И.В. ПЕТРОВ (Компания “Пролог”)

Отладка прикладных ПЛК программ в CoDeSys (часть 5)[1]

Существует совершенно особая категория ошибок – это ошибки исполнения. Условия для их возникновения складываются в процессе работы при определенных значениях или комбинациях данных. Обычно возникновение таких значений не предполагалось в процессе разработки. Например, не вызывает сомнения, что возможность переполнения или деления на ноль нужно предвидеть и не допускать. Это обычная практика надежного программирования. Но теоретически мы или разработчики используемых нами библиотек могли где-либо упустить необходимую проверку. Хорошая среда исполнения (операционная система) обязана иметь обработчики подобных ситуаций. Технически это несложно. Главная проблема состоит в том, что универсального ответа на вопрос “что делать при обнаружении попытки деления на 0” не существует. Не обладая информацией о смысле выполняемых действий, нельзя принять правильного решения. Например, можно считать ответом произвольное число и продолжить работу. Для многих программ это может оказаться приемлемым решением. Но иногда это может иметь фатальные последствия. В итоге, очень хорошие системы программирования должны давать возможность перехвата ошибок исполнения в прикладной программе.

В каждом контроллере, поддерживающем программирование в CoDeSys, размещена система исполнения CoDeSys SP. Именно она отделяет аппаратные ресурсы от МЭК программ. Она получает от процессора сообщения об ошибках исполнения и выполняет некоторые действия по умолчанию. Во многом они определяются техническими возможностями аппаратуры и могут быть изменены изготовителем контроллера. В некоторых ПЛК есть специальный светодиодный индикатор ошибок исполнения. В результате мы не можем дать общие описания всех обработчиков ошибок исполнения. Они отличаются в разных целевых платформах, но в любом случае они есть. В обычной практике программисты CoDeSys сосредотачивают усилия на том, чтобы не допускать условий для возникновения таких ошибок. Если при выполнении программы в режиме online произошла ошибка исполнения, то CoDeSys даст соответствующее диагностическое сообщение (рис. 1). Естественно, для этого необходимо намеренно создать условия, приводящие к возникновению ошибки. Это совсем не просто. Автор программы слишком хорошо ее знает, и ему никогда не придет в голову задавать подобные ненормальные условия. Эту работу должны выполнять другие люди. Для генерации тестов существуют специальные методики и программные инструменты. При недостаточном тестировании ошибки исполнения имеют свойство возникать в начале эксплуатации. Перехват их обработчиков в программе позволяет замаскировать их последствия и выявить истинные причины.

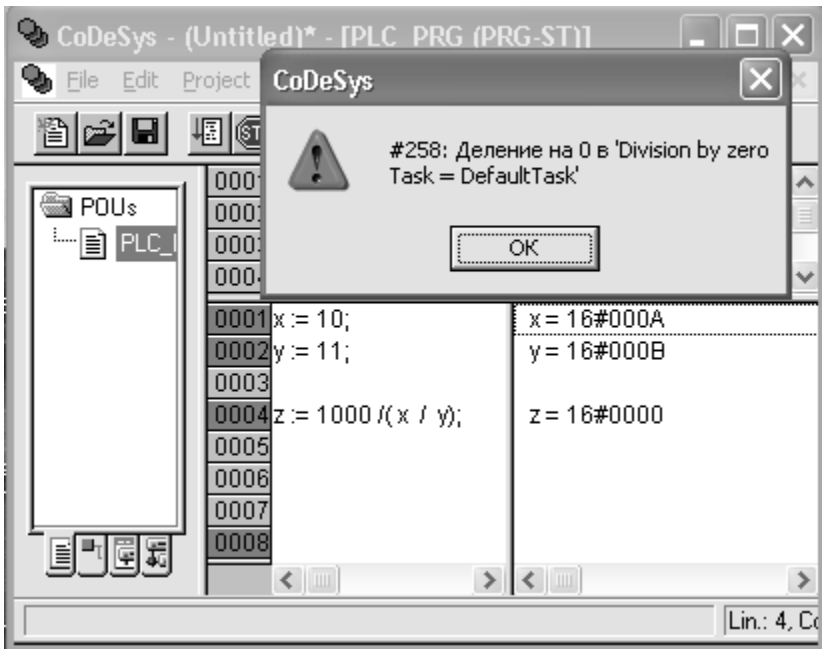

Рис. 1. Система исполнения CoDeSys обнаружила попытку деления на ноль.

Для типовых ошибок времени исполнения CoDeSys позволяет добавить в проект собственные функции-обработчики, которые мы рассмотрим ниже. Это опциональная возможность. Если данные функции отсутствуют в проекте, то по умолчанию будут работать обработчики, имеющиеся в системе исполнения. Создание собственных

[1] *Продолжение. Начало в №№ 2-5, 2006 г.*

 © НАУЧТЕХИЗДАТ 2006г.

обработчиков можно порекомендовать только опытным программистам. В руководстве по программированию в CoDeSys эти возможности намеренно описаны конспективно. Если вы недавно начали работать с CoDeSys, то вам лучше пока отложить данную статью.

Помимо вычислительного ядра любой ПЛК имеет ряд аппаратных узлов, потенциально способных давать сбои (блок питания, модули ввода/вывода, контроллеры сетей и т.д.). Многие из них система исполнения автоматически тестирует при включении питания и в процессе работы. Поскольку такие сбои невозможно исключить заранее, то логично отнести их к ошибкам исполнения. Список диагностических сообщений для конкретного ПЛК включается в состав файлов поддержки целевой платформы (target) CoDeSys. Большой список свидетельствует о наличии развитой системы встроенной самодиагностики контроллера. Так в контроллерах Wago 750 их более 20.

Контроль деления

На большинстве современных процессоров целочисленные вычисления выполняются значительно быстрее, чем операции с данными типа REAL, и применяются очень широко. Нельзя забывать, что в силу ограниченного диапазона значений операции сложения, вычитания и умножения целых чисел могут вызвать переполнение. Кроме того, операция целочисленного деления не замкнута на множестве целых чисел, то есть при делении целых точный результат не всегда можно выразить целым числом. Кроме снижения точности результата, это может иметь и более глубокие последствия. Например:

```
uiX := 10;
uiY := 11;

IF uiY <> 0 THEN
      uiZ := 1000/(uiX / uiY);
END_IF
```

Где ошибка в этой конструкции? На первый взгляд все очень хорошо. Но округление всегда выполняется вниз. Если делимое меньше делителя, то частное всегда будет равно нулю. То есть 10/11 = 0.

Обратите внимание на то, что эмулятор контроллера в CoDeSys при делении на ноль дает максимально возможный результат (16#FFFF для INT и UINT) и не генерирует исключение. Для проверки примера необходимо использовать любой реальный контроллер или демонстрационную версию PLC WinNT, включенную в стандартный дистрибутив (рис. 2). Результат попытки выполнения этого примера показан на рис. 1.

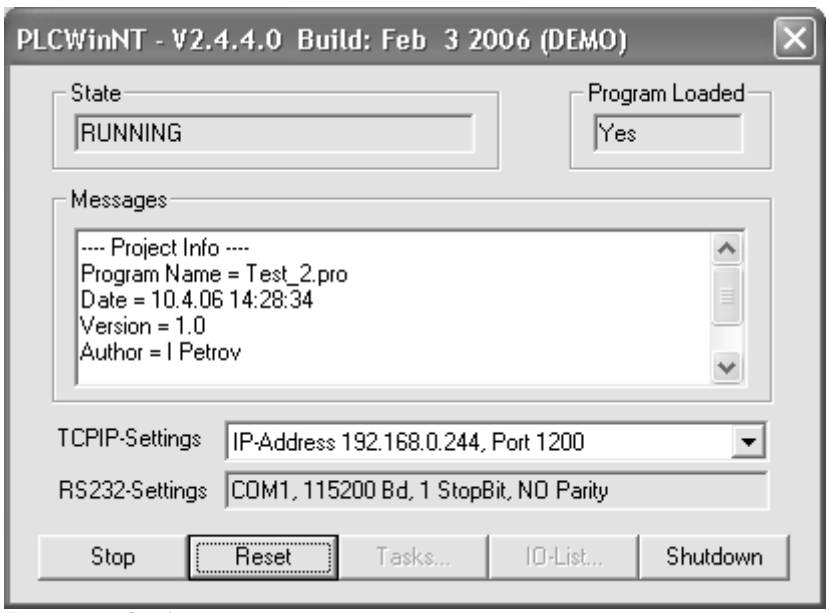

Рис. 2. PLC WinNT.

При использовании целочисленного деления необходимо учитывать эффект округления. Для локализации вычислительных ошибок в программах желательно иметь функцию, позволяющую контролировать деление. С этой целью в CoDeSys предусмотрены функции *CheckDivByte, CheckDivWord, CheckDivDWord*. Они позволяют контролировать деление 8-, 16- и, соответственно, 32-разрядных данных. Так, если в проект включить функцию *CheckDivWord*, то она будет вызываться при делении переменных типов INT, UINT и WORD (рис. 3). Функция по-

лучает в качестве параметра значение делителя. В операции деления будет использовано значение делителя, возвращаемое функцией. В простейшем случае она может выглядеть так:

```
FUNCTION CheckDivWord : WORD
VAR_INPUT
    divisor: WORD;
END_VAR
IF divisor = 0 THEN
    CheckDivWord := 1;
ELSE
    CheckDivWord := divisor;
END_IF
```

Если делитель имеет нулевое значение, то мы заменяем его единицей. Конечно, частное будет неверным, но, по крайней мере, мы исключили нежелательный останов программы при подобных сбоях. Вы можете дополнить функцию проверкой подозрительно больших значений делителя для конкретной задачи и формированием флагов ошибок.

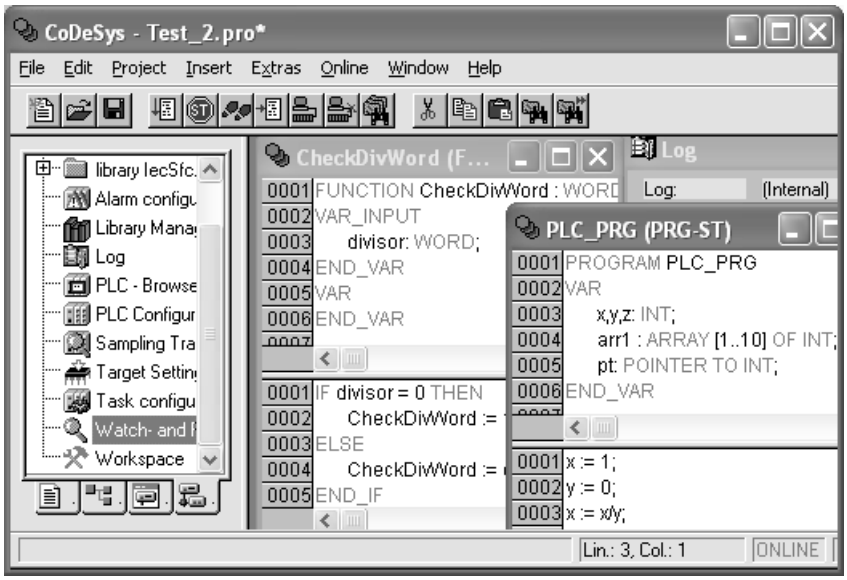

Рис. 3. Применение функции CheckDivWord.

Для контроля операций деления REAL существует аналогичная функция *CheckDivReal*.

Совершенно не обязательно включать функции контроля непосредственно в проект. Вы можете создать библиотеку таких функций и подключать ее на этапе отладки. В качестве основы удобно использовать библиотеку *Check.lib*. Она штатно устанавливается вместе с CoDeSys и включает шаблоны всех функций контроля. Вы можете исправить данную библиотеку в соответствии со своими требованиями, сохранить под другим именем и включать в проект при необходимости. Имена всех функций контроля зарезервированы. Использовать функции с аналогичными именами для иных целей нельзя.

В вычислительных выражениях могут присутствовать числовые константы. По умолчанию они приводятся к типу результата выражения. В стандарте МЭК 61131-3 предусмотрена возможность явно указывать тип констант. Например:

```
iX1: = SINT#120 – siX2;
```

Используйте типизированные константы в случаях, когда их тип может влиять на результат.

Для повышения точности целочисленных вычислений имеет смысл использовать функцию деления с симметричным округлением до ближайшего целого. Так, для 19/10 она даст результат 2, а не 1. Пример реализации такой функции описан в работе[1].

[1] *Петров И.В.* Программируемые контроллеры. Стандартные языки и приемы прикладного проектирования / Под ред. проф. В.П. Дьяконова. М.: СОЛОН-Пресс, 2004.

Ограничение диапазона значений

Разбирая типы данных CoDeSys, мы упоминали о переменных с ограниченным диапазоном значений. На базе любого целого типа мы можем сконструировать собственный тип данных. Например:

```
TYPE
  ADC: INT (-4095..4095);
END_TYPE
```

Кроме того, можно указать диапазон значений переменной непосредственно при ее объявлении:

```
In1 : INT (-4095..4095);
```

При попытке присвоить переменной с ограниченным диапазоном константу, не попадающую в заданный диапазон, CoDeSys даст сообщение об ошибке при компиляции. При необходимости контролировать переменные с ограниченным диапазоном значений в процессе исполнения применяются функции *CheckRangeSigned* (для переменных со знаком) и *CheckRangeUnsigned* (для переменных без знака). Они позволяют обрабатывать попытки присвоить переменной недопустимое значение именно так, как нужно в нашей программе. В простейшем случае мы можем ограничить присваиваемое значение или сформировать флаги ошибки.

Функции *CheckRange…* имеют 3 параметра. Это присваиваемое значение (value), нижняя (lower) и верхняя (upper) границы диапазона. Функция возвращает значение, которое реально будет присвоено. Вызов контролирующей функции производится автоматически. В простейшем случае функция *CheckRangeSigned* может быть реализована так:

```
FUNCTION CheckRangeSigned : DINT
VAR_INPUT
  value, lower, upper: DINT;
END_VAR
IF (value < lower) THEN
      CheckRangeSigned := lower;
ELSIF (value > upper) THEN
      CheckRangeSigned := upper;
ELSE
      CheckRangeSigned := value;
END_IF
```

При попытке присвоить переменной недопустимое значение данная функция обрезает его по соответствующей границе диапазона. Все функции *Check…* не отключают обработчики ошибок системы исполнения, но позволяют избежать условий их вызова.

Контроль индексов массивов

При объявлении массивов обязательно указываются диапазоны индексов. Например:

```
arr1 : ARRAY [1..10,1..10] OF INT;
```

Это необходимо для выделения соответствующего объема памяти в области данных. Если мы используем константы при обращении к элементам массива, то компилятор имеет возможность проверить допустимость заданных индексов. Но в качестве индекса может служить переменная. Если ее значение выйдет за рамки индекса массива, то мы получим обращение к области памяти, занятой другими переменными. Выявить такую ошибку непросто.

Здесь на помощь приходит функция *CheckBounds*. Она имеет 3 параметра: значение запрашиваемого индекса (index) и допустимые границы диапазона (lower, upper). Функция возвращает значение, которое реально будет использовано в качестве индекса. Вызов функции производится автоматически при обращении к любому элементу любого массива в проекте. Простейшая функция *CheckBounds* может выглядеть так:

```
FUNCTION CheckBounds : INT
VAR_INPUT
  index, lower, upper: INT;
END_VAR
```

```
IF index < lower THEN
    CheckBounds := lower;
ELSIF index > upper THEN
    CheckBounds := upper;
ELSE
    CheckBounds := index;
END_IF
```

Она ограничивает индекс массива заданными границами. Помимо отладочных целей функцию *CheckBounds* иногда используют с целью упрощения алгоритмов путем исключения многочисленных проверок индексов.

Контроль указателей

CoDeSys поддерживает работу с указателями. Указатель объявляется с помощью ключевого слова POINTER TO. Далее следует тип данных. Например:

```
Pt1: POINTER TO INT;
X : INT;
```

Указатель – это адрес переменной в памяти данных. Он может использоваться для косвенного обращения к переменным или данным экземпляров функциональных блоков. При объявлении указатель получает нулевое значение. Перед использованием ему необходимо присвоить адрес соответствующей переменной:

```
Pt1 := ADR(X);
```

Для обращения через указатель используется оператор “^” после его имени:

```
Pt1^ := 10;
```

Указатели широко применяются в параметрах функций системных и внешних библиотек. Их использование в иных целях не рекомендуется и является небезопасным. В разных аппаратных платформах указатели имеют разную разрядность. Могут отличаться порядок следования и способы выравнивания байт данных в памяти. В итоге использование указателей снижает переносимость программ. Указатели открывают доступ ко всей области данных контроллера. Ошибки в их применении могут давать непредсказуемые проявления. Отыскать истинную причину дефекта исключительно сложно. В CoDeSys имеется функция *CheckPointer*, позволяющая перехватывать обращения к указателям и анализировать их использование. Она имеет 3 параметра: значение адреса, по которому происходит обращение к данным (dwAddress), размер данных (iSize) и тип обращения (bWrite). При записи данных bWrite имеет значение TRUE. Функция возвращает значение, которое реально будет использовано в качестве адреса данных.

В следующем примере функция *CheckPointer* отслеживает попытку модификации значения глобальной переменной *iVar1Y* посредством указателей.

```
FUNCTION CheckPointer : DWORD
VAR_INPUT
dwAddress : DWORD;
iSize : INT;
bWrite: BOOL;
END_VAR

IF bWrite AND dwAddress = ADR(iVar1) THEN
    bPntErr := TRUE;
END_IF
CheckPointer := dwAddress;
```

Для контроллеров на 16-разрядных процессорах функция *CheckPointer* имеет тип WORD. Информация о физическом распределении памяти контроллера в CoDeSys представлена на вкладке Memory Layout опций целевой платформы (Target Settings). Детально исследовать содержимое памяти контроллера позволяет браузер ПЛК.

ПЛК-браузер

На вкладке ресурсов проекта CoDeSys содержится ПЛК-браузер (PLC-Browser). Фактически, это простая терминальная программа, способная передавать команды контроллеру и отображать его ответы в текстовом виде (рис. 4). Существует типовой набор команд, включающий просмотр дампа памяти, информации о типе контроллера, проекте и файловые операции. Он может быть дополнен либо урезан изготовителем контроллера.

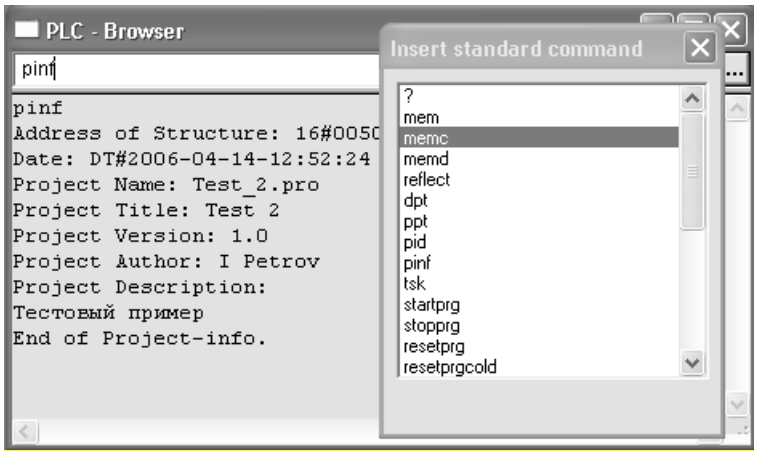

Рис. 4. ПЛК браузер

ПЛК-браузер функционирует только в режиме online. Список команд, поддерживаемых контроллером, дает команда "**?**". Ввод любой команды должен заканчиваться нажатием клавиши Enter. Чтобы получить более подробную справку о конкретной команде, нужно указать ее наименование после знака вопроса. Например: ? mem.

Команды не обязательно вводить вручную. Используйте Ассистент ввода (вызывается клавишей F2) CoDeSys. Он автоматически формирует список поддерживаемых команд и позволяет вводить их двойным щелчком мышки. В командах ПЛК-браузера допустимо использовать специальные макросы, позволяющие получить индексы программных компонентов, адреса сегментов переменных и их размер. Например, команда mem %V.PLC_PRG.x даст дамп памяти, начиная с адреса, по которому размещена локальная переменная X из программы PLC_PRG. Экраны с ответами ПЛК можно листать, распечатывать и сохранять в файле. Специальные команды для этих действий вы найдете в меню Extras CoDeSys.

Чаще всего ПЛК-браузер используется специалистами компаний, изготавливающих контроллеры для целей диагностики и тестирования аппаратных средств, а также настройки параметров (например, IP адреса). Тем не менее, это открытый инструмент, доступный пользователям CoDeSys.

Бортжурнал

Представьте себе следующую ситуацию. Наш нормально работавший контроллер неожиданно перешел в режим стоп или стал функционировать с ошибками. Все симптомы указывают на то, что программа была кем-то повреждена, либо были случайно изменены или зафиксированы значения переменных. Как это проверить? Это одна из типовых проблем, которую позволяет решить бортжурнал CoDeSys. Он представляет собой детальный протокол работы в online сессиях. Чтобы включить работу бортжурнала, откройте вкладку Log опций проекта (Project, Options). Здесь необходимо установить флажок Activate logging и настроить фильтр записываемых событий. Мы можем фиксировать сервисы, запрашиваемые пользователем CoDeSys (User actions), изменение состояния контроллера (Status changes), внутренние действия (Internal actions) и ошибки исполнения (Exceptions). Все события хронометрируются по времени и длительности.

Окно бортжурнала открывается выбором объекта Log в разделе ресурсов менеджера проекта (рис. 5) или командой Window, Log. Протокол автоматически разбивается на online сессии. По умолчанию отображается текущий (internal) бортжурнал. Все или выборочные сессии можно сохранить в двоичный (*.log) файл для последующего анализа.

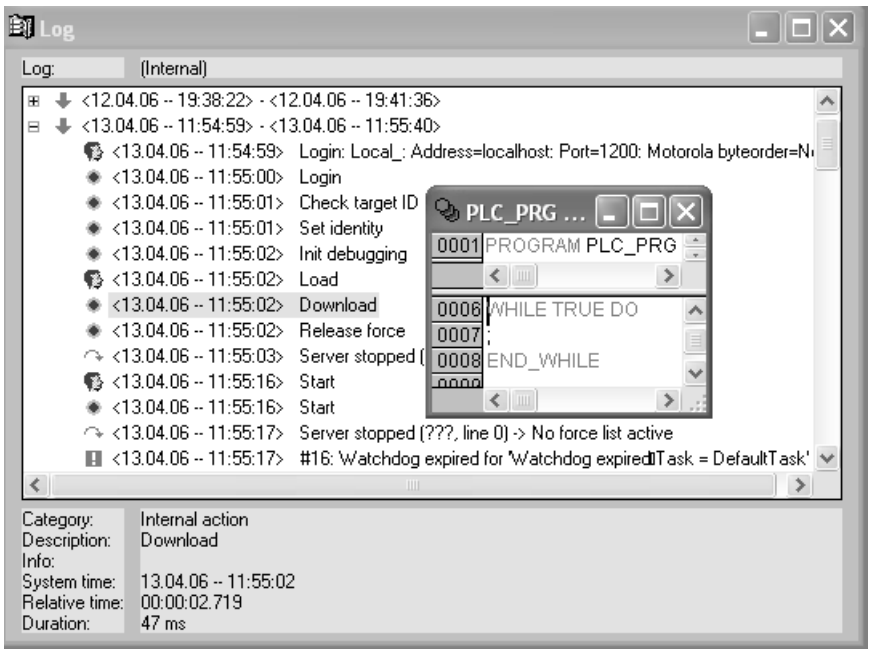

Рис. 5. Окно бортжурнала.

Диагностика канала связи

Связь контроллера с системой программирования CoDeSys или OPC-сервером всегда происходит через промежуточный Gateway Server. Он может быть запущен на любой машине в сети. По умолчанию Gateway автоматически запускается при старте CoDeSys. Физически связь с контроллером происходит через RS-232, RS-485, CAN, Ethernet или оригинальный интерфейс изготовителя контроллера.

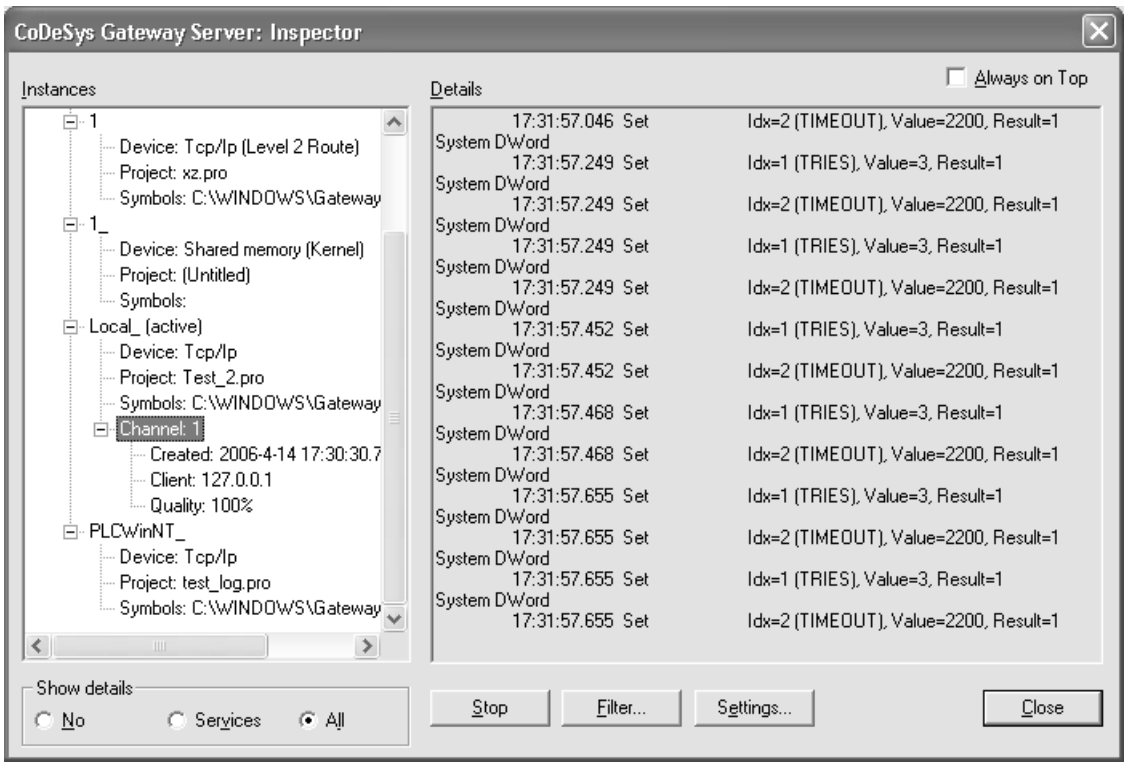

Рис. 6. Протокол работы Gateway Server.

В любом случае, CoDeSys передает контроллеру запросы на выполнение определенных сервисов (подключение, загрузку, чтение значений переменных и т.д.) и принимает ответные сообщения. Открыв в Gateway Server активное соединение, мы можем включить протоколирование работы канала связи (Settings, Logging) в файл или просмотр

текущих данных в окне Gateway (Settings, Tracing). Эти возможности используются в диагностических целях для оценки качества линии связи. В свойствах активного соединения присутствует параметр “качество связи”. Если канал связи работает устойчиво, то есть на каждый запрос приходит соответствующий ответ, то качество связи будет иметь значение 100 % (рис. 6).

Когда все усилия тщетны

Предположим самый худший случай. Мы исчерпали все идеи и не смогли разобраться с насущной проблемой. Возникло подозрение на системную ошибку, исправить которую может только изготовитель контроллера. Короче говоря, необходимо обращаться за помощью. Естественно, заявление типа “не работают таймеры” не будет адекватно воспринято службой технической поддержки. CoDeSys – широко распространенный надежный комплекс. Подобные дефекты в нем если и были, то уже устранены много лет назад. Нам необходимо абсолютно точно описать свой проект и условия возникновения проблемы. Как это сделать? На такой случай в CoDeSys есть специальный инструмент. Дайте команду “File”, “Save/Mail Archive”. Данная команда открывает диалоговое окно, в котором перечислены все детали, касающиеся проекта: версии компонентов CoDeSys, файл проекта, все библиотеки, символьные файлы, записи из системного реестра, конфигурационные файлы и файлы настройки целевой системы. Здесь же мы можем добавить текстовое описание. В итоге получается zip-архив, позволяющий специалистам технической поддержки проанализировать нашу проблему. Кроме того, данный инструмент позволяет найти отличия, если вдруг обнаруживается, что проект, нормально компилирующийся на одном персональном компьютере, вызывает ошибки на другом.

Заключение

Все вышерассмотренные инструменты и приемы используются на практике нечасто. Часть из них предназначена для системных диагностических целей изготовителей контроллеров. Сверх того, в CoDeSys существует возможность генерации ассемблерных листингов компилятора. Надеемся, что вам никогда не придется заниматься этим самостоятельно. Но знание этих возможностей дает уверенность, особенно в условиях, когда разработчики расположены отнюдь не в соседнем офисе.

Постепенно мы погрузились достаточно глубоко в тонкости CoDeSys, однако незаслуженно пропустили немало более значительных вопросов. В их числе оказался один из самых интересных языков программирования МЭК 61131-3.

Продолжение следует.

 © НАУЧТЕХИЗДАТ 2006г.