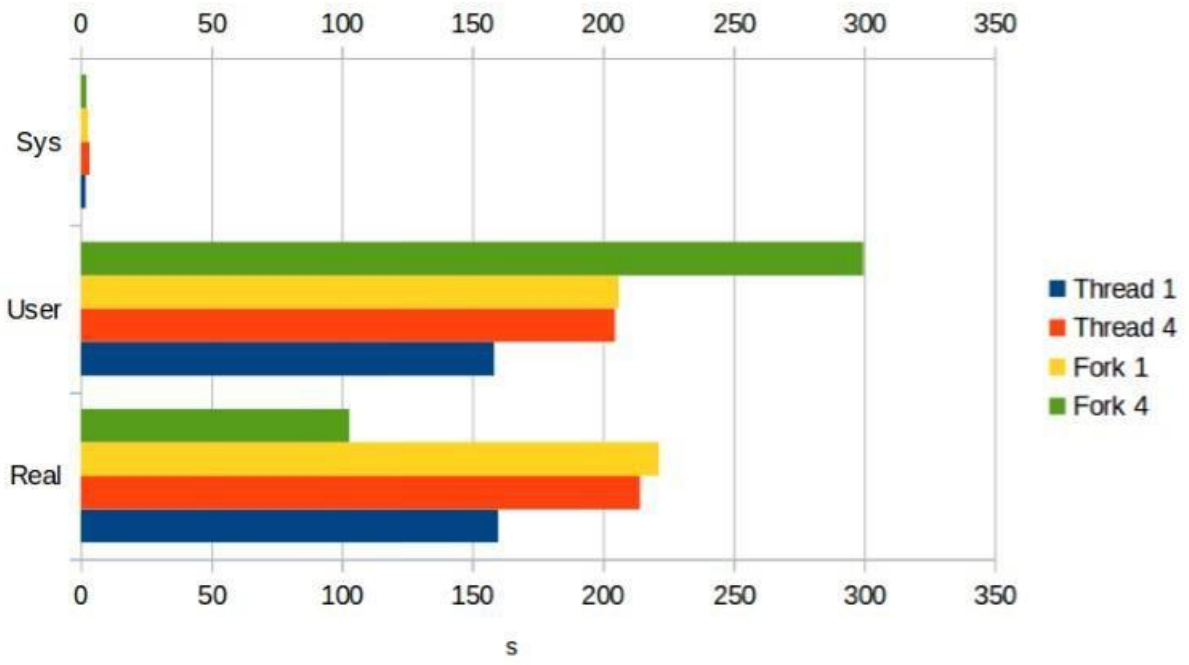


Во всех случаях нам не удалось сэкономить время. Соединение HDD-SSD - это наша так называемая «бутылочная горлышко», то есть файлы обрабатывались настолько быстро, что на самом деле это было похоже на простое копирование. В результате было невозможно получить более быстрые результаты.

Форки

Мы добились бы хороших результатов, если бы процессоры тратили больше времени на работу с потоком. Итак, давайте добавим код для его загрузки. Лучшее решение - использовать модуль bcrypt. Теперь мы уменьшим размер читаемой части до 5 Мб и добавим нагружаем процессор вызовом: 10.times {BCrypt :: Password.create ('secret')}

Мы получим следующую корреляцию обработки части файла / вычисления времени: 0,169 / 0,0883 с.

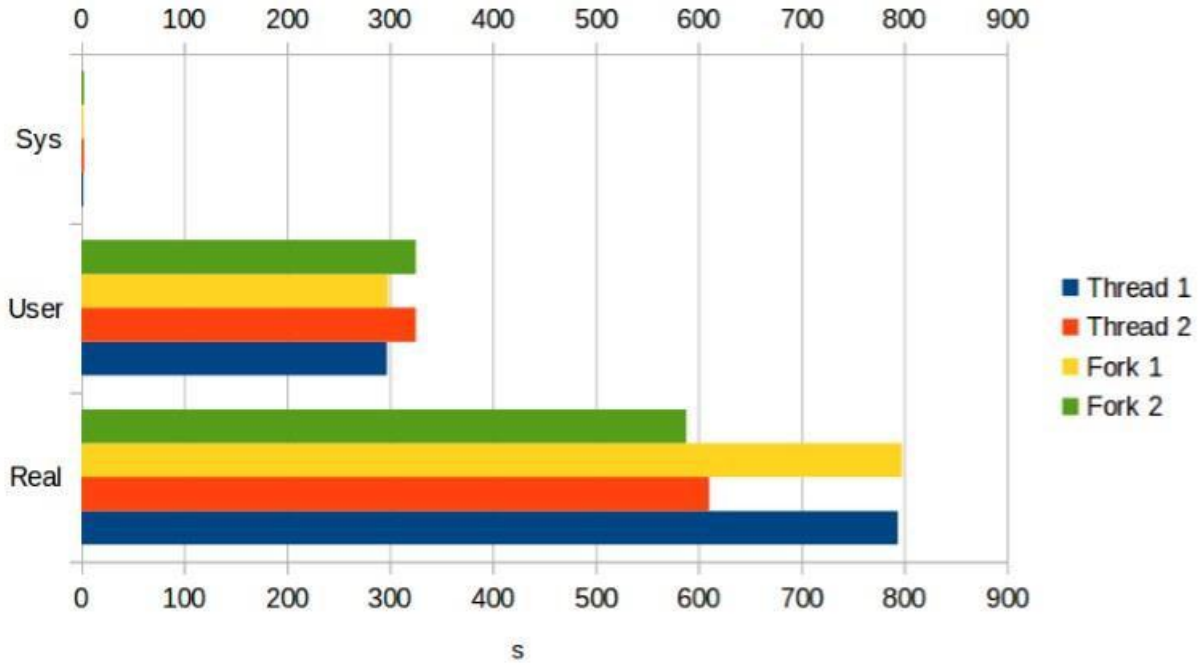


Теперь мы преимущества использования форков очевидны, и хотя 4 отдельных процесса загружают процессор, мы достигаем лучших результатов. Потоки не дают нам ничего полезного из-за GIL, поэтому у нас одновременно работает только один поток, который переключает выполнение процесса при чтении каждой части.

Потоки

Мы должны понимать, что мы вряд ли опередим форк из-за постоянного переключения контекста. GIL переключает обработку потоков при каждом событии, поэтому, если мы хотим добиться успеха, нам нужно, чтобы IO выводил нашу бутылочную горловину. Таким образом мы сможем продлить время обработки bcrypt на IO .Самым простым способом, будет работа напрямую с картой памяти, в качестве медленного IO устройства.

Поскольку скорость карты памяти очень низкая, мы уменьшаем количество потоков до 2 и добавляем fsync для записи файла. Эта опция указывает на немедленную запись данных без кэширования.



Форки и потоки имеют схожие результаты, и время выборки bcrypt было частично скрыто при вводе / выводе.

Заключение

Если наш алгоритм использует процессор + память, в этом случае нам могут помочь только форки. Тем не менее, мы можем получить переполнение памяти, а так же необходимость управлять синхронизацией потоков.

Если у нас есть «бутылочное горлышко» - входящий IO, то потоки позволяют нам частично скрывать время обработки за событиями IO. Кроме того, синхронизация данных достигается легко. GIL позволяет нам не беспокоиться об управлении потоками, потому что в один момент времени существует только один поток, выполняемый за раз.

Однако если существуют проблемы с IO, нет смысла использовать как форки, так и потоки.

Список использованных источников:

1. The fork() system call [Электронный ресурс] – Электронные данные – Режим доступа: http://www.csl.mtu.edu/cs4411.ck/www/NOTES/process/fork/create.html – Дата доступа: 07.02.2019.

2. Ruby Multi-Threading [Электронный ресурс] – Электронные данные – Режим доступа: https://www.tutorialspoint.com/ruby/ruby_multithreading.html – Дата доступа: 07.02.2019.

3. Nobody understands the GIL [Электронный ресурс] – Электронные данные – Режим доступа: https://www.jstorimer.com/blogs/workingwithcode/8085491-nobody-understands-the-gil – Дата доступа: 07.02.2019.