УТВЕРЖДЕН

643.ИАИА.00001-01 81 01-ЛУ

ШАБЛОН ПРОГРАММЫ-СЕРВЕРА ДЛЯ *QNX6*

Пояснительная записка

643.ИАИА.00001-01 81 01

Листов 12

2018

Литера

Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ

Наименование программы «Шаблон программы-сервера для *QNX6*».

Обозначение 643.ИАИА.00001-01.

Разработка осуществлялась по личной инициативе.

2. НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ

Шаблон предназначен для разработчиков программного обеспечения для *QNX6*. Он является основой для написания компонентов программных комплексов. Самостоятельного применения не имеет.

3. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ

3.1. Постановка задачи

Практически все компоненты программных комплексов для *QNX6* построены по серверной схеме, то есть имеют однотипную архитектуру [1].

По техническому заданию требуется выделить фиксированную часть, которая одинакова для всех модулей и определяет работу программы как сервера. Переменную часть, в которой реализованы специфические для прикладных программ действия, оформить в виде шаблона, включив в нее обработку системных сообщений. В шаблон заложить механизмы расширения функциональности путем добавления обработки сообщений, а также путем наследования в новом модуле обработки сообщений в созданном ранее модуле.

3.2. Выделение фиксированной части

Рассмотрим псевдокод программы-сервера.

```
Initialization();
while (1) {
     rcvid = Receive();
     if (rcvid != 0) {
          ProcessMsg();
          Reply();
     } else
          ProcessPulse();
}
Completing();
```

После инициализации выполняется цикл приема и обработки сообщений и импульсов.

Рассмотрим операции перед циклом подробнее. Сначала выполняется инициализация программы. После этого осуществляется подготовка к работе, включающая в себя, в частности, создание канала приема сообщений. И, наконец, в

ряде случаев необходимо выполнить действия, зависящие от созданного канала. К таким случаям относится создание событий, например, для таймеров.

Аналогично, при завершении работы необходимо выполнить действия как для прикладной программы, так и для серверной части.

С учетом подробного рассмотрения получим следующий псевдокод.

```
Initialization();
Prepare();
ChannelDepended();
while (1) {
     rcvid = Receive();
     if (rcvid != 0) {
          ProcessMsg();
          Reply();
     } else
          ProcessPulse();
}
FixCompleting();
Completing();
```

Жирным шрифтом выделены пять функций, в которых заключены операции, специфические для прикладных программ.

Таким образом, все действия, обеспечивающие работу программы как сервера, можно выделить в отдельный фиксированный файл, который не будет изменяться. Переменная часть будет содержаться в пяти функциях, размещенных в другом файле.

В целях большей гибкости эти функции лучше выполнить в виде внешних вызовов. Для этого в фиксированной части создать пять указателей на функции, а в переменной части присваивать этим указателям значения реальных функций. На языке C++ решение заключается в абстрактных методах и наследовании класса.

3.3. Шаблон программы

Одного лишь выделения фиксированной части недостаточно для создания шаблона. Для получения понятной, хорошо читаемой программы остальной текст надлежит структурировать и разбить на файлы.

Функции, вызываемые из фиксированной части, размещены в отдельном файле. Обработчики сообщений и импульсов (*ProcessMsg()* и *ProcessPulse()*) имеют следующую структуру.

```
switch (type) {
case t1: F1();
     break;
case t2: F2();
     break;
…
case tn: Fn();
     break;
}
```

В них по типу команды или по коду импульса вызываются соответствующие фактические обработчики, которые и выполняют все необходимые действия.

Именно в фактических обработчиках заложена суть работы модуля, поэтому их следует также выделить в отдельный файл.

Описания типов и структур данных, определяющих сообщения, обрабатываемые программой, размещаются в *h*-файле.

Для полноты охвата возможностей добавим еще два файла.

Некоторые программы могут получать параметры из строки запуска. Обработка таких параметров может быть достаточно объемной, что вполне является критерием выделения. В это же файл логично поместить текст, выдаваемый по команде *use*.

Для обращения к модулю необходимо заполнить поля соответствующих структур и вызвать функцию *MsgSend()*. В целях упрощения работы с модулем рекомендуется для каждой команды создать функцию-обертку. Эти функции не

входят в состав программы, но образуют библиотеку для работы с ним. Их следует их разместить в отдельном файле.

Дадим вышеперечисленным файлам имя, совпадающее с названием программы, для отличия добавим суффиксы.

Таким образом, получаем, что кроме выделенной фиксированной части, реализующей работу программы как сервера, шаблон программы состоит из пяти файлов:

- внешние вызовы;
- фактические обработчики;
- типы и структуры данных, описывающих сообщения;
- обработка параметров строки запуска;
- библиотека для работы с модулем.

В реальной программе могут быть добавлены дополнительные файлы, содержащие, например, описания типов или внутренние классы.

3.4. Системные и общие вызовы

В ряде ситуаций программа получает сообщения и импульсы от самой операционной системы. Так как шаблон предполагается сделать полным и всеобъемлющим, то необходимо добавить соответствующие обработчики.

Кроме того, есть ряд команд, которые могут использоваться во всех программах или достаточно часто. Например, установление связи с программой, завершение ее работы. Такие команды также надо включить в шаблон. В дальнейшем список может расширяться.

3.5. Расширение функциональности

Для получения прикладной программы необходимо иметь возможность расширять функциональность шаблона, то есть добавлять обработку новых сообщений.

Это можно сделать двумя методами.

Во-первых, простое добавление. Для этого надо задать тип сообщения, описать структуры принимаемого и ответного сообщений, если они есть. В функцию

`ProcessMsg()` добавить вызов обработчика по типу. Написать собственно обработчик, и, наконец, добавить функцию в библиотеку работы с модулем.

Аналогичные действия выполняются для импульсов. Для обработки событий, например от таймеров, надо дополнительно инициализировать событие в функции `ChannelDepended()`.

Второй метод – наследование.

Скопируем весь шаблон, исключив фиксированную часть и реализованные обработчики. Переименуем файлы и некоторые идентификаторы. При этом внешние вызовы будут настроены на новые функции.

В новых обработчиках сообщений и импульсов в варианте *default* в операторах *switch* вызовем обработчики из созданного ранее шаблона.

Таким образом, расширять функциональность нового модуля можно методом добавления, а для реализованных ранее сообщений вызываются обработчики родительского модуля.

Механизм наследования может быть применен на нескольких уровнях.

3.6. Разнесение диапазонов значений

При наследовании значения обрабатываемых типов команд и кодов импульсов родительского и дочернего модулей, как правило, не должны пересекаться. Непрерывность значений обеспечить трудно, но в этом и нет необходимости.

Предложено решение о выделении каждому уровню наследования диапазона возможных значений типов команд и кодов импульсов.

При реализации задается константа для базового значения и шаг для каждого уровня наследования. Шаг заведомо больше, чем количество используемых значений в реальных программах.

Типы команд и коды импульсов задаются в перечислениях. Первый элемент перечисления должен иметь значение

```
base + step * N,
```

где *base* и *step* – базовая константа и шаг, соответственно, а *N* – уровень наследования. Нумерация уровней начинается с нуля.

4. ОЖИДАЕМЫЕ ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

Использование шаблона упростит и ускорит написание компонентов, так как в нем полностью обеспечивается функционирование программы как сервера, и разработчику остается реализовать только прикладные аспекты модуля.

Надежность программ повысится ввиду того, что в шаблоне учтены все вопросы организации сервера, а при разработке модуля с нуля, некоторые моменты могут быть упущены.

Применение единого шаблона улучшит понимание и сопровождение программных комплексов, особенно при совместной деятельности нескольких разработчиков.

5. ИСТОЧНИКИ, ИСПОЛЬЗОВАННЫЕ ПРИ РАЗРАБОТКЕ

1. Кртен Р. Введение в QNX Neutrino. Руководство для разработчиков приложений реального времени. – 2-е изд. – СПб.: БХВ-Петербург, 2011. – 368 с.

Лист регистрации изменений									
Номера листов (страниц)					Всего листов (страниц) в докум.	№ документа	Входящий № сопроводительного докум. и дата	Подп.	Дата
Изм.	измененных	замененных	новых	аннулированных					

Изм. *Подпись* *Дата*