Программные технологии проектирования высоконагруженных систем обработки данных

Смекалин С. А.

Смекалин Сергей Александрович / Smekalin Sergey Aleksandrovich - магистрант,
кафедра телекоммуникационных систем,
факультет микроприборов и технической кибернетики,
Московский институт электронной техники, г. Зеленоград

Аннотация: *с каждым днем потребности пользователей в Интернете растут. Чтобы удовлетворить все их запросы необходимо качественно проектировать центры обработки данных. Цель работы - познакомить начинающих программистов с разработкой, связанной с повышением отказоустойчивости высоконагруженных систем обработки данных. В статье рассматриваются подходы к обработке входящих запросов на стороне высоконагруженной системы. Описывается, как устроена обработка запросов на процессах, потоках и асинхронная событийная модель, выявляются преимущества и недостатки данных подходов.*

Ключевые слова: *высокая нагрузка, системы обработки данных, асинхронная событийная модель.*

В наше время практически каждый человек пользуется Интернетом. Пользователи Интернета каждый день ищут, загружают, скачивают необходимую информацию. Для обработки запросов пользователей используются центры обработки данных, которым необходимо выдерживать высокую нагрузку и обслуживать всех клиентов. От уровня устойчивости таких систем к отказам зависит прибыль компаний. Соответственно системы обработки данных должны выдерживать высокую нагрузку и оставаться при этом быстродействующими.

Для обработки входящих запросов часто используется архитектура системы, построенная на процессах. Для каждого соединения создается процесс и таким образом обеспечивается многопоточность и система может обрабатывать большое количество соединений. К примеру, так реализован известный веб-сервер Apache с модулем мультипроцессовой обработки Prefork. Prefork заранее создает определенное количество процессов, которые не требуют времени на отдельный вызов при поступлении запроса от клиента и не нуждаются в выполнении маршалинга [1]. Схема работы приведена на рисунке 1.

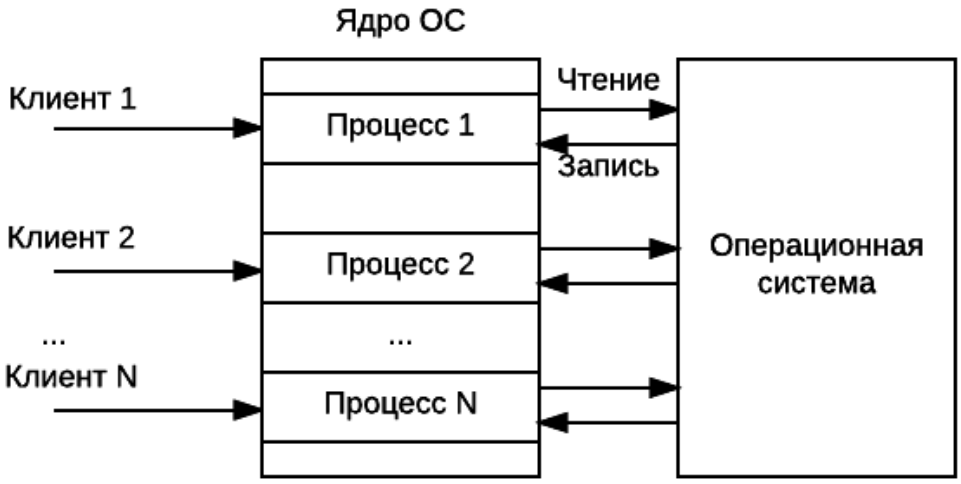

Рис. 1. Обработка запросов процессами ядра ОС

Одним из важнейших преимуществ такого подхода является большая безопасность, так как каждый процесс изолирован от других и не может повредить или прочитать память другого процесса. Но есть недостатки такого подхода - создание процесса для каждого соединения требует выделения большого количества памяти сервера. Чем больше количество запросов, тем больше памяти требуется. Также при обработке запроса процесс блокируется и пока клиент или сервер не завершит соединение, процесс будет потреблять ресурсы системы. Это наиболее заметно, когда при запросах клиент включает в HTTP запрос режим поддержки соединения keep-alive, но при этом обращается к серверу пару раз в минуту, из-за этого процесс простаивает большую часть времени.

Следующая модель обработки данных основывается на потоках. Обработка запросов использует потоки вместо процессов, тем самым значительно улучшая обработку большого количества соединений. Рассмотрим на примере модуля Apache Worker. Worker запускает несколько процессов, каждый из которых создает несколько, так называемых дочерних потоков. Дочерние потоки, как и процессы, ожидают входящие запросы. Схема работы представлена на рисунке 2.

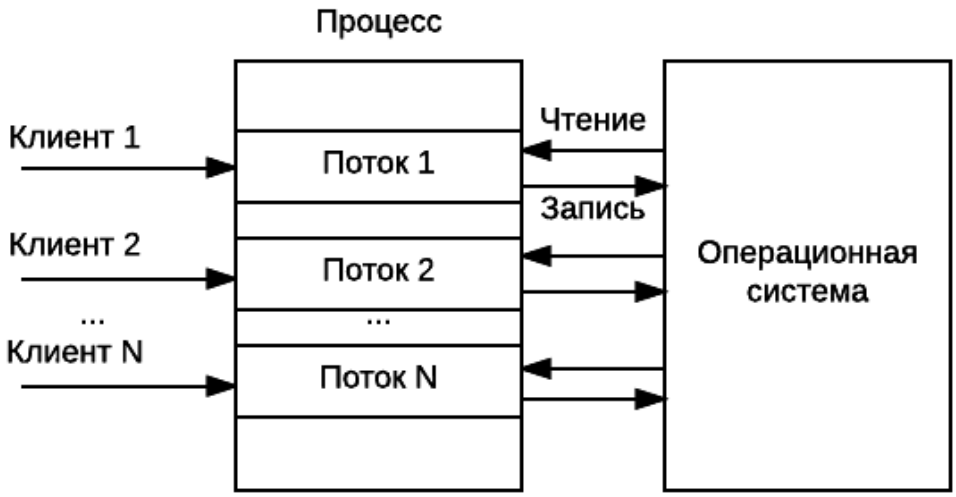

Рис. 2. Модель обработки клиентских запросов с помощью потоков

Такой подход является менее ресурсоемким в плане потребления оперативной памяти, в отличие от первого подхода. Также улучшается обработка большого количества одновременных запросов. Недостаток потоковой обработки запросов это относительная нестабильность, так как проблемы в одном процессе могут затронуть и другие соединения. Это связано с тем, что потоки одного процесса имеют общее адресное пространство и память [2].

Следующая модель обработки данных является самой современной и быстрой из существующих, но в то же время самой сложной в реализации и отладке. Разберем модель асинхронной обработки ввода/вывода на примере объекта ядра Linux, называемого Epoll. Epoll является современным подходом реализации данной модели в отличие от select и poll, которые не имеет смысла рассматривать в данной статье. На сервере создается определенное количество потоков, которые будут заниматься обработкой запросов от клиентов, также создается объект Epoll, который представляет собой структуру данных, содержащая дескрипторы. При установлении соединения с пользователем на сервере создается клиентский дескриптор, от которого ожидается появление данных. Каждый дескриптор добавляется к наблюдению в объект Epoll и вызывается функция epoll_wait(), которая возвращает массив дескрипторов, в которых поменялось состояние, то есть там содержатся дескрипторы, которые готовы к вводу/выводу. При появлении событий от дескриптора ядро ОС передает его потоку, который непременно принимает запрос и обрабатывает его. После обработки одного запроса поток отправляет ответ и возвращается в пул потоков, готовый для обработки следующих запросов [3]. Таким образом, если клиент редко посылает запросы, но при этом поддерживает соединение keep-alive, он будет занимать поток не на весь сеанс соединения, а только на один запрос.

Литература

1. [Электронный ресурс]. Документация по Apache prefork. Режим доступа: http://httpd.apache.org/docs/current/mod/prefork.html/ (дата обращения: 18.12.2016).
2. [Электронный ресурс]. Документация по Apache worker. Режим доступа: https://httpd.apache.org/docs/2.4/mod/worker.html/ (дата обращения: 18.12.2016).
3. [Электронный ресурс]. Документация Linux по epoll. Режим доступа: http://man7.org/linux/man-pages/man7/epoll.7.html/ (дата обращения: 18.12.2016).