**Системы реального времени**

Курсовая работа

Проектирование и разработка клиент-серверной системы, содержащей собственный макет RTS “GateWay”

Проверил: Федоров С.А.

Выполнил: Пронин Д.В.

Группа: 5084/14

# Постановка задачи

В рамках данного курса мною была придумана система, в которой можно было бы задействовать один из основных критериев, отличающих системы реального времени от других систем – это жесткое время (оно может выражаться по-разному в зависимости от требований: это может быть реально жесткое время, не больше и не меньше заданного значения, или же просто не больше, чем заданное значение, либо не меньше, чем заданное значение с каким-то позволительным для системы запозданием и так далее) выдачи результата или сервиса.

Общая задача по-просту может быть рассмотрена как получение результата сложения двух результатов умножения матрицы на вектор: e = A\*b + C\*d. Для усложнения и приближения задачи к реальной, я придумал следующие характеристики системы:

* в системе присутствует RTS “GateWay” и 5 клиентов.
* 2 клиента передают элементы соответственно матриц A и C
* 2 клиента передают элементы соответственно векторов b и d
* выше указанная передача элементов осуществляется в систему GateWay, которой необходимо обработать элементы, выполнить необходимые операции
* результат работы RTS необходимо отправить пятому клиенту “Станция Назначения”
* передача элементов матриц и векторов происходит в случайное время в рамках некоторого промежутка времени. Это время передачи распределено по равномерному закону
* элементы матриц и векторов передаются не по порядку, а в разнобой, так же равномерно вероятно выбирая из массивов элементы
* RTS и каждый клиент – это отдельно существующий не зависящий от других процесс, поэтому для взаимодействия необходимо использовать средства IPC
* выполнение операций запроса элементов для дальнейшего подсчета результатов и отправки пятому клиенту происходит по инициативе пятого клиента (то есть станции назначения). Станция назначения посылает запрос системе Gateway на определенную компоненту результирующего вектора и уходит в ожидание на определенное жесткое время, после чего опрашивает RTS о результате. При правильной работе RTS она должна выдать результат уже проведенной совокупности операций за то время, которое давалось RTS. В этом смысл жесткого времени исполнения
* RTS по запросу станции назначения с полученными параметрам запроса начинает опрашивать клиенты, которые должны вернуть по запросу RTS необходимые элементы для проведения операции. Чтобы получить один компонент результирующего вектора, необходимо запросить по одной строке каждой из матриц и по вектору. У каждого клиента-матрицы запрашивается по строке, а у каждого клиента-вектора запрашивается вектор. При этом клиенты возвращают необходимые элементы строки матрицы и векторов по одному элементу в течение случайного промежутка времени. Они передают эти элементы до тех пор, пока есть что передавать в строке матрицы или в векторе.
* Если RTS не справляется со своей задачей (не имеет значения по какой причине – то ли из-за медленных ее клиентов, то ли из-за собственных медленных алгоритмов подсчета результата), то генерируется исключение и программа моделирования завершается с кодом ошибки «Ошибка реального времени»
* RTS имеет возможность подсчитать результат быстрее, чем это требует станция назначения, но не позднее, чем этого требует станция назначения

# Реализация

**Destination Station**

**VClient d**

**VClient b**

**MClient C**

**MClient A**

**RTS**

**“GateWay”**

## Клиент “Mclient A”

Клиент передает элементы матрицы А. По запросу от RTS с параметром «передай элементы такой строки», беспорядочно через промежутки времени, распределенные по равномерному закону, передаются элементы этой строки. После окончания передачи снова ожидается команда от RTS. Связь происходит через юникс-сокеты, в самом начале устанавливается соединение с RTS. Клиент читает матрицу из файла, сохраняет у себя в памяти и начинает процесс общения с RTS, ожидая от нее команд.

## Клиент “Mclient C”

Аналогично клиенту, работающему с матрицей A, но только здесь клиент работает с матрицей C

## Клиент “VClient b”

Клиент передает элементы вектора b. Так же связан с RTS с помощью сокетов, так же получает команды от RTS, только этот клиент не нуждается в указании от RTS каких либо параметров вместе с командой, кроме самой команды, так как в любом случае по типу команду клиент знает, что делать: передавать вектор или заканчивать работу. Чтение элементов вектора из файла.

## Клиент “VClient d”

Аналогичен клиенту, работающему с вектором b, только этот клиент работает с вектором d.

## Клиент “DestStation”

Задача этого клиента – получение результатов по компонентам результирующего вектора, за компонентами необходимого вектора станция обращается к RTS “GateWay” и совершенно ничего не знает о других клиентах, ее это не интересует. Интересует только то, что RTS декларирует, что результат будет получен не более, чем за указанное (требуемое) время отклика. По результатом каждой проведенного запроса к RTS на стандартный вывод пишется вектор, которым на данной итерации обладает станция. Если результат не получен вовремя, но генерируется фатальная ошибка реального времени и работа прекращается. Соединяется и общается с RTS при помощи сокетов.

## RTS “GateWay”

Связующее звено во всей системе. Является сервером для всех остальных частей, создает серверный сокет и прослушивает на попытку соединения. При удачной попытке соединения, отправляет запрос на тип клиента, получает тип и регистрирует в своей системе клиент с полученным от него типом, а так же добавляет дополнительные параметры: регистрационный номер, обработчик сообщений от клиента, флаг «зарегистрирован», дескриптор сокета уникального соединения для опознания клиента, от которого пришло сообщение.

После установки соединения, отсылает команду “Start” клиенту “DestStation”, и рабочий процесс начинается. Все клиенты, отсылающие компоненты векторов и матриц, ожидают от RTS каких-либо команд, а инициатором того, чтобы RTS эти команды начал отсылать, является станция назначения. RTS ждет от станции команды, как только команда от станции получена и идентифицирована, происходит следующее:

* создается дополнительный (второй) поток на обработку задач, вставших в очередь задач системы. Да, система обладает планировщиком (пока примитивным, но его можно расширить до поддержания приоритетов, пока планировщик «пихает» в очередь в том порядке, в какой пришли задачи). Планировщик управляет очередью задач, в системе определяется список задач, которые RTS может выполнять и загружать в очередь, из которой потом диспетчер, обладающий вторым потоком процесса, поочередно выгружает задачи и выполняет их. Сами задачи могут ставить в очередь другие задачи после выполнения некоторых условий и операций. Это способствует эффективной обработке данных и получению результирующих данных с большой скоростью.
* Если в очереди задач нет и не выполнена последняя задача, которая определяется как «особенная», а именно «задача отправления результата обратно станции назначения», то поток «висит» до тех пор, пока в очереди снова не появятся задачи.
* Задачи инициируются не только самими задачами, а в основном именно клиентами, которые присылают данные. После того, как получена команда «дать компоненту результирующего вектора», RTS оправляет всем клиентам запрос на необходимые элементы для того, чтобы удовлетворить потребностям станции. Клиенты начинают хаотично и беспорядочно по равномерному закону отсылать беспорядочные элементы. От каждого клиента сообщение обрабатывается и ставится соответствующая задача в очередь. Эти задачи диспетчер и будет обрабатывать.
* Получаем то, что основной поток прослушивает внешние сообщения от клиентов, а второй поток как бы «внутренние», то есть занимается выполнением задач, спровоцированных тем, что положил в очередь основной поток. Узким местом в такой системе является очередь задач, которую надо защищать с помощью мьютекса, что я и делаю. Больше данные между потоками нигде не пересекаются, а RTS получается устойчивой, безопасной и многопоточной.

Сокеты клиентов на RTS ставятся в массив fd\_set, который контролирует системный вызов select. В случае, если дескриптор сокета поменял свое состояние (в него было что-то записано), select срывается, а мы начинаем обработку того сообщения, которое пришло в соответствующий сокет (или несколько сокетов).

RTS обладает своей собственной базой данных (списком массивов и переменных) для сохранения промежуточных результатов между выполнениями задач, так как задачи выполняются независимо друг от друга (а в будущем планируется перейти на приоритеты, чтобы задачи еще и выполнялись не строго последовательно, а в зависимости от важности).

Диспетчер RTS GateWay записывает лог своей работы в файл. Основной поток выводит лог своей работы на экран. Работа второго потока завершается, как только «особенная» задача выполнена. Второй поток заново создается при инициировании станции назначения новой фазы операций.

По завершению работы клиенты матриц и векторов в системе RTS снимаются с регистрации, но перед этим им отправляется сообщение о том, что работа закончена. Работа закончена тогда, когда станция назначения пошлет сигнал, что работа закончена. Тогда и станция назначения и RTS и клиенты завершают свою работу.

Форматы данных и типы сообщений заранее известны всем общающимся и случающим процессам, тип команды приходит в виде массива байт в заголовке сообщения, а с ним и тело с дополнительными параметрами.

Проект находится под версионным контролем, обеспечивающий системой GIT.

# Проведенные испытания

Начальные значения распределений для клиентов, значение компонент матриц и векторов:

* MClient A:

#define MIN\_SLEEP\_TIME\_MLS 400

#define MAX\_SLEEP\_TIME\_MLS 500

Matrix:

10 0 0 0 0 0 0

0 10 0 0 0 0 0

0 0 10 0 0 0 0

0 0 0 10 0 0 0

0 0 0 0 10 0 0

0 0 0 0 0 10 0

0 0 0 0 0 0 10

* MClient C:

#define MIN\_SLEEP\_TIME\_MLS 400

#define MAX\_SLEEP\_TIME\_MLS 500

Matrix:

1 0 0 0 0 0 0

0 1 0 0 0 0 0

0 0 1 0 0 0 0

0 0 0 1 0 0 0

0 0 0 0 1 0 0

0 0 0 0 0 1 0

0 0 0 0 0 0 1

* VClient b:

#define MIN\_SLEEP\_TIME\_MLS 100

#define MAX\_SLEEP\_TIME\_MLS 300

vector [2 3 4 5 6 7 8]

* VClient d:

#define MIN\_SLEEP\_TIME\_MLS 100

#define MAX\_SLEEP\_TIME\_MLS 300

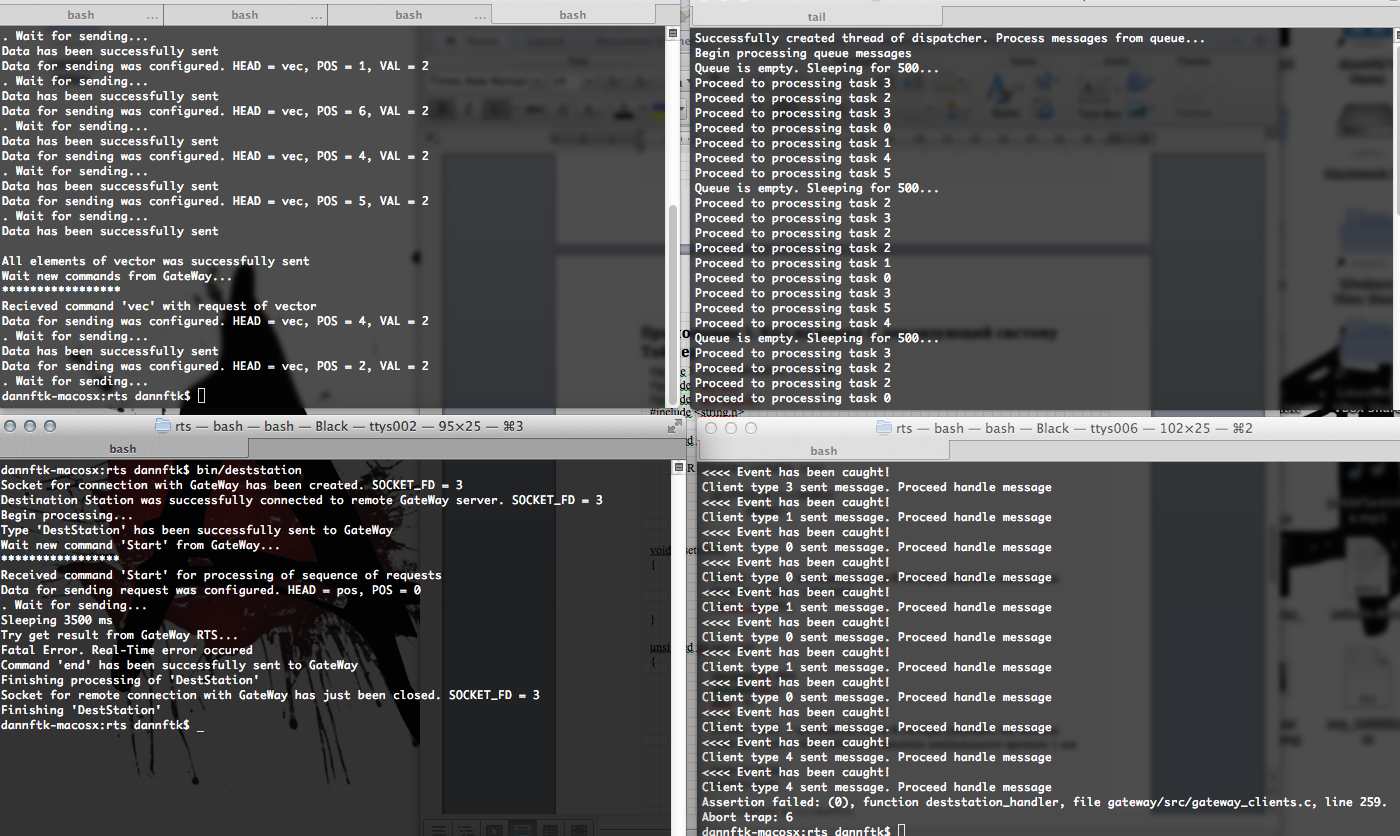
vector [2 2 2 2 2 2 2]

Если подсчитать грубо время, которое является точным заверенным провалом работы RTS, то оно считается как максимальное время задержки на отдачу элемента по каждой матрице и вектору, умноженные на размерность + системные издержки самой RTS, которые, на самом деле в рамках этой задачи, малы по сравнению с выше перечисленным

Тогда для установленных значений, имеем точное время провала, равное 3.5 секунды, то есть за это время система с вероятностью 1 не сможет выполнить свою задачу.

Время, за которое система точно сможет выполнить задачу: 6.5 секунд

Проверим на самой системе:

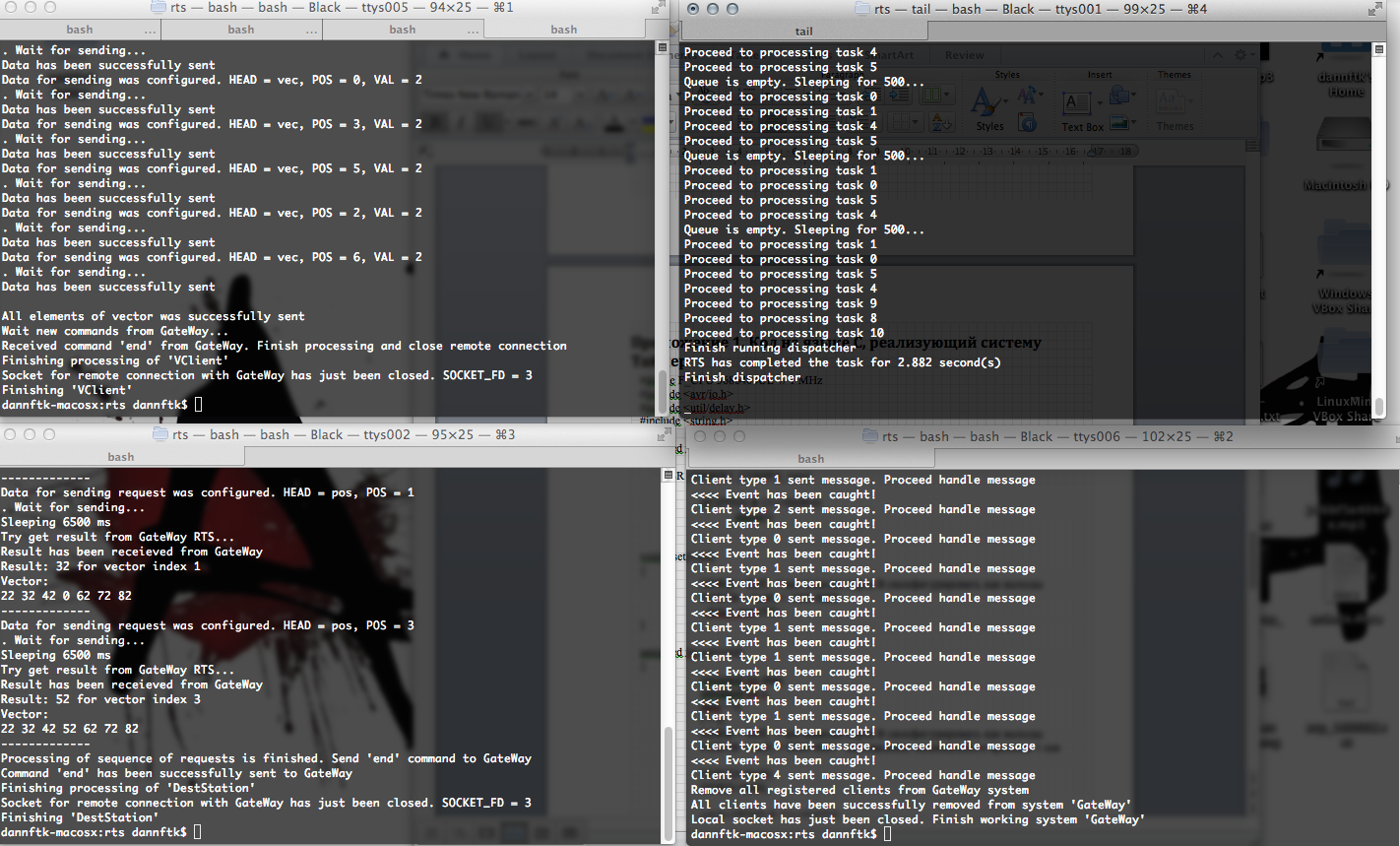


В левом верхнем углу представлена консоль, где отображены четыре вкладки на клиенты элементов матриц и векторов, в принципе они делают свою работу, и даже на результаты одного из них можно посмотреть. Он пишет, какие элементы отправил.

В правом верхнем углу отображен лог диспетчера: в нем пишется, какой тип задачи был вытащен из очереди, спал ли диспетчер на время отсутствия в очереди элементов.

В левом нижнем углу видим лог станции, которая упала из-за того, что реальное время не было выполнено RTS “GateWay”, лог которого мы видим в правом нижнем углу. Он тоже упал, так как получил команду “end” тогда, когда не должен был ее получать, так как в очереди еще находились задачи для выполнения операций.

Теперь проверим, как система работает с более мягким временем, заданным станцией, а именно 6.5 секунд:



Наблюдаем успешное выполнение всех операций, правильное завершение всех процессов, результирующий вектор, который был посчитан за минимальное время в условиях такой системы. На правильность работы программы можно проверить, посчитав вручную данный пример и убедившись в том, что алгоритм отработал четко.

В проекте практически 2000 строк кода:  
  
-------------------------------------------------------------------------------

Language files blank comment code

-------------------------------------------------------------------------------

C 13 373 43 1512

C/C++ Header 17 117 24 349

D 13 0 0 45

make 1 12 1 32

-------------------------------------------------------------------------------

SUM: 44 502 68 1938

-------------------------------------------------------------------------------

# Выводы

Рассмотренная выше система была полностью спроектирована мной, в проекте почти 2000 строк кода, он структурирован и в нем продумана архитектура, которая позволяет его расширять и добавлять изменения. Система является неплохим макетом, который приближен к реальным условиям, хоть и является выдуманной задачей. При необходимости локальные сокеты можно будет заменить на сетевые, если передача будет передаваться от хоста к хосту. Эта задача кажется мне более интересной, так как тут появляется зависимость от сети, что накладывает определенные проблемы на проектирование RTS.

Как выяснилось, RTS “GateWay” должна учитывать, по какому закону и по каким параметрам регулируются задержки в передачи данных, для того, чтобы спрогнозировать жесткое время выполнения своей операции. Для станции назначения есть только RTS и ее сервис, а что внутри под этим сервисом подразумевается, ее не интересует. Строгое соответствие тому, что задача должна быть вовремя выполнена – вот основная задача RTS.

Исходники и все необходимое здесь: <https://github.com/dannftk/rts>