Нина Кузьмина

Реализация TCP- и UDP-сокетов на контроллере FASTWEL CPM723-01 в среде разработки CODESYS V3

Введение

В распределённых системах управления обмен данными является одним из ключевых моментов работы системы.

Новый контроллер модульной линейки FASTWEL I/O CPM723-01 (рис. 1) позволяет отправлять и получать данные по промышленному протоколу Modbus TCP на базе протокола TCP/IP с использованием двух портов Ethernet и по протоколу Modbus RTU/ACSII на базе последовательных сетей RS-485/RS-232 с помощью коммуникационных модулей NIM741/NIM742. Кроме того, система исполнения контроллера CPM723-01 поддерживает механизм сетевого обмена данными между контроллерами, принадлежащими одной подсети, средствами специального протокола прикладного уровня CODESYS V3 [1]. Но иногда возникает необходимость использования протоколов низкого уровня, которые позволяют обмениваться большим количеством сообщений между различными устройствами с помощью стека TCP/IP. Также на базе данного стека можно создавать протоколы более высокого уровня модели OSI (рис. 2) [2].

TCP/IP основывается на соединениях, устанавливаемых между двумя устройствами, обычно называемыми *клиентом* и *сервером*. Взаимодействие между устройствами в рамках стека TCP/IP осуществляется с помощью связки IP-адреса и порта. Пара адрес и порт образует *сокет* (от английского socket – «гнездо») [3]. Сокет является программным интерфейсом, который обеспечивает обмен данными между устройствами на низком уровне (рис. 3). Различают сокет клиента и сокет сервера.

Рис. 1. Новый контроллер модульной линейки FASTWEL I/O CPM723-01

Для протокола версии IPv4 IP-адреса записываются в 32-битной форме, представляемой в виде mmm.nnn.ppp.qqq – адрес, разбитый на четыре поля, разделённых точками, по одному байту в поле, например, 192.168.102.101 [2]. Номер порта задаётся в диапазоне от 0 до 65535.

Рис. 2. Пример протоколов стека TCP/IP в соответствии с моделью OSI

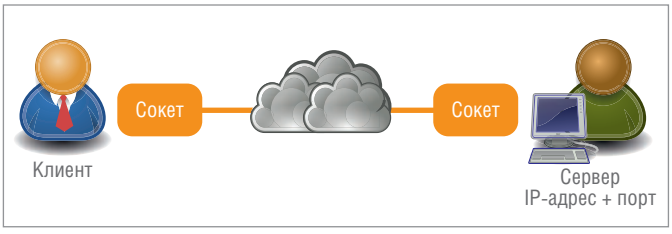

Рис. 3. Общение с помощью сокетов

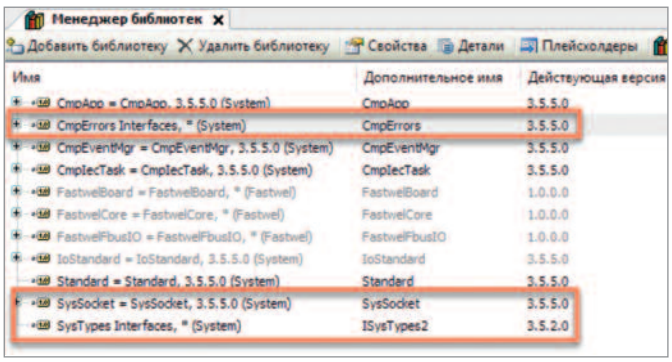

Рис. 4. Библиотеки CODESYS V3, используемые для реализации сокетов

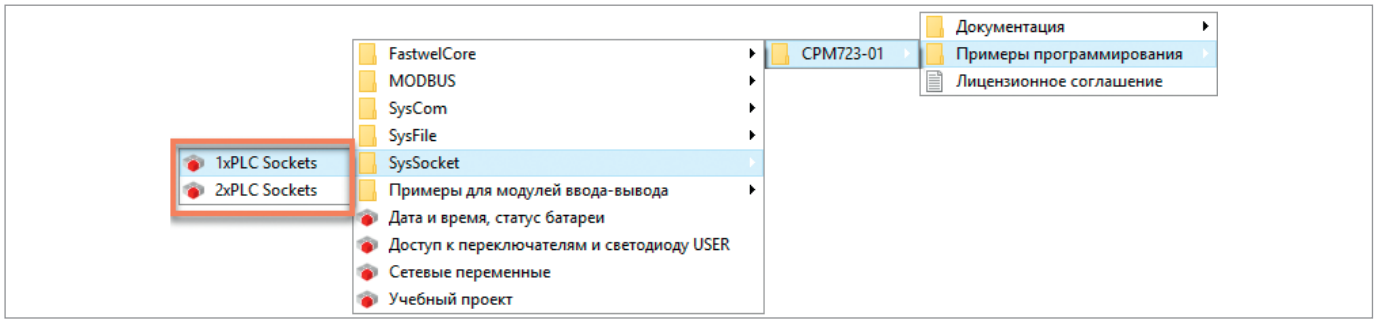

Рис. 5. Примеры для библиотеки SysSocket на базе контроллера CPM723-01

Для организации общения клиент должен знать IP-адрес и номер порта сервера, по которым он подключается к удалённому устройству. В рамках стека протоколов TCP/IP различают два типа сокетов – TCP и UDP [4]. Также TCP-сокеты называют *потоковыми*, а UDP – *датаграммными*.

Реализация сокетов в CODESYS V3

Системные библиотеки

Реализовать работу TCP- и UDP-сокетов на базе контроллера CPM723-01 можно с помощью системной библиотеки *SysSocket*. Эта библиотека включена в комплект поставки среды разработки CODESYS V3 и позволяет создавать сокеты на всех устройствах, поддерживающих данную платформу (рис. 4).

В качестве дополнительных библиотек в проект должны быть включены *CmpErrors* и *ISysTypes2 Interfaces* (рис. 4).

Библиотека *ISysTypes2 Interfaces* необходима для создания системных идентификаторов *hServer* и *hClient*, представленных типом *RTS_IEC_HANDLE*. Кроме того, некоторые функции *SysSocket* возвращают результат (переменная *result*) в виде кодов ошибок, представленных типом *RTS_IEC_RESULT*, которые также декларированы в системной библиотеке *ISysTypes2 Interfaces*.

Тип *RTS_IEC_RESULT* предназначен для передачи приложению кода ошибки вызова системной функции. Перечень кодов ошибок находится в библиотеке *CmpErrors* в списке констант *Errors*.

Пример сокетов для CPM723-01

Текущая версия пакета адаптации CODESYS V3 для контроллеров FASTWEL 1.0.1.0 (от 29.12.2017) содержит готовые примеры программирования для библиотеки SysSocket (рис. 5): проект для одного контроллера и для двух контроллеров для TCP- и UDP-сокетов, в которых можно посмотреть полностью реализованный код для контроллера. Также готовые примеры реализации сокетов для контроллера CPM723-01 и CODESYS V3 можно скачать по ссылке: ftp://ftp.prosoft.ru/pub/Hardware/Fastwel/Fastwel_IO/AppNotes/AN-0001/.

TCP-сокеты

TCP-сокеты используют TCP-соединения, в которых на транспортном уровне (рис. 2) обеспечивается надёжная доставка данных. TCP-протокол отвечает за установление и поддержание соединения, сегментацию, доставку и буферизацию данных, упорядочивание и избавление от дублированных TCP-сегментов данных, контроль ошибок и скорости передачи данных [5]. Схема работы простого TCP-сокета [6] представлена на рис. 6.

В процессе обмена данными с помощью сокетов участвуют две стороны: сервер и клиент. Рассмотрим вначале работу серверного TCP-сокета. На рис. 6 слева представлена блок-схема работы простого серверного TCP-сокета. Для удобства использованы функции из библиотеки *SysSocket* среды разработки CODESYS V3.

Серверный TCP-сокет

При старте программы, отвечающей за управление сервером, вначале происходит инициализация сокета. Данный процесс осуществляется один раз.

С помощью функции *SysSockCreate()* создаётся системный идентификатор («хэндл» – от английского handle) сокета. Данная функция в качестве входных параметров принимает аргументы, задающие тип и протокол сокета. Для использования TCP-протокола функция *SysSockCreate()* должна получить входные аргументы, как показано в листинге 1.

Далее сокет сервера привязывается к определённому IP-адресу и порту с помощью функции *SysSockBind()*. Для привяз-

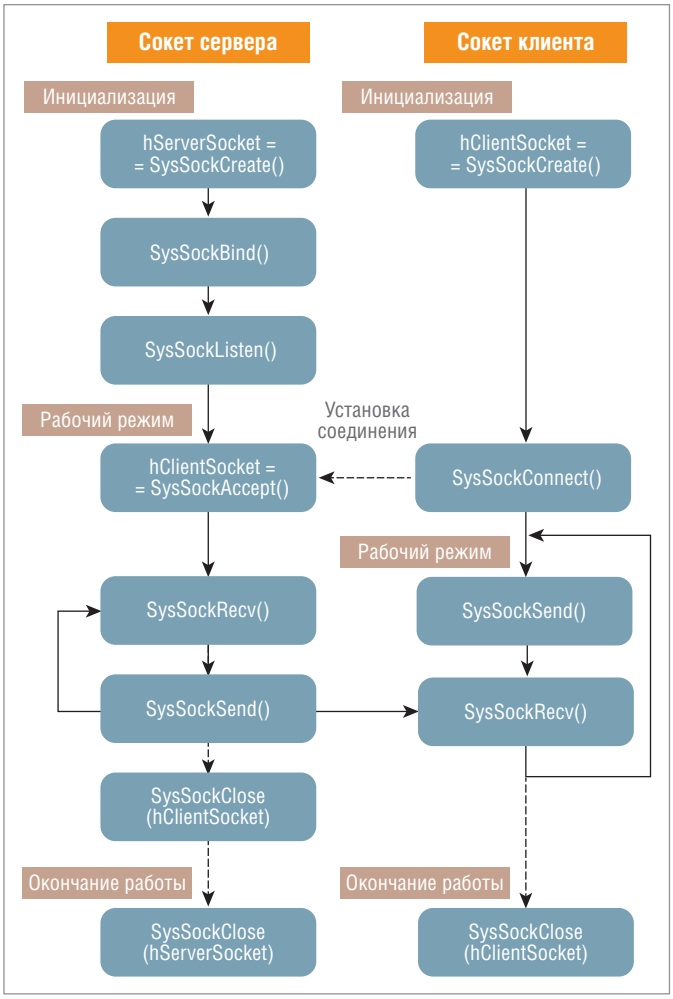

Рис. 6. Работа простых клиентского (справа) и серверного (слева) TCP-сокетов

Листинг 1. Создание серверного TCP-сокета

```
hServerSocket := SysSockCreate(SOCKET_AF_INET, SOCKET_STREAM, SOCKET_IPPROTO_TCP, ADR(result));
// hServerSocket – системный идентификатор типа RTS_IEC_HANDLE, создаваемый SysSockCreate;
// SOCKET_AF_INET задаёт сетевой протокол версии IPv4
// SOCKET_STREAM определяет тип создаваемого сокета, в данном случае потоковый режим (TCP);
// SOCKET_IPPROTO_TCP определяет протокол сокета, в данном случае TCP;
// ADR(result) – указатель на системный идентификатор (handle) результата функции;
// result имеет тип структуры RTS_IEC_RESULT и содержит коды ошибок, возникающих
// при работе с сокетами
```

Листинг 2. Задание адреса серверного TCP-сокета

```
// Задание адреса сокета для подключения к клиенту
// Выбор семейства протоколов: SOCKET_AF_INET соответствует IPv4 serverAddress.sin_family := SOCKET_AF_INET;

// Указывается IP-адрес сервера, с которым будет связан сокет
// IP := '10.0.0.100';
result := SysSockInetAddr(IP,ADR(serverAddress.sin_addr.ulAddr));

// Указывается порт сокета
// port := 503;
serverAddress.sin_port := SysSockHtons(port);
```

Листинг 3. Связывание серверного TCP-сокета с адресом

```
result := SysSockBind(hServerSocket, ADR(serverAddress), SIZEOF(serverAddress));
// Дескриптор серверного сокета hServerSocket связывается с адресом сокета serverAddress,
// описываемым структурой SOCKADDRESS.
```

Листинг 4. Старт прослушивания серверного TCP-сокета

```
result := SysSockListen(hServerSocket, maxConnections);
// hServerSocket – системный идентификатор серверного сокета;
// maxConnections – максимальное количество входящих соединений, например maxConnections := 3;
```

Листинг 5. Создание системного идентификатора клиентского TCP-сокета

```
hClientSocket := SysSockAccept(hServerSocket, ADR(clientAddress), ADR(adressSize), ADR(result));
// hclientSocket – системный идентификатор клиентского сокета;
// clientAddress – структура SOCKADDRESS, где хранится адрес клиента;
// ADR(adressSize) – указатель на размер структуры SOCKADDRESS (тип DINT).
```

Листинг 6. Получение данных от TCP-клиента

```
bytesRead := SysSockRecv(hClientSocket, ADR(recvBuffer), SIZEOF(recvBuffer), 0, ADR(result));
// bytesRead – количество полученных байт сообщения. В случае ошибки возвращается 0;
// hClientSocket – системный идентификатор клиентского сокета;
// ADR(recvBuffer) – указатель на переменную, в которой сохраняется принимаемое сообщение;
// SIZEOF(recvBuffer) – размер принимаемого сообщения;
// Вместо 0 могут быть установлены дополнительные опции для приёма сообщений
// (подробнее в описании функции в библиотеке SysSocket);
// ADR(result) – указатель на идентификатор результата.
```

Листинг 7. Отправка данных TCP-клиенту

```
bytesSend := SysSockSend(hClientSocket, ADR(sendBuffer), SIZEOF(sendBuffer), 0, ADR(result));
// bytesSend – количество отправленных байт. В случае ошибки возвращается 0;
// hClientSocket – системный идентификатор клиентского сокета;
// ADR(sendBuffer) указатель на переменную, которая содержит отправляемое сообщение;
// SIZEOF(sendBuffer) – размер принимаемого сообщения;
// Вместо 0 могут быть установлены опции приёма сообщений;
// ADR(result) – указатель на идентификатор результата.
```

Листинг 8. Закрытие TCP-сокета сервера

```
SysSockClose(hServerSocket);
// hServerSocket – системный идентификатор серверного сокета.
```

ки к определённому IP-адресу функция *SysSockBind()* ссылается на структуру *SOCKADDRESS*, в которой хранится заданный адрес сокета для привязки.

Адрес сокета сервера задаётся один раз при инициализации (листинг 2). Переменная *serverAddress* имеет тип *SOCKADDRESS*. При инициализации необходимо выбрать семейство протоколов, указать IP-адрес (записанный в виде текстовой переменной типа STRING, например, '10.0.0.100') и номер порта (переменная типа WORD, например, 503).

После этого вызывается функция *SysSockBind()*, которая привязывает сокет к данному адресу (листинг 3).

При успешной привязке к адресу функция *SysSockListen()* включает прослушивание входящих соединений (ожидание подключений клиентов к серверу). Функцией *SysSockListen()* также определяется максимальное количество подключений к серверу (листинг 4). Например, если максимальное количество подключений равно 3 и все три клиента уже подключились к серверу, то четвёртому будет отказано в подключении.

Как только сервер включает режим прослушивания, он переходит в рабочий режим и ждёт входящие соединения от клиентов. Когда клиент подключается к сокету сервера, с помощью функции *SysSockAccept()* создаётся системный идентификатор клиентского сокета *hClientSocket* и соединение считается открытым (листинг 5).

Сообщения принимаются серверным сокетом с помощью функции *SysSockRecv()*. В данной функции задаётся указатель на переменную, куда будут сохраняться принимаемые сообщения (листинг 6).

После приёма сообщений сервер может отправить ответ. Это осуществляется с помощью функции *SysSockSend()*. В данной функции задаётся указатель на переменную, в которой хранятся отправляемые данные (листинг 7).

После успешных приёма и передачи данных возможны несколько вариантов поведения программы.

1. Программа может закрыть клиентское соединение. В этом случае в следующих циклах программы сервер будет ожидать подключения с новым клиентом. Такой режим работы не является эффективным, так как контроллеру придётся во время каждого цикла закрывать клиентское соединение и подключать нового клиента (или того же самого, что и в предыдущем цикле).
2. Программа может не закрывать клиентский сокет, а сохранить установленное соединение. В этом случае, один раз установив соединение, клиент будет постоянно отправлять и получать данные от сервера. Такой режим работы более эффективный, но может возникнуть ситуация, когда все клиентские соединения будут заняты и новый клиент не сможет подключиться к серверу. Решить данную ситуацию можно различными способами. Один из вариантов – наблюдать за последним временем активности клиентских сокетов и отключать самое старое соединение в случае, если в очереди обнаружился новый клиент (рис. 7).

В рабочем режиме серверный сокет всегда остаётся открытым. Закрытие серверного сокета может происходить при внешнем событии или при возникновении критических ошибок.

Ошибки при создании и работе сокетов отображаются в системном идентификаторе *result*, который имеет тип структуры *RTS_IEC_RESULT*. Обозначение кодов ошибок описано в системной библиотеке *CmpErrors Interfaces* в глобальных константах *Errors* (рис. 8).

Закрывает сокетное соединение функция *SysSockClose()* (листинг 8).

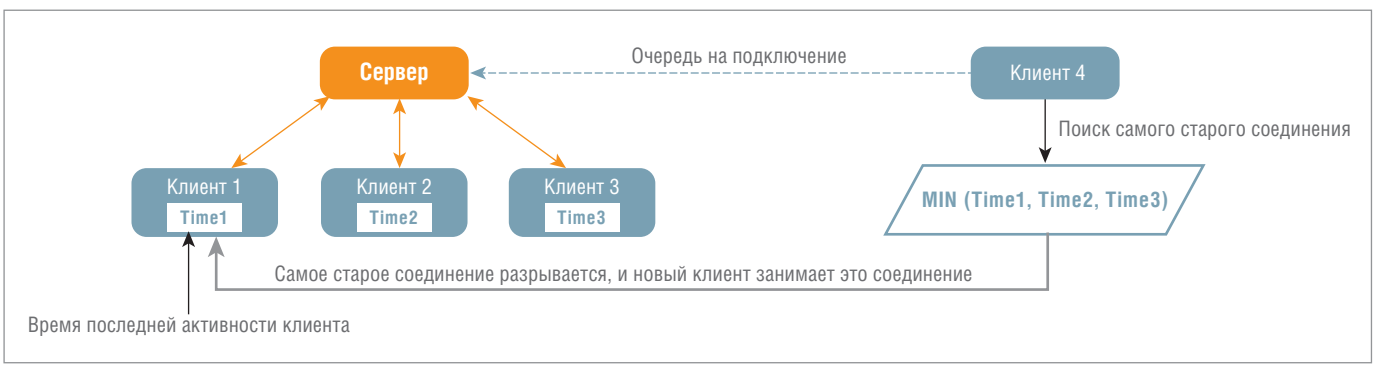

Рис. 7. Обработка подключения нового клиента

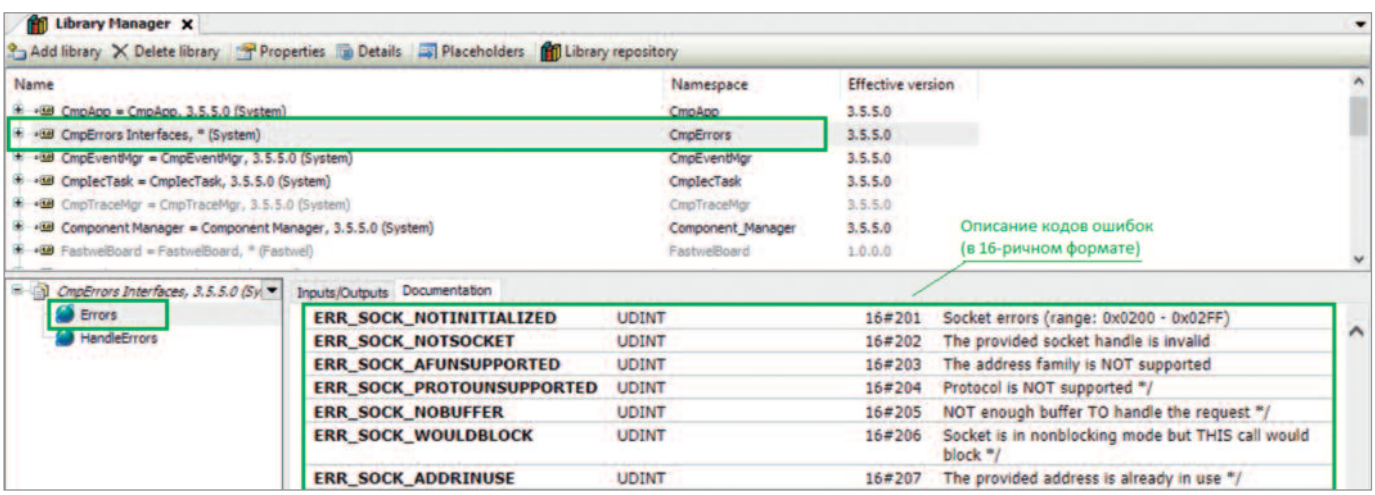

Рис. 8. Расшифровка кодов ошибок работы сокетов

Клиентский TCP-сокет

Схема работы клиентского сокета отображена на рис. 6 справа. Так же как и в случае серверного сокета, клиентский сокет вначале создаётся функцией *SysSockCreate()*, в результате которой создаётся системный идентификатор сокета (листинг 9).

Для подключения клиент должен знать IP-адрес и порт сервера, который хранится в переменной *clientAddress* (листинг 10).

После этого клиент переходит в рабочий режим и начинает обмениваться данными с сервером (листинг 11).

Обмен данными между клиентом и сервером осуществляется с помощью функций *SysSockSend()* и *SysSockRecv()*. Данные функции отправляют сообщения серверу и получают от него ответ (листинг 12).

После обмена данными сокет может быть закрыт с помощью *SysSockClose()* (листинг 13).

Однако, с точки зрения циклического обмена данными реального времени, закрытие сокета при каждом цикле нецелесообразно. Поэтому после успешной установки соединения обмен данными осуществляется в бесконечном цикле, до тех пор пока по какой-то причине не появляется необходимость закрыть сокет.

Особенности TCP-сокетов

Использование TCP-сокетов позволяет приложениям клиента и сервера обмениваться данными почти прозрачно, не заботясь о поддержании сетевого соединения, доставке пакетов по сети, порядке передачи пакетов и буферизации. TCP-сокеты гарантируют доставку сообщений и правильный порядок пакетов, а также пересылают пакеты повторно, если подтверждение о передаче не приходит в течение определённого промежутка времени [4]. Таким образом, использовать TCP-сокеты уместно там, где необходима гарантированная доставка данных.

Несмотря на многие преимущества, TCP-сокеты имеют и негативные стороны. Например, необходимость поддержания TCP-соединения уменьшает пропускную способность обмена данными в распределённых системах. Также в системах обмена данными реального времени повторная передача потерянных пакетов может привести к тому, что система получит данные, которые утратили свою актуальность.

UDP-сокеты

Все перечисленные недостатки TCP-сокетов связаны с особенностью TCP-протокола. Если в системе присутствие дан-

Листинг 9. Создание клиентского TCP-сокета

```
hClientSocket := SysSockCreate(SOCKET_AF_INET, SOCKET_STREAM, SOCKET_IPPROTO_TCP,ADR(result));
// hClientSocket – системный идентификатор клиентского сокета;
// SOCKET_AF_INET задаёт сетевой протокол IPv4;
// SOCKET_STREAM определяет тип сокета, в данном случае потоковый сокет (TCP);
// SOCKET_IPPROTO_TCP определяет протокол сокета, в данном случае TCP;
// ADR(result) – указатель на системный идентификатор (handle) результата функции.
```

Листинг 10. Задание адреса для подключения клиентского TCP-сокета к TCP-серверу

```
// Задаётся адрес сокета для подключения
// Выбор семейства протоколов: SOCKET_AF_INET соответствует IPv4
clientAddress.sin_family := SOCKET_AF_INET;

// Задание порта сокета для подключения
// port := 503;
clientAddress.sin_port:=SysSockHtons(port);

// Задание IP-адреса сервера
// IP := '10.0.0.100';
result := SysSockInetAddr(IP, ADR(clientAddress.sin_addr));
```

Листинг 11. Подключение клиентского сокета к серверу

```
result := SysSockConnect(hClientSocket, ADR(clientAddress), SIZEOF(clientAddress));
// hClientSocket – системный идентификатор клиентского сокета;
// ADR(clientAddress) – указатель на структуру SOCKADDRESS с адресом сокета сервера;
// SIZEOF(clientAddress) – размер структуры адреса сокета.
```

Листинг 12. Обмен данными между TCP-клиентом и TCP-сервером

```
bytesSend := SysSockSend(hClientSocket, ADR(sendMessage), SIZEOF(sendMessage), 0,ADR(result));
bytesRecv := SysSockRecv(hClientSocket, ADR(recvMessage), SIZEOF(recvMessage), 0, ADR(result));
// hClientSocket – системный идентификатор клиентского сокета;
// ADR(sendMessage/recvMessage) – указатель на отправляемое/полученное сообщение;
// SIZEOF(sendMessage/recvMessage), размер отправленного/полученного сообщения;
// ADR(result) – указатель на системный идентификатор (handle) результата функции.
```

Листинг 13. Закрытие клиентского TCP-сокета

```
SysSockClose(hClientSocket);
// где hClientSocket – системный идентификатор серверного сокета.
```

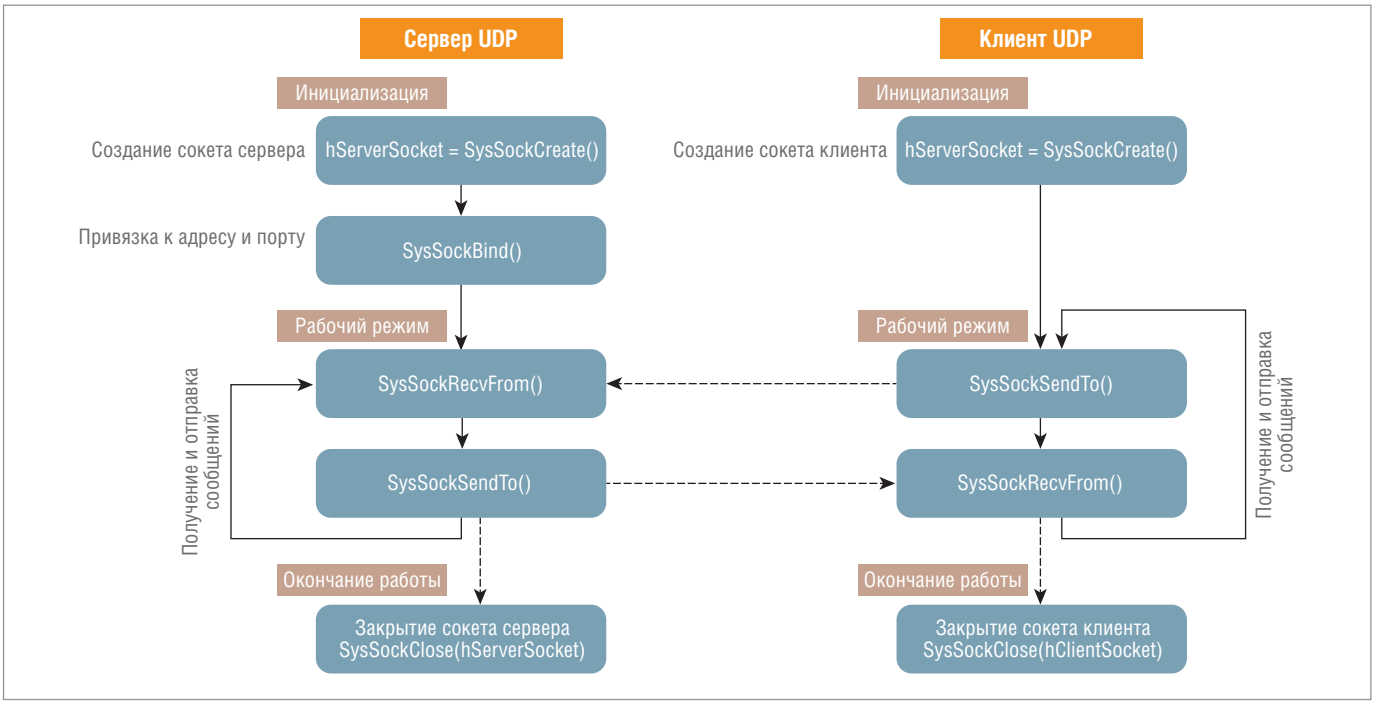

Рис. 9. Схема работы простых UDP-сокетов: серверного (слева) и клиентского (справа)

ных факторов крайне нежелательно, а гарантированность доставки сообщений не является критичным требованием, то в качестве альтернативы TCP-сокетов могут использоваться UDP-сокеты.

UDP-сокеты устроены проще, чем TCP. В качестве транспортного уровня используется протокол UDP, который не требует установления соединения и подтверждения приёма [4]. Информация пересылается в предположении, что принимающая сторона её ожидает. Датаграммные сокеты не контролирует ничего, кроме целостности полученных датаграмм. Несмотря на это, UDP-сокеты нашли своё применение в системах, где на первом месте стоит именно актуальность данных и их быстрая доставка, а не гарантия доставки каждого сообщения.

Например, сервер в ответ на запросы клиента передаёт по сети текущие значения некоторого параметра, а клиент формирует управляющий сигнал на основе принятых значений. Если клиент опрашивает сервер быстрее, чем скорость изменения параметра, то потеря одного-двух UDP-сообщений от сервера существенно не повлияет на качество формирования управляющего сигнала. В случае использования TCP потерянный пакет будет автоматически передан повторно, что может привести к получению клиентом неактуального значения параметра, а значит, к ошибке формирования управляющего сигнала.

На рис. 9 представлена схема работы простых серверных и клиентских UDP-сокетов.

Серверный UDP-сокет

Так же как в случае TCP-сокетов, системный идентификатор UDP-сервера создаётся с помощью функции *SysSockCreate()* (листинг 14).

После создания сокет сервера привязывается к определённому IP-адресу и порту с помощью функции *SysSockBind()*. В отличие от TCP, UDP-сокет не включает прослушивание входящих соединений, а сразу подготавливается к получению данных по сети (листинг 15).

Для получения данных по UDP используется функция *SysSockRecvFrom()* (в отличие от *SysSockRecv()* для TCP-сокетов). Её главная особенность в том, что она не просто принимает данные от клиента, но и записывает адрес и порт клиента в специальную структуру для хранения адреса *SOCKADDRESS*, чтобы в дальнейшем сервер знал, куда отправлять ответное сообщение (листинг 16).

Ответное сообщение отправляется с помощью функции *SysSockSendTo()*, которая аналогична *SysSockSend()* для TCP-протокола, но в качестве аргумента принимает ссылку на адрес структуры *SOCKADDRESS*, где хранится записанный ранее адрес клиента (листинг 17).

После отправки данных сокет сервера снова переходит к функции *SysSockRecvFrom()* и остаётся открытым. Но в случае необходимости серверный сокет можно закрыть аналогично TCP-сокету (листинг 8):

Клиентский UDP-сокет

Клиент UDP работает аналогично клиентскому сокету TCP, за исключением использования функций *SysSockSendTo()* и *SysSockRecvFrom()* для отправки и получения сообщений (рис. 9).

Функция *SysSockCreate()* создаёт системный идентификатор сокета. Так же как и в случае сервера, для клиента необходимо создать потоковый сокет (листинг 18).

В отличие от TCP-сокетов, при использовании UDP-протокола клиентский сокет не устанавливает соединения с сервером, а сразу после создания клиентского сокета переходит к обмену данными с помощью функций *SysSockSendTo()* и *SysSockRecvFrom()* (листинг 19).

После обмена данными сокет может быть закрыт с помощью *SysSockClose()* (листинг 13).

Однако, с точки зрения циклического обмена данными реального времени, каждый раз закрывать и открывать сокет заново неэффективно. Поэтому после успешной установки соединения обмен данными осуществляется в бесконечном цикле.

Дополнительные настройки сокетов

На рис. 6 показана работа простых серверного и клиентского TCP-сокетов. Но на деле такая простая схема имеет некоторые ограничения и недостатки.

Блокирующий режим

По умолчанию некоторые функции библиотеки *SysSocket* являются *блокирующими*. Это значит, что вызов функции не возвращает управление программному коду до тех пор, пока он не выполнится. Блокирующими функциями являются *SysSockAccept()*, *SysSockSend()*, *SysSockRecv()*, *SysSockSendTo()*, *SysSockRecvFrom()* и так далее.

Например, сервер включает прослушивание входящих соединений с помощью неблокирующей функции *SysSockListen()*, сразу после которой идёт вызов *SysSockAccept()*. И до тех пор пока в очереди установленных соединений не появится хотя бы одно подключение, программа не будет исполняться дальше. Такой режим работы также называется синхронным.

Естественно, такое поведение программы не является безопасным, и при циклическом вызове программы в ПЛК может сработать сторожевой таймер или произойти выход в безопасный режим – контроллер будет считать, что программа зависла.

Листинг 14. Создание серверного UDP-сокета

```
hServerSocket := SysSockCreate(SOCKET_AF_INET, SOCKET_DGRAM, SOCKET_IPPROTO_UDP, ADR(result));
// hServerSocket – системный идентификатор сокета сервера;
// SOCKET_AF_INET – семейство, соответствующее сетевому протоколу IPv4;
// SOCKET_DGRAM – тип создаваемого сокета (датаграммный сокет для UDP);
// SOCKET_IPPROTO_UDP – протокол сокета UDP;
// ADR(result) – указатель на идентификатор результата (RTS_IEC_RESULT).
```

Листинг 15. Связывание серверного UDP-сокета с адресом

```
result := SysSockBind(hServerSocket, ADR(serverAddress), SIZEOF(serverAddress));
// Дескриптор серверного сокета hServerSocket связывается с адресом сокета;
// serverAddress, заданным структурой SOCKADDRESS.
```

Листинг 16. Получение данных от UDP-клиента

```
bytesRead := SysSockRecvFrom(hServerSocket, ADR(recvBuffer), SIZEOF(recvBuffer), 0,
                         ADR(clientAddress), SIZEOF(clientAddress),ADR(result));
// hServerSocket – системный идентификатор сокета сервера;
// bytesRead – количество полученных байт, в случае ошибки возвращается 0;
// hServerSocket – системный идентификатор клиентского сокета;
// ADR(recvBuffer) – указатель на переменную, в которую запишется получаемое сообщение;
// SIZEOF(recvBuffer) – размер принимаемого сообщения;
// Вместо 0 могут быть установлены опции приёма сообщений;
// ADR(clientAddress) – указатель на структуру SOCKADDRESS, в которую запишется адрес клиента;
// ADR(result) – указатель на идентификатор результата.
```

Листинг 17. Отправка данных UDP-клиенту

```
bytesSend := SysSockSendTo(hServerSocket, ADR(sendBuffer), SIZEOF(sendBuffer), 0,
                         ADR(clientAddress), SIZEOF(clientAddress),ADR(result));
// ADR(sendBuffer) и SIZEOF(sendBuffer) – указатель на отправляемое сообщение
// и размер отправляемого сообщения;
// ADR(clientAddress) – указатель на структуру, в которой записан адрес сокета клиента.
```

Листинг 18. Создание клиентского UDP-сокета

```
hClientSocket := SysSockCreate(SOCKET_AF_INET, SOCKET_DGRAM, SOCKET_IPPROTO_UDP, ADR(result));
// hClientSocket – системный идентификатор клиентского сокета;
// SOCKET_AF_INET – семейство, соответствующее сетевому протоколу IPv4;
// SOCKET_DGRAM – тип создаваемого сокета (датаграммный сокет для UDP);
// SOCKET_IPPROTO_UDP – протокол сокета (UDP);
// ADR(result) – указатель на идентификатор результата (RTS_IEC_RESULT).
```

Листинг 19. Обмен данными между UDP-клиентом и UDP-сервером

```
bytesSend := SysSockSendTo(hClientSocket, ADR(sendMessage), SIZEOF(sendMessage), 0,
                         ADR(clientAddress), SIZEOF(clientAddress), ADR(result));

bytesRecv := SysSockRecvFrom(hClientSocket, ADR(recvMessage), 256, 0,
                         ADR(clientAddress), SIZEOF(clientAddress), ADR(result));
// hClientSocket – системный идентификатор сокета сервера;
// bytesRead, bytesRecv – количество полученных и отправленных байт,
// в случае ошибки возвращается 0;
// ADR(clientAddress) – указатель на структуру SOCKADDRESS, в которую запишется адрес клиента;
// ADR(result) – указатель на идентификатор результата.
```

Листинг 20. Включение неблокирующего режима

```
result := SysSockIoctl(hSocket, SOCKET_FIONBIO, ADR(mode));
// hSocket – системный идентификатор сокета клиента или сервера;
// ADR(mode) – указатель на переменную, включающую опцию (значение переменной равно 16#1).
```

Листинг 21. Дополнительные настройки работы сокета

```
result := SysSockSetOption(hSocket, SOCKET_SOL, SOCKET_SO_REUSEADDR, ADR(mode), SIZEOF(mode));
// hSocket – системный идентификатор сокета клиента или сервера;
// SOCKET_SOL – уровень протокола, соответствующий уровню сокетов;
// SOCKET_SO_REUSEADDR – опция, позволяющая связать сокет с локальным адресом,
// который уже используется на другом открытом сокете;
// ADR(mode) – указатель на переменную, включающую опцию (значение переменной 16#1).
```

Листинг 22. Работа с несколькими клиентами

```
result := SysSockSelect(maxConnections+1, ADR(readSet), ADR(writeSet), ADR(exceptSet),
                                                ADR(timeSelect), ADR(socketReady));
// maxConnections+1 – количество проверяемых дескрипторов. В качестве аргумента
// устанавливается максимальное количество соединений + 1;
// ADR(readSet), ADR(writeSet), ADR(exceptSet) – указатель на набор дескрипторов
// (SOCKET_FD_SET), которые следует проверять на готовность к чтению, записи и
// наличию исключительных ситуаций;
// SysSockSelect() является блокирующей функцией, она возвращает управление, если
// хотя бы один из проверяемых сокетов готов к выполнению соответствующей операции;
// ADR(timeSelect) – указатель на интервал времени timeSelect, по прошествии которого
// функция вернёт управление в любом случае;
// timeSelect имеет структуру SOCKET_TIMEVAL и задаёт максимальное время, которое
// функция SysSockSelect будет ожидать для получения ответа;
// ADR(socketReady) – указатель на количество сокетов, готовых к работе, которое
// возвращает функция SysSockSelect.
```

Для того чтобы использовать функции в неблокирующем режиме, необходимо после создания сокета *SysSockCreate()* вызвать функцию *SysSockIoctl()* с входным аргументом *SOCKET_FIONBIO* (листинг 20), которая является командой перевода сокета в неблокирующий режим. При неблокирующем (асинхронном) режиме функция возвращает управление программе вне зависимости от того, закончена операция приёма/передачи или нет (рис. 10).

Также дополнительные настройки работы сокета можно сделать с помощью функции *SysSockSetOptions()*, например, включить возможность повторного использования порта (листинг 21).

Подключение нескольких клиентов

Серверный TCP-сокет, работающий согласно схеме на рис. 5, подходит для обмена данными в режиме точка–точка, когда существует одно входящее клиентское соединение. В случае если к серверу будет подключаться несколько клиентов, может возникнуть путаница с принимаемыми и отправляемыми сообщениями, а также может появиться очередь на ожидание подключения.

Для того чтобы эффективно работать с несколькими клиентами, используется *SysSockSelect()* (листинг 22). Данный метод проверяет состояние нескольких идентификаторов сокетов одновременно. Сокеты можно проверять на готовность к чтению, записи или на наличие исключительных ситуаций, то есть ошибок.

Если хотя бы один сокет клиента готов, например, к отправке данных, *SysSockSelect()* сообщит об этом программе и соединение с данным клиентом будет установлено. Схема работы серверного сокета с использованием *SysSockSelect()* показана на рис. 10.

Функция *SysSockSelect()* является блокирующей, она возвращает управление, если хотя бы один из проверяемых сокетов готов к выполнению соответствующей операции. Но в качестве настройки в функции можно указать интервал времени, по истечении которого она вернёт управление в любом случае.

Особенности использования внешних ресурсов контроллера

При создании сокетов для CPM723-01 необходимо иметь в виду, что система исполнения приложений версии CODESYS V3 не имеет в своём составе подсистемы автоматического учёта и освобождения внешних ресурсов, запрошенных приложением [1]. К таким внешним ресурсам относятся файлы, сокеты и другие системные ресурсы. Для того чтобы был обеспечен повторный доступ к ним, необходимо самостоятельно освобождать полученные системные идентификаторы (в нашем случае *hServerSocket* и *hClientSocket*) в обработчике системного события *PrepareExit*, в котором вызывать действия по освобождению ресурсов, требуемых в программах (рис. 11).

Таким образом, каждый раз перед завершением работы приложения будет закрываться доступ к портам, и при следующем запуске приложения данные порты будут открыты для использования.

Заключение

TCP- и UDP-сокеты отвечают за обмен данными между различными устройствами и процессами. На базе обмена данными по сокетам можно создавать протоколы стека TCP/IP более высокого уровня.

Обмен по TCP и UDP в контроллере CPM723-01 может потребоваться там, где устройство, с которым необходимо орга-

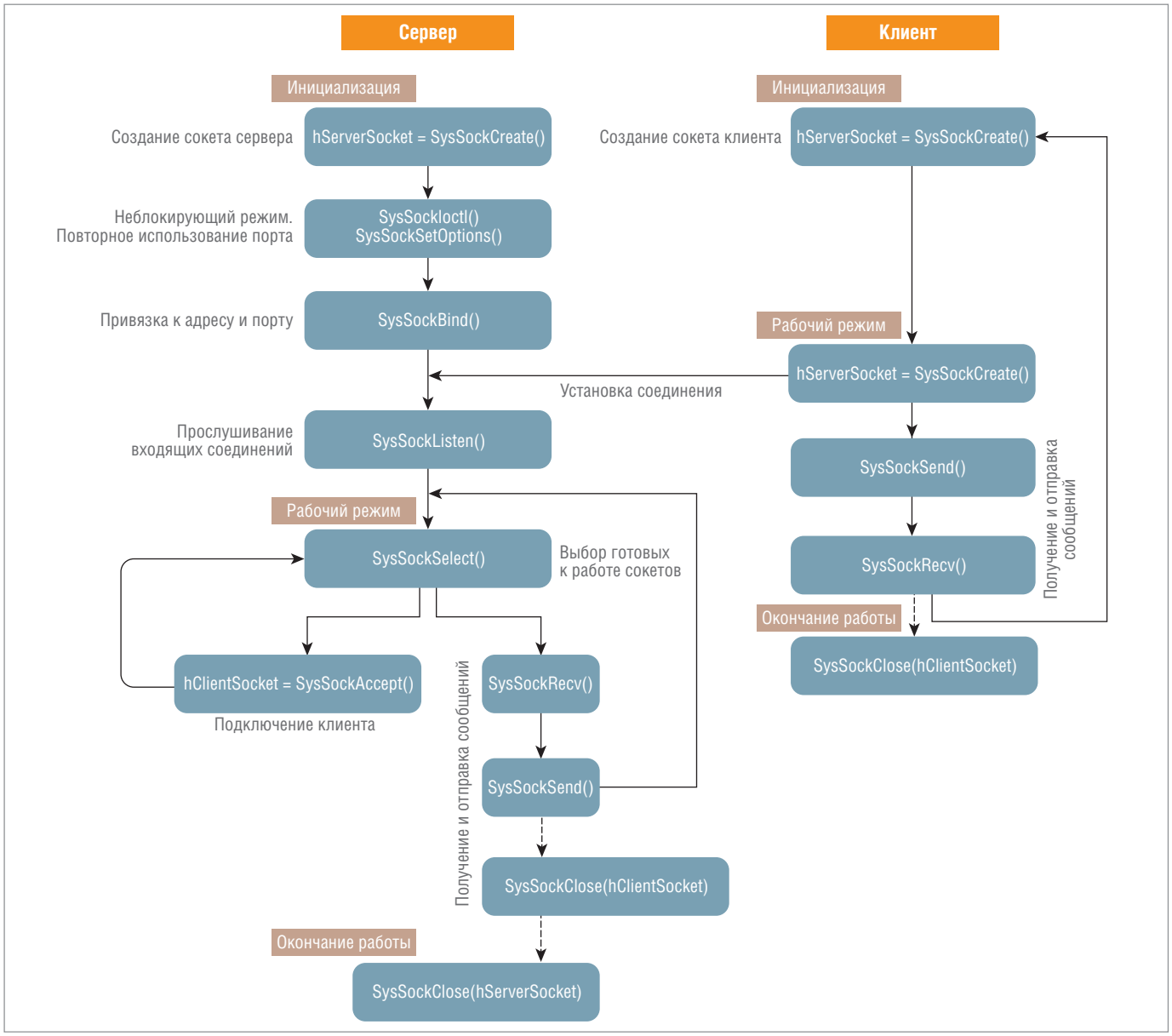

Рис. 10. Схема работы сокетов с использованием неблокирующих опций и функции SysSockSelect()

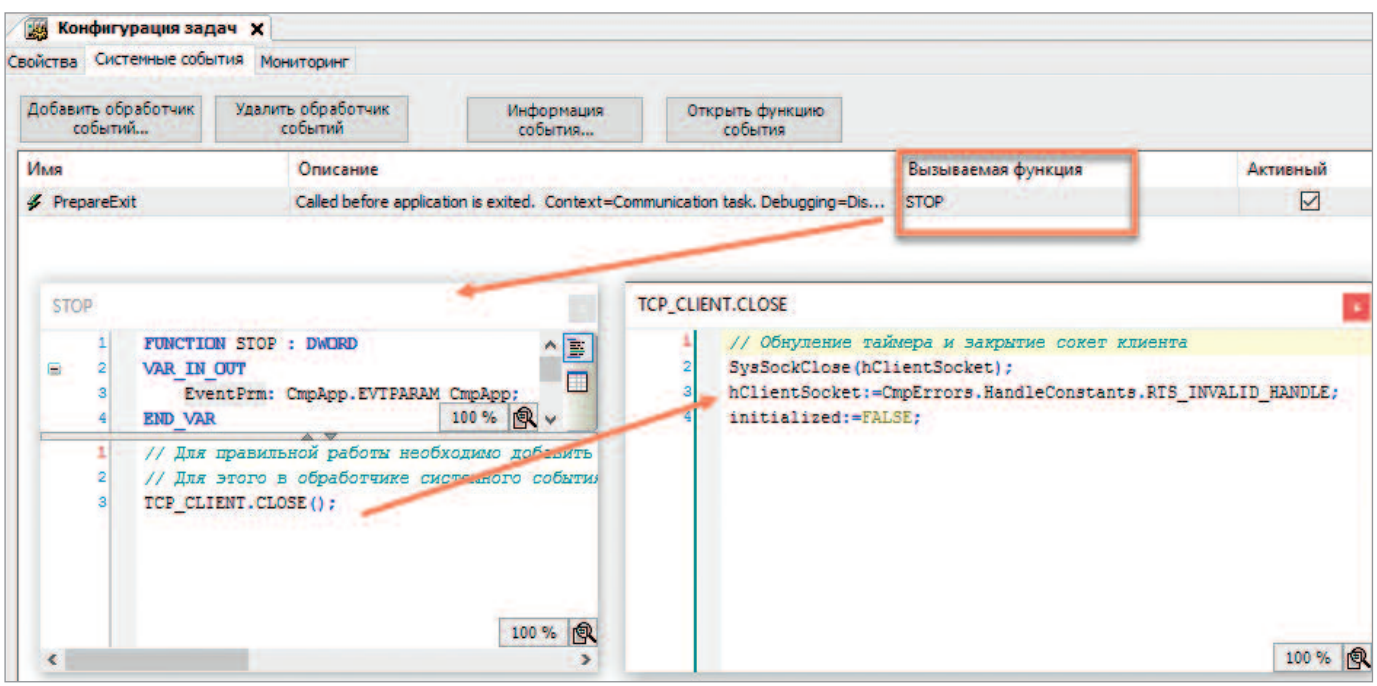

Рис. 11. Системное событие, освобождающее системные идентификаторы

низовать связь, не поддерживает промышленный протокол Modbus TCP.

TCP-сокеты необходимы там, где требуется надёжная доставка сообщений, а скорость передачи данных не критична. UDP-сокеты лучше всего использовать там, где требуется эффективность на быстрых сетях с короткими соединениями и данные реального времени, а гарантированность доставки сообщений не нужна [4]. ●

Литература

1. Система ввода-вывода FASTWEL I/O. Контроллер программируемый CPM723-01.Руководство по конфигурированию и программированию ИМЕС.00300-03 33 02-1. [Электронный ресурс] // Режим доступа : https://tp.prosoft.ru/docs/shared/%D0%A2%D0%B5%D1%85%D0%BF%D0%BE%D1%80%D1%82%D0%B0%D0%BB/Fastwel_IO/Manual/CPM723-01_CDSV3_UM.pdf.
2. Parziale L., Britt D.T., Davis Ch., Forrester J., et al. TCP/IP Tutorial and Technical Overview. – USA : IBM, December 2006.
3. Основы TCP-связи [Электронный ресурс] // Режим доступа : http://www.sbcr.jp/books/img/Linuxnet_02.pdf. – Текст на яп.
4. Fiedler G. UDP vs. TCP. Which protocol is best for games? [Электронный ресурс] // Режим доступа : https://gafferongames.com/post/udp_vs_tcp/.
5. Лейкин А. Протоколы транспортного уровня UDP, TCP и SCTP: достоинства и недостатки // Первая миля. – 2013. – № 5.
6. Network communication under UNIX System Services [Электронный ресурс] // Режим доступа : https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.3.0/com.ibm.zos.v2r3.cbcpx01/ipchap.htm.

Автор – сотрудник фирмы ДОЛОМАНТ
Телефон: (495) 232-1698
E-mail: fio@fastwel.ru

НОВОСТИ НОВОСТИ НОВОСТИ НОВОСТИ НОВОСТИ НОВОСТИ

Новости ISA

4 марта 2018 года на площадке Инженерной школы ГУАП был проведён 2-й этап открытого регионального чемпионата «Молодые профессионалы» (WorldSkills Russia) Санкт-Петербурга по компетенции «Интернет вещей». К участию в соревнованиях были приглашены команды студентов колледжей Санкт-Петербурга и молодые специалисты предприятий города. Возрастная категория от 16 до 22 лет. Чемпионат собрал команды и экспертов по направлению «Интернет вещей» из ведущих средних профессиональных учебных заведений города. Компетенция «Интернет вещей» относится к категории Future Skills (профессии будущего) и направлена на подготовку специалистов по комплексной автоматизации и роботизации производства с использованием самых передовых промышленных технологий. Компетенция создана в 2016 году при поддержке компаний PTC и Fanuc. На церемонии открытия, прошедшей в доброжелательной и деловой атмосфере, к участникам чемпионата с приветствием обратились представитель Регионального координационного центра WorldSkills в Санкт-Петербурге Н.Е. Смирнова и председатель организационного комитета соревнования, начальник Управления информатизации ГУАП, активный член Российской секции ISA А.В. Сергеев.

В период с 16 по 20 апреля в ГУАП прошла 71-я Международная научная студенческая конференция. В рамках мероприятия была проведена XI студенческая конференция ISA. Студенты и аспиранты шести университетов из США, Италии, Испании, Российской Федерации и Индонезии выступили с докладами. Руководил работой конференции профессор университета штата Индиана (США), президент ISA 2009 года Gerald Cockrell. Россию представляли студенты ГУАП Ростислав Шаниязов и Ангелина Добровольская. Решением международного жюри доклады студентов ГУАП признаны лучшими. Студенты и их научные руководители А.В. Сергеев и Н.Н. Майоров награждены почётными дипломами ISA.

Группа компаний (ГК) InfoWatch и ГУАП открыли совместную исследовательскую лабораторию кибербезопасности. Соглашение о создании лаборатории подписали 18 апреля 2018 года на площадке Петербургского цифрового форума ректор ГУАП (президент Российской секции ISA 2014 года) Юлия Анатольевна Антохина и президент ГК InfoWatch Наталья Ивановна Касперская. Помимо подписания соглашения 18 апреля состоялось торжественное открытие лаборатории в Инженерной школе Интернета вещей ГУАП. Лаборатория создана с целью развития инновационной и образовательной деятельности, проведения НИОКР в области защиты информации с использованием технологий, решений, продуктов ГК InfoWatch. Планируется, что лаборатория станет базой для подготовки специалистов по созданной в 2017 году компетенции WorldSkills «Корпоративная защита от внутренних угроз ИБ» в национальном масштабе.


XI студенческая конференция ISA

27–28 апреля 2018 года делегация Российской секции ISA приняла участие в работе Исполкома Европейского совета ISA в Мадриде. На заседании Исполкома обнародованы итоги XIV Европейского конкурса на лучшую студенческую научную работу ISA (ESPC-2018). Студенты и аспиранты ГУАП – члены студенческой секции ISA – в очередной раз показали прекрасные результаты. Золотых медалей удостоены А. Чабаненко, Р. Шаниязов, Б. Акопян, А. Шабанова, М. Шелест, М. Тарала. Серебряные медали вручаются А. Добровольской, Ю. Соколовой, С. Герасимову, А. Виноградову. Бронзовые награды получают Е. Григорьев, Л. Ефимова, Ф. Рыжов, Н. Исакова и Г. Емельянов. Традиционно медали победителям вручены ректором ГУАП, президентом Российской секции ISA 2014 года Ю.А. Антохиной на заседании учёного совета ГУАП 24 мая 2018 года.

Почётными дипломами ISA награждены активные члены Российской секции ISA И.А. Киршина, И.А. Павлов (президент Российской секции ISA 2010 года), Е.Г. Семёнова (президент Российской секции ISA 2011 года), А.С. Будагов (президент Российской секции ISA 2018 года). ●