электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

УДК 004.94

Обнаружение блокирующих зависимостей в многопоточной сетевой службе

Агеев Д. Б., *студент*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Программное обеспечение ЭВМ и информационные технологии»

Научный руководитель: Градов В.М., д.т.н, профессор
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана
gradov@bmstu.ru

Введение

Блокирующей зависимостью будем называть проявление ошибки в сетевой службе при обработке сетевых соединений, из-за которой действия одного из клиентов приводят к приостановке обработки сетевых соединений других клиентов на неопределенный промежуток времени без гарантии возврата к нормальной обработке.

В большинстве случаев для выявления такой некорректной работы используют нагрузочное тестирование. С точки зрения рассматриваемой проблемы недостатком такого метода является отсутствие гарантии проявления блокирующей зависимости.

Зависимости можно также выявить, определяя корректность работы с конкретными объектами ядра операционной системы [1]. В таком случае задача требует рассмотрения всех правил работы с объектами, но это не гарантирует покрытия всех возможных ситуация, так как ошибка может быть не связана с рассматриваемыми объектам.

В данной работе предлагается использовать перехват и подмену системных вызовов для обеспечения наступления событий, проявляющих блокирующую зависимость. Схожий подход используется в библиотеке dmalloc [2], которая перехватывает функции стандартной библиотеки Си для выявления ошибок при работе с памятью.

В отличие от простого нагрузочного тестирования, предложенный метод заключается в создании серии клиентов, часть из которых вызывает блокировку сетевых соединений.

Обнаружение возможных блокирующих зависимостей с помощью пробных клиентов

Рассмотрим сетевую службу, в которой осуществляется мультиплексирование для чтения данных, но отсутствует для записи, при этом все сетевые соединения обрабатываются в одном потоке выполнения. В данном случае между всеми сетевыми соединениями существует блокирующая зависимость. Блокирующие зависимости могут существовать и между сетевыми соединениями, обработка которых производится в разных потоках выполнения.

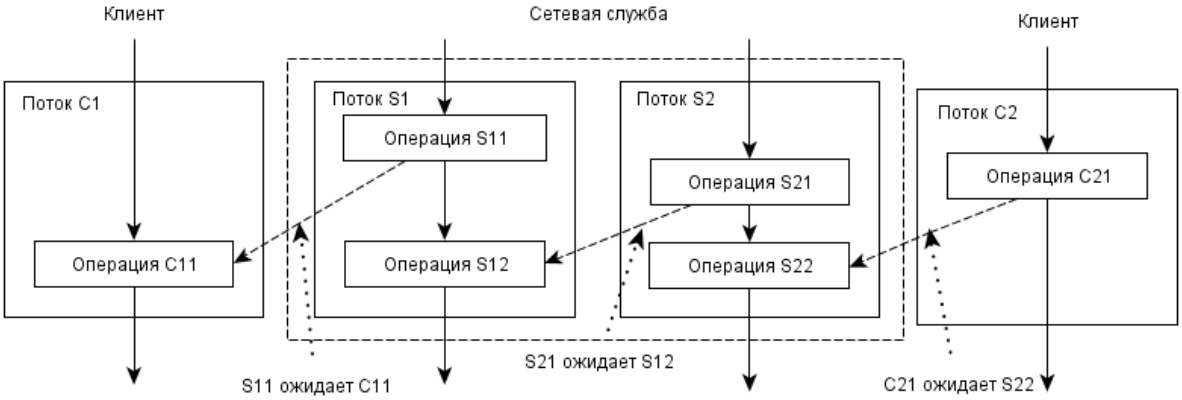

Рис. 1 Возникновение блокирующей зависимости в многопоточных сетевых службах

На рис.1 представлены четыре потока выполнения: *C1*, *C2*, *S1* и *S2*. Поток *S1* и поток *S2* являются потоками выполнения сетевой службы. Потоки *C1* и *C2* – потоки выполнения клиентов, которые взаимодействуют с данной сетевой службы. Поток *S2* на своей первой операции ожидает выполнения потоком *S1* своей второй операции. При достижении потоком *S1* первой операции, поток *S1* переходит в состояние ожидания до тех пор, пока поток *C1* не выполнит свою первую операцию. Время возврата из состояния ожидания потоком *S1* потока *C1* является негарантированным и неопределенным, поэтому возврат из состояния ожидания потоком *S2* является также негарантированным и неопределенным, а как следствие обработка всех сетевых соединений потоком *S2* будет приостановлено. Таким образом, блокирующая зависимость может существовать не только между сетевыми соединениями одного потока, но и между сетевыми соединениями разных потоков выполнения сетевой службы.

Предлагаемый метод выявления блокирующих зависимостей состоит из этапа расчета критического времени, расчета времени работы серии клиентов, в которой находятся пробные клиенты, а также сравнения и принятия решения о том, существует ли зависимость, либо такой зависимости нет.

На рис.2 представлена функциональная декомпозиция разработанного метода, выявляющего блокирующие зависимости.

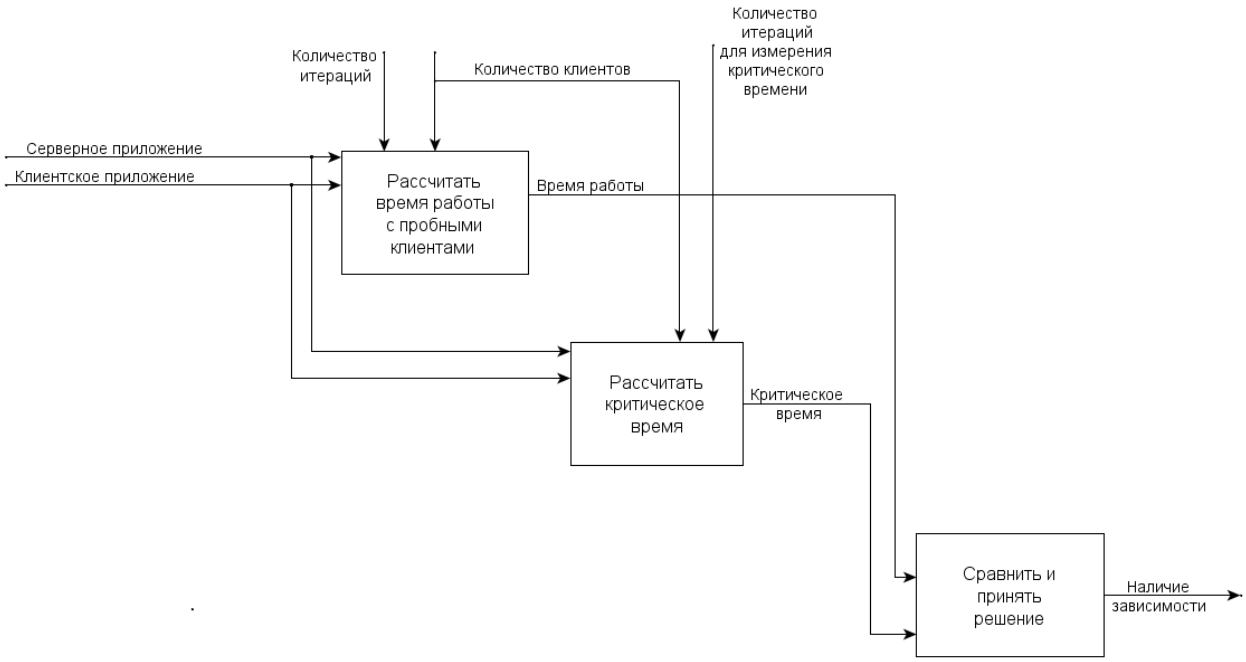

Рис.2 Этапы выявления блокирующих зависимостей

Пробными клиентами будут называться клиенты, активность которых приводит к блокировке потоков выполнения сетевых служб.

Критическим временем будет называться время, по истечению которого можно с большой вероятностью сказать, что для всех незавершенных сетевых соединений обработка запросов от соответствующих клиентов приостановлена.

Производится расчет среднего времени выполнения серии клиентов и стандартное отклонение (предполагается, что нагрузка центрального процессора постоянна). Среднее время работы серии клиентов рассчитывается по формуле $\mu = \sum_{i=1}^{n} t_i$, где t_i – время работы серии клиентов на i-ой итерации, n – количество итераций, μ – среднее время работы серии клиентов. Возможно рассчитать стандартное отклонение по формуле $s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(t_i - \mu)^2}$. Как известно, для нормального распределения значение случайной величины с вероятностью *99.7* находится в интервале значений *(μ - 3s; μ + 3s)*, поэтому значение критического времени можно принять равным μ + *3s*.

Для выявления наличия зависимостей будут использоваться три типа клиента: пробный клиент по записи, пробный клиент по чтению и обычный клиент. Пробные клиенты блокируют потоки выполнения сетевой службы, в то время как обычные клиенты

необходимы для выявления негативных последствий, которые возникают в результате действий пробных клиентов.

Обозначим общее количество клиентов n, пробных клиентов по записи – q, а пробных клиентов по чтению – p. Количество обычных клиентов равно $d=n-q-p$. В таком случае $n > q + p$, так как обязательно наличие каждого типа клиентов. Предположим, что только один клиент из n способен перевести сетевую службу в такое состояние, которое позволит выявить наличие зависимостей.

Вероятность того, что наугад взятый из n клиентов клиент окажется требуемым клиентом равно $\frac{1}{n}$. Предположим, что вероятность того, что требуемый клиент должен быть пробным клиентом по чтению равно r. Так как эти события независимы, то вероятность их одновременного наступления равна произведению их вероятностей, то есть равна $\frac{1}{n}$. Если предоставляется выбрать m клиентов из n, то вероятность того, что из m выбранных клиентов один будет необходимого типа равна $r\frac{m}{n}$, в том случае, если пробный клиент является пробным клиентом по чтению и $(1-r)\frac{m}{n}$ в противном случае. Таким образом, вероятность того, что требуемый клиент должен быть пробным клиентом по записи и то, что из p выбранных пробных клиентов по чтению один окажется требуемым клиентом, равна $r\frac{p}{n}$.

Аналогично для пробного клиента по записи – $(1-r)\frac{q}{n}$. Вероятность наступления одного из двух событий равна сумме вероятностей этих событий без вероятности их совместного наступления. Два рассмотренных события не могут наступить одновременно, поэтому вероятность наступления одного из двух событий равна $\frac{rp+(1-r)q+q}{n}$. Таким образом, вероятность того, что, взяв p пробных клиентов по чтению, q пробных клиентов по записи из n клиентов, будет найден клиент, действия которого приводят к проявлению существующей зависимости, при том, что вероятность того, что данный пробный клиент является пробным клиентом по чтению, равно $\frac{rp+(1-r)q+q}{n}$.

Значения p и q должны быть такими, что $\frac{rp+(1-r)q+q}{n} \to max$, а так, как n известно, то должно выполняться следующее условие $rp + (1-r)q + q \to max$. Представим это в виде системы уравнений.

$$\begin{cases} f = rp + (1-r)q \to max \\ p + q = n - d \\ p, q \geq 0 \\ p, q \in N \end{cases}$$

При этом существуют дополнительные ограничения $r \in [0;1]$, $n \in N$ и $d=n-p-q$. Можно свести эту систему к задаче целочисленного программирования, заменив $p+q = n-d$ на $p+q \le n-d$ и $p+q \ge n-d$. Решим задачу целочисленного программирования. Приведем задачу целочисленного программирования к стандартной форме. Для нахождения первоначального базисного допустимого решения введем переменную x_3.

$$\begin{cases} f = rp + (1-r)q \to max \\ p + q + x_1 = n - d \\ p + q - x_2 + x_3 = n - d \\ p, q \ \ge 0 \\ p, q \ \in N \end{cases}$$

Целевая функция для вспомогательной задачи $w = -x_3 = -(n-d) + p + q - x_2$.

Для решения данной задачи использую симплекс-таблицу. Включаем в базис p, исключаем из базиса x_1. Переходим к решению исходной задачи, представленной в приведенной ниже таблице. Так как $r \in [0;1]$, то $-r < 0$, а $1-2r$ в зависимости от значения r может принимать как положительное, так и неположительное значение. Если $r < 0.5$, то $p' = n-d$, а $q' = 0$, иначе необходимо включить в базис q и исключить p. В таком случае $q'' = n-d$, а $p'' = 0$. Так как 0.5 является серединой интервала $[0;1]$, то, не учитывая середину интервала, вероятности того, что $r \ge 0.5$ и $r < 0.5$ равны 0.5. Вероятностное среднее $p = \left[\frac{1}{2}(p' + p'')\right] = \left[\frac{n-d}{2}\right]$, а $q = \left[\frac{1}{2}(q' + q'')\right] = \left[\frac{n-d}{2}\right]$, где квадратными скобками показана операция взятия целой части. Таким образом, оптимальное значение количества пробных клиентов по чтению и пробных клиентов по записи равно $\left[\frac{n-d}{2}\right]$.

Вероятность того, что один из пробных клиентов совпадет с требуемым клиентом, равна $\frac{rp+(1-r)q}{n}$. Если взять оптимальное количество пробных клиентов по чтению и по записи, то эта вероятность будет равна $\frac{r\left[\frac{n-d}{2}\right]+(1-r)\left[\frac{n-d}{2}\right]}{n} = \frac{\left[\frac{n-d}{2}\right]}{n}$.

Рассмотрим случай, когда $n \to \infty$, тогда можно пренебречь операцией взятия целой части. В таком случае $\lim_{n\to\infty} \frac{\left[\frac{n-d}{2}\right]}{n} = \frac{1}{2}$. Таким образом, вероятность правильного выбора пробных клиентов больше, либо равна 0.5, а вероятность ошибки также стремится к 0.5.

Пусть m – количество экспериментов для выявления зависимостей. В таком случае вероятность не нахождения зависимостей, в случае их присутствия, одновременно во всех m экспериментах равна $\left(\lim_{n\to\infty} \frac{\frac{n-d}{2}}{n}\right)^m$.

Вероятность того, что хотя бы в одном из экспериментов требуемый клиент буде выявлен корректно, равна $1-\left(\lim_{n\to\infty}\frac{\frac{n-d}{2}}{n}\right)^m$. Если устремить значение $m\to\infty$, то данная вероятность равна $\lim_{n\to\infty}\left(1-\left(\lim_{n\to\infty}\frac{\frac{n-d}{2}}{n}\right)^m\right)=1$. Это означает, что вероятность не выявить зависимость стремится к нулю. Таким образом, вероятность не выявить ошибку, когда она присутствует, то есть вероятность ошибки второго рода, при увеличении числа экспериментов уменьшается.

Программная реализация

Библиотека glibc – библиотека языка Си для Linux систем. В ней содержаться все основные функции, необходимы для написания программного кода, в том числе важные для данной работы функции межмашинного взаимодействия. В рамках данной работы осуществляется перехват именно этой библиотеки.

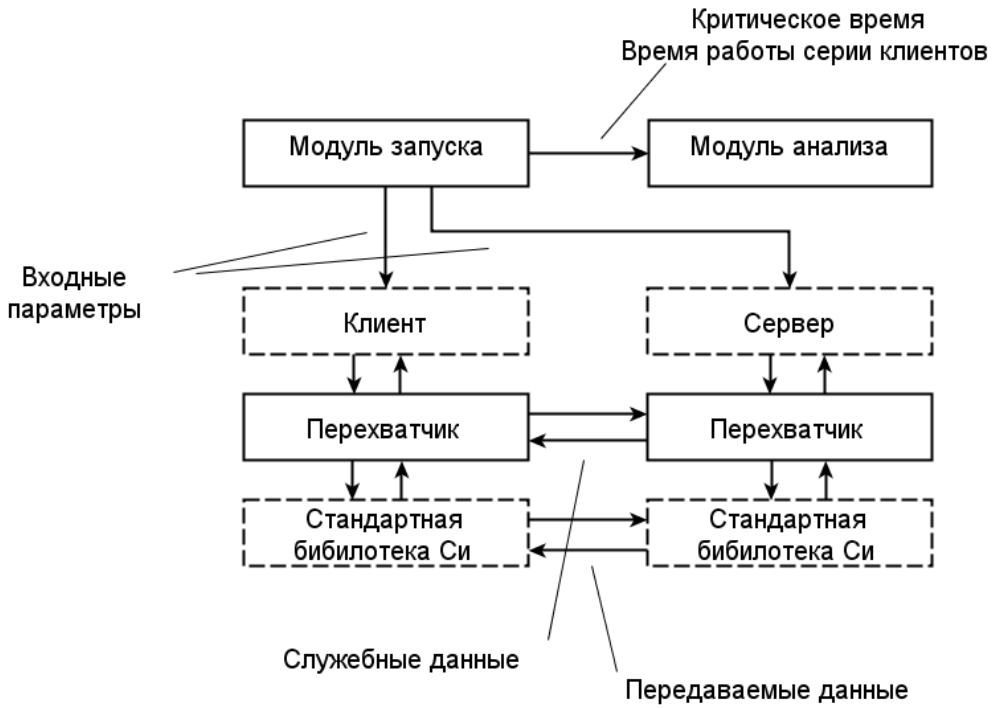

Рис.3 Способ перехвата системных вызовов

Программная реализация метода основывается на перехвате системных вызовов, отвечающих за межсетевое взаимодействие. Для перехвата функций библиотеки glibc используется переменная окружения LD_PRELOAD, в которой указывается адрес загружаемой библиотеки [3]. Способ перехвата представлен на рис.3.

Перехватываются функции взаимодействия между клиентами и сетевой службой. Рассмотрим вначале перехват функций на стороне клиента. Прежде всего необходимо получить указатель на оригинальную функцию. Затем по данному указателю с параметрами, которые получила перехватывающая функция, вызвать соответствующую оригинальную функцию, результат работы функции необходимо запомнить, так как именно этот результат будет возращен перехватывающей функцией. Отрицательное, либо равное нулю, значение возвращаемой функции свидетельствует о том, что произошло непредвиденное обстоятельство, поэтому целесообразно не производить дальнейшие действия и вернуть полученное неположительное значение из перехватывающей функции.

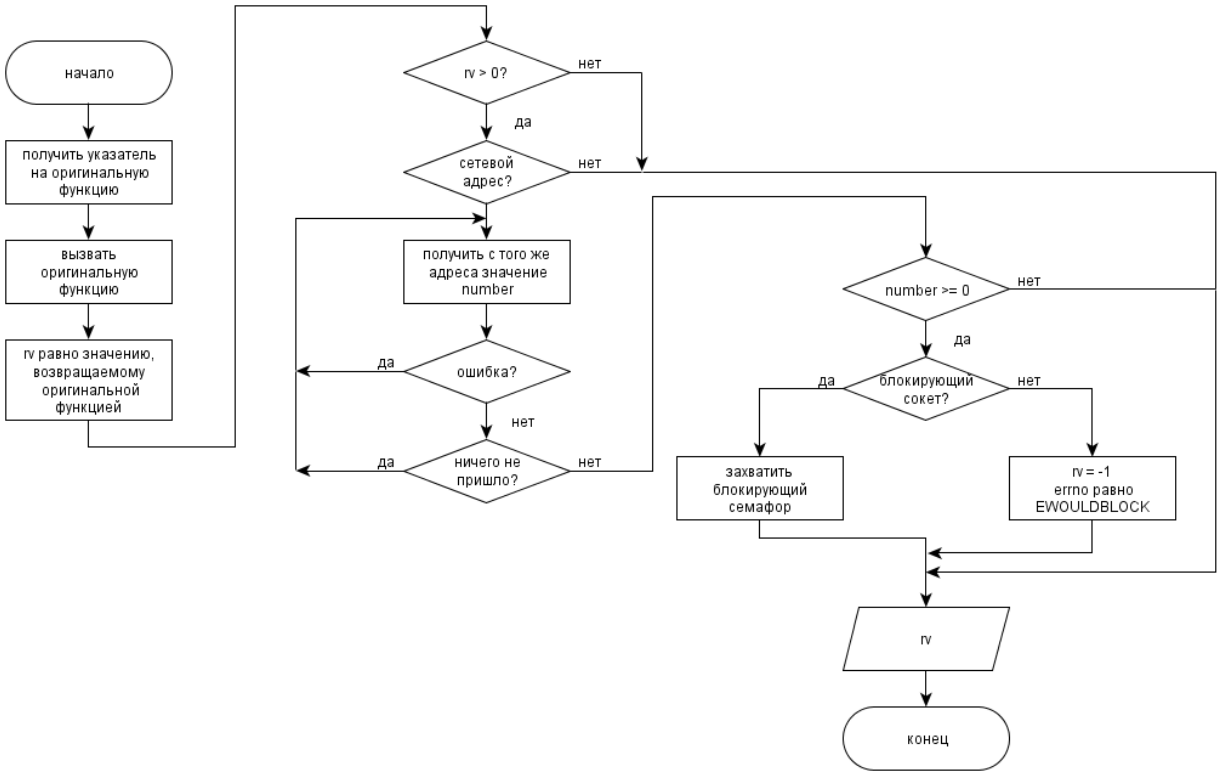

Рис.4 Перехват системных вызовов на стороне сетевой службы

Затем проверяется, является ли клиент пробным или нет. Если перехватываемая функция является функцией *send*, а клиент является пробным по записи, или если перехватываемая функция является функцией *recv*, а клиент является пробным по чтению, то необходимо отправить положительное число. Если клиент не является пробным или является пробным, но по чтению во время вызова функции *send* и по записи при вызове функции *recv*, то серверу передается отрицательное значение. Таким образом, сервер сможет определить взаимодействует ли он с пробным или с обычным клиентом. В

завершении необходимо вернуть значение, полученное от перехватываемой функции. Алгоритм перехвата системных вызовов на стороне сетевой службы представлен на рис.4.

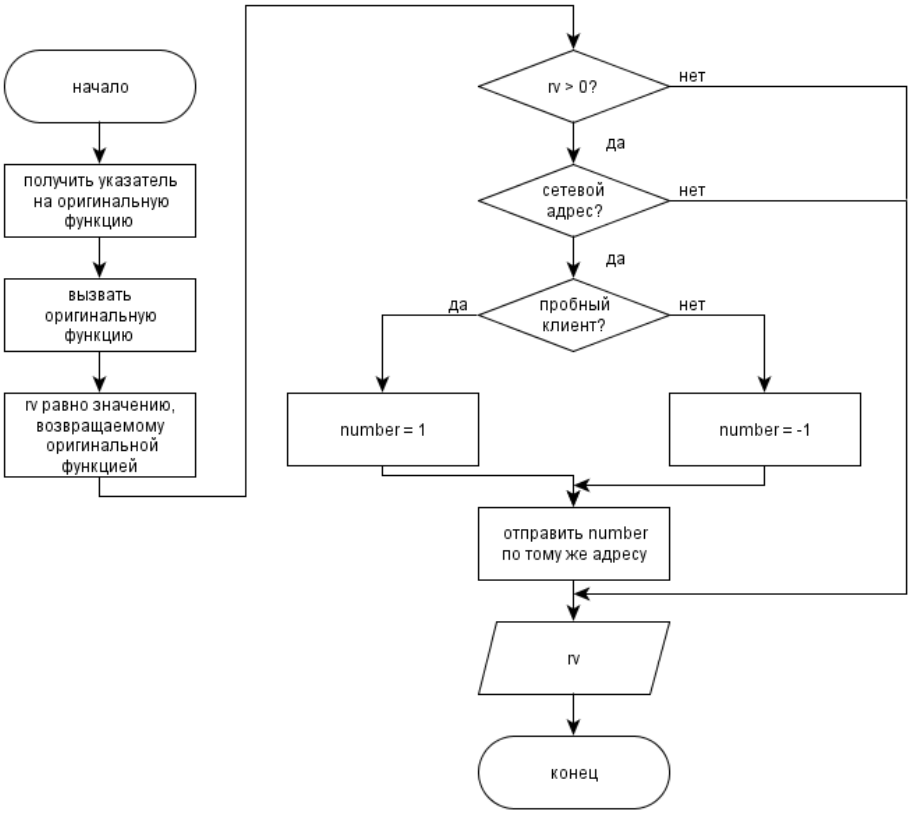

Рис.5 Перехват системных вызовов на стороне клиентов

Как и при перехвате функций *send* и *recv*, на серверной стороне осуществляется определение указателя на оригинальную функцию, вызов функции по данному указателю и получение значения, которое было возращено этой функцией. Затем анализируется возвращенное значение, если оно неположительное, то на этом работа перехватчика завершается, и он возвращает это значение. Если значение положительное, то необходимо проверить сокет, с которым работает перехватывающая функция, если это локальный сокет, то нужно вернуть полученное от оригинальной функции значение, иначе необходимо ожидать получения дополнительных данных по данному сокету. В качестве дополнительных данных выступает переданное с клиентской стороны значение либо положительное, которое означает, что клиент является пробным, либо отрицательное в противном случае. Соответственно, сервер может получить положительное число только при вызове функции *recv*, когда пробный клиент является пробным по записи, и при вызове функции *send*, когда клиент является пробным по чтению. Если пробный клиент является пробным по чтению при вызове *recv* и по записи при вызове *send*, то сервер

должен получить отрицательное значение. Так как серверное приложение для работы с клиентами может использовать неблокирующие сокеты, то для получения дополнительное информации от клиента необходимо использовать вызов *recv* в цикле до тех пор, пока функция *recv* не вернет ненулевое значение. Это позволит дождаться серверу необходимой информации. После получения дополнительной информации необходимо проверить полученное значение, если оно отрицательное, то перехватывающая функция возвращает полученное от перехватываемой функции значение. Если значение положительное, то необходимо заблокировать, либо вернуть EWOULDBLOCK в зависимости от типа семафора, предварительно, как и при работе с клиентом, проверив, является ли сокет сетевым. Алгоритм перехвата системных вызовов на стороне сетевой службы представлен на рис.5.

Проведение эксперимента

Чтобы выявить наличие зависимости в обработке сетевых соединений, программный комплекс многократно запускает сетевую службу и серию клиентов. Предположим, что при измерении работы серии клиентов один из результатов значительно отклонился от среднего значения.

Если данное значение отклонилось от среднего значения в меньшую сторону, то критическое время будет меньше, чем в случае без отклонившегося значения. Это приведет к уменьшению общего времени работы программного комплекса, но при этом увеличится вероятность принятия ошибки первого рода, то есть возрастает вероятность выявления наличия зависимости в сетевой службе, в которых на самом деле нет зависимостей. Если данное значение отклонилось от среднего значения в большую сторону, то работа серии клиентов будет выполняться дольше, чем могло бы при отсутствии отклонившегося значения, а вероятность ошибки первого рода уменьшится. Так как вероятность ошибки перового рода была незначительной, то факт уменьшения вероятности ошибки перового рода не является критичным для принятия решения о наличии блокировки.

Увеличение критического времени значительно влияет на продолжительность выявления наличия зависимостей, так как программный комплекс после запуска серии клиентов ожидает время, равное критическому времени, для того, чтобы дать возможность завершиться клиентам, на работу которых не повлияла попытка вывода сервера из состояния корректной работы. Для того чтобы время работы программного комплекса не превышало необходимого для выявления наличия зависимостей в обработке

сетевых соединений времени, необходимо увеличить количество запусков серии клиентов при расчете критического времени, чтобы снизить влияние отклонившегося значения на значение критического времени. Уменьшение критического времени может внести больший вклад в общее время работы программного комплекса, чем увеличение итераций расчета этого критического времени.

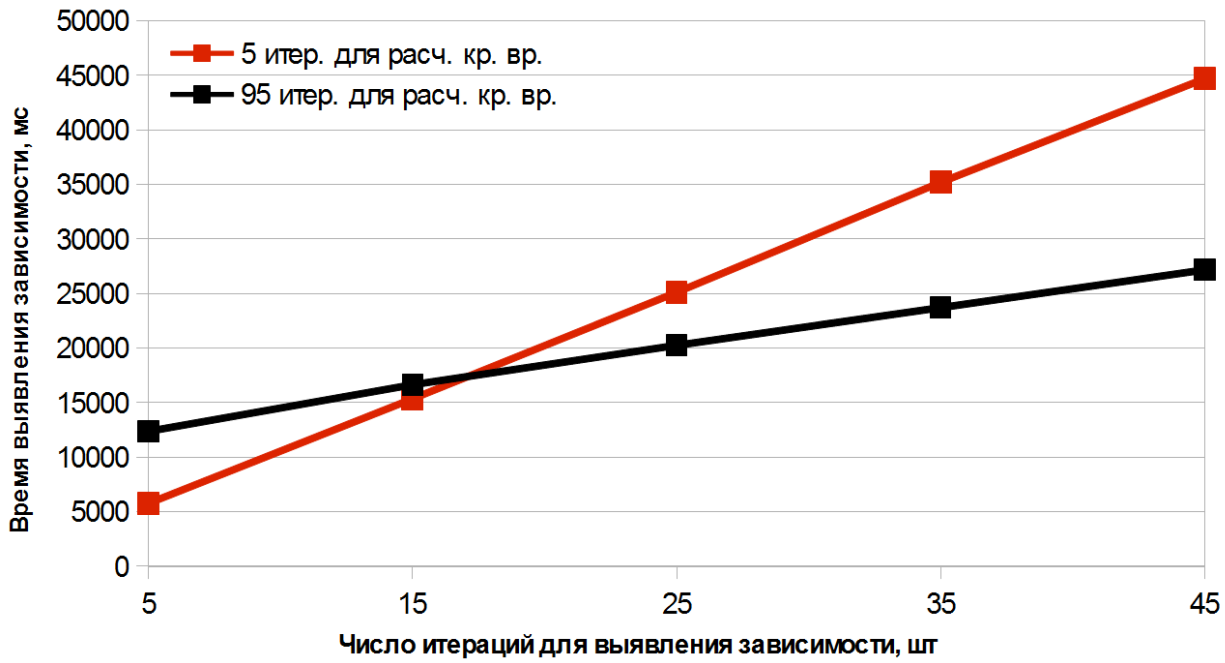

Рис. 6 Время выявления зависимости при разном числе итераций вычисления

Проведён эксперимент, который заключается в определении времени работы программного комплекса, когда расчете критического времени появляется значение, значительно отклонившееся от среднего значения времени работы программы. В эксперименте искусственно увеличивается работа в одной из итераций на этапе расчёта критического времени. Эксперимент проводится при 5 итерациях и при 95 итерациях расчета критического времени (рис.6). Из результатов видно, что программный комплекс при наличии единичного всплеска при расчете критического времени дает хуже результат при 95 итерациях, чем при 5 итерациях при небольшом количестве итераций для непосредственно выявления зависимости. При увеличении итераций для непосредственно выявления зависимостей время работы программы с 5 итерациями для расчета критического времени становится хуже, чем программы с 95 итерациями. Таким образом, при большом значении числа итераций для непосредственно выявления зависимостей следует увеличивать число итераций, необходимых для расчета критического времени,

чтобы сократить влияние возможных отклонений на значение критического времени, что позволит уменьшить общее время работы метода.

Список литературы

1. Watson G. Debug Malloc Library Version 5.5.2 Available at: https://www.student.cs.uwaterloo.ca/~cs241/cLibs/dmalloc552/dmalloc-5.5.2-solaris/docs/dmalloc.pdf, accessed 01.10.2014.
2. Joshi P., Park C., Sen K., Naik M. A Randomized Dynamic Program Analysis Technique for Detecting Real Deadlocks // ACM SIGPLAN Conference on Programming Language Design and Implementation. PLDI, 2009. P. 110-120.
3. Roy D. Let's Hook a Library Function. Available at: http://www.linuxforu.com/2011/08/lets-hook-a-library-function/, accessed 01.10.2014.