РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ СИСТЕМЫ ВИДЕОНАБЛЮДЕНИЯ НА ОСНОВЕ ОДНОПЛАТНОГО КОМПЬЮТЕРА

Горбунов Иван Сергеевич
студент, Гомельский государственный технический университет им. П.О. Сухого,
Беларусь, г. Гомель

Курочка Константин Сергеевич
научный руководитель, канд. техн. наук, доц., Гомельский государственный технический университет им. П.О. Сухого,
Беларусь, г. Гомель

Введение. Под определенную модель одноплатного компьютера разрабатывается операционная система, которая записывается на внешнем накопителе. Специалисты однозначно определяют операционную систему Linux, как наиболее стабильную при долгосрочной непрерывной работе и устойчивую к внешним воздействиям – вирусы, пиратские взломы и т.п. Но, есть возможности поставить даже Windows 10, хоть она является довольно требовательной. Одноплатный компьютер открывает перед пользователем неограниченный доступ к программной части, например, вы можете установить новые программы или драйвера, удалить старые, также без проблем написать свои собственные. В данной статье будет рассмотрен вариант разработки программного обеспечения для данной системы видеонаблюдения.

Архитектура и сетевое взаимодействие. В основе работы приложения лежит так называемая **модель взаимодействия клиент-сервер,** которая позволяет разделять функционал и вычислительную нагрузку между клиентскими приложениями и серверными приложениями. **Клиент и сервер** взаимодействую друг с другом в сети Интернет или в любой другой компьютерной сети при помощи различных сетевых протоколов, например, *IP* протокол, *HTTP* протокол, *FTP* и другие [1]. Протоколов на самом деле очень много и каждый протокол позволяет оказывать ту или иную услугу. Например, при помощи *HTTP* протокола браузер отправляет специальное *HTTP* сообщение, в котором указано какую информацию и в каком виде он хочет

получить от сервера, сервер, получив такое сообщение, отсылает браузеру в ответ похожее по структуре сообщение (или несколько сообщений), в котором содержится нужная информация, обычно это *HTML* документ. Преимуществом модели взаимодействия клиент-сервер является то, что программный код клиентского приложения и серверного разделен. Если мы говорим про локальные компьютерные сети, то к преимуществам архитектуры клиент-сервер можно отнести пониженные требования к машинам клиентов, так как большая часть вычислительных операций будет производиться на сервере, а также архитектура клиент-сервер довольно гибкая и позволяет администратору сделать локальную сеть более защищенной.

Безопасность. В качестве потока обмена информацией предлагается *SslStream*. *SslStream* – это поток, используемый для связи клиент-сервер, который использует протокол безопасности *Secure Socket Layer* (*SSL*) для аутентификации сервера и, при необходимости, клиента. Протоколы *SSL* помогают обеспечить конфиденциальность и проверку целостности сообщений, передаваемых с использованием *SslStream*. Соединение *SSL*, такое как предоставляемое *SslStream*, должно использоваться при передаче конфиденциальной информации между клиентом и сервером. Использование *SslStream* помогает предотвратить чтение и вмешательство кого-либо в информацию, когда она находится в сети. Экземпляр *SslStream* передает данные, используя поток, который вы предоставляете при создании *SslStream*. Когда вы предоставляете этот базовый поток, у вас есть возможность указать, закрывает ли *SslStream* также базовый поток. Как правило, класс *SslStream* используется с классами *TcpClient* и *TcpListener*. Метод *GetStream*() предоставляет *NetworkStream*, подходящий для использования с классом *SslStream*. После создания *SslStream* сервер и, необязательно, клиент должны пройти аутентификацию. Сервер должен предоставить сертификат *X*509, который устанавливает подтверждение его личности, и может запросить, чтобы клиент также сделал это. Аутентификация должна выполняться перед передачей информации с использованием *SslStream*. Клиенты инициируют

аутентификацию, используя синхронные методы *AuthenticateAsClient*, которые блокируют до завершения аутентификации, или асинхронные методы *BeginAuthenticateAsClient*, которые не блокируют ожидание завершения аутентификации. Серверы инициируют аутентификацию с использованием синхронных методов *AuthenticateAsServer*() или асинхронных методов *BeginAuthenticateAsServer*(). На рисунке 2.3 отображена информация о сертификате, предоставляемым сервером. И клиент, и сервер должны инициировать аутентификацию. Аутентификация обрабатывается поставщиком канала поставщика поддержки безопасности (*SSPI*). Клиенту предоставляется возможность контролировать проверку сертификата сервера, указав делегата *RemoteCertificateValidationCallback* при создании *SslStream*. Сервер также может управлять проверкой, предоставляя делегат *CertificateValidationCallback*. Метод, на который ссылается делегат, включает сертификат удаленной стороны и любые ошибки *SSPI*, обнаруженные при проверке сертификата. Обратите внимание, что, если сервер указывает делегата, метод делегата вызывается независимо от того, запрашивал ли сервер аутентификацию клиента. Если сервер не запрашивал аутентификацию клиента, метод делегата сервера получает пустой сертификат и пустой массив ошибок сертификата. Если сервер требует аутентификации клиента, клиент должен указать один или несколько сертификатов для аутентификации. Если у клиента более одного сертификата, он может предоставить делегата *LocalCertificateSelectionCallback*, чтобы выбрать правильный сертификат для сервера. Сертификаты клиента должны находиться в хранилище сертификатов текущего пользователя «Мой». Проверка подлинности клиента с помощью сертификатов не поддерживается для протокола *SSL2*. Если аутентификация не удалась, вы получаете *AuthenticationException*, и *SslStream* больше не может использоваться. Вы должны закрыть этот объект и удалить все ссылки на него, чтобы он мог быть собран сборщиком мусора. Когда процесс аутентификации, также известный как рукопожатие *SSL*, завершается успешно, устанавливается идентификация сервера (и, возможно, клиента), и *SslStream* может

использоваться клиентом и сервером для обмена сообщениями. Процесс аутентификации между сервером и клиентов представлен на рисунке 2.4. Перед отправкой или получением информации клиент и сервер должны проверить службы безопасности и уровни, предоставляемые *SslStream*, чтобы определить, соответствуют ли выбранный протокол, алгоритмы и сильные стороны их требованиям целостности и конфиденциальности. Если текущих настроек недостаточно, поток должен быть закрыт. Вы можете проверить службы безопасности, предоставляемые *SslStream*, используя свойства *IsEncrypted* и *IsSigned*. После успешной аутентификации вы можете отправлять данные, используя синхронные методы *Write*() или асинхронные методы *BeginWrite*(). Вы можете получать данные, используя синхронные методы *Read*() или асинхронные методы *BeginRead*() [2]. Если вы указали *SslStream*, что основной поток следует оставить открытым, вы несете ответственность за закрытие этого потока, когда закончите его использовать. Если приложение, которое создает объект *SSLStream*, выполняется с учетными данными обычного пользователя, приложение не сможет получить доступ к сертификатам, установленным в хранилище локального компьютера, если пользователю явно не было предоставлено разрешение на это.

Вывод. В результате проделанной работы было создано программное обеспечение для системы видеонаблюдения на основе одноплатного компьютера, отличающееся простотой реализации, масштабируемостью функционала и безопасностью.

Список литературы:

1. Олифер В.Г., Олифер Н.А. Компьютерные сети: принципы, технологии, протоколы. СПб, 2008.
2. Официальный сайт документации класса SslStream. Режим доступа: https://docs.microsoft.com/en-us/dotnet/api/system.net.security.sslstream