УТВЕРЖДЕН

643.ИАИА.00001-01 13 01-ЛУ

ШАБЛОН ПРОГРАММЫ-СЕРВЕРА ДЛЯ *QNX6*

Описание программы

643.ИАИА.00001-01 13 01

Листов 20

2018

Литера

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

АННОТАЦИЯ

В документе подробно описаны все файлы, входящие в состав шаблон программы-сервера. Адресован для специалистов, желающих изучить внутреннее строение шаблона.

СОДЕРЖАНИЕ

1. ОБЩИЕ СВЕДЕНИЯ

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

Обозначение 643.ИАИА.00001-01.

2. ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ

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

3. ОПИСАНИЕ ЛОГИЧЕСКОЙ СТРУКТУРЫ

В документе описан шаблон на языке Си. Версия на C++ имеет аналогичную структуру.

Описание ведется в терминах файлов. В данном контексте понятие файл по смыслу ближе к классу, то есть представляет собой совокупность переменных и функций.

3.1. Соглашение об именовании

В программных комплексах один компонент может обращаться к нескольким компонентам. Кроме того, возможно расширение функциональности путем наследования. Во избежание коллизий идентификаторов необходимо выработать принципы именования.

К файлам применяются суффиксы, к идентификаторам – префиксы.

Для различения имен разных модулей каждому модулю присваивается уникальный двухсимвольный префикс. Для различения имен в рамках одной программы добавляется однобуквенный префикс.

Далее в таблицах приведены правила именования. В примерах предполагается, что программа имеет имя *program*, уникальный префикс *pr*.

Формат имен файлов: <имя_программы>_<суффикс>. Суффиксы описаны в таблице 1. Дополнительные, внутренние для программы файлы могут иметь произвольные имена.

Таблица 1 – Правила именования файлов

Суффикс	Пример	Содержание файла
отсутствует	*program*	Фактические обработчики
ser	*program_ser*	Внешние вызовы
opt	*program_opt*	Обработка параметров строки запуска программы
lib	*program_lib*	Библиотека для обращения к программе
srr	*program_srr*	Описание сообщений для обмена

Имена библиотечных функций и фактических обработчиков совпадают с именами команд. К именам библиотечных функций и внешним вызовам добавляется префикс программы (см. таблицу 2).

Таблица 2 – Правила именования функций

Префикс	Пример	Назначение функции
отсутствует	*Start*	Фактический обработчик команды *start*
pr	*pr_Start*	Библиотечная функция для выдачи команды *start*
pr	*pr_Initialization*	Внешний вызов

Имена в описаниях сообщений для обмена составляются из уникального префикса программы и однобуквенного префикса типа описания. Имена типов команд, кодов импульсов, принимаемых и ответных сообщений составляются на базе имени команды (см. таблицу 3).

Таблица 3 – Правила описаний сообщений

Префиксы	Пример	Описываемый тип
pr	*pr_types_t*	Тип перечисления типов команд
pr	*pr_codes_t*	Тип перечисления кодов импульсов
pr_t	*pr_t_start*	Тип команды *start*
pr_c	*pr_c_start*	Код импульса *start*
pr_r	*pr_r_start_t*	Описание принимаемого сообщения команды *start*
pr_p	*pr_p_start_t*	Описание ответного сообщения команды *start*
pr_r	*pr_r_msg_t*	Объединение принимаемых сообщений *program*
pr_p	*pr_p_msg_t*	Объединение ответных сообщений *program*

В функциях библиотеки для работы с программой в функции *MsgSend()* используются переменные *s_msg* для передаваемого сообщения и *l_msg* для принимаемого в ответ сообщения. Они имеют тип принимаемого и ответного сообщения для подаваемой команды.

3.2. Файл *server_fix*

Фиксированная (неизменяемая) часть программы-сервера (рис. 1), которая реализует работу программы как сервера.

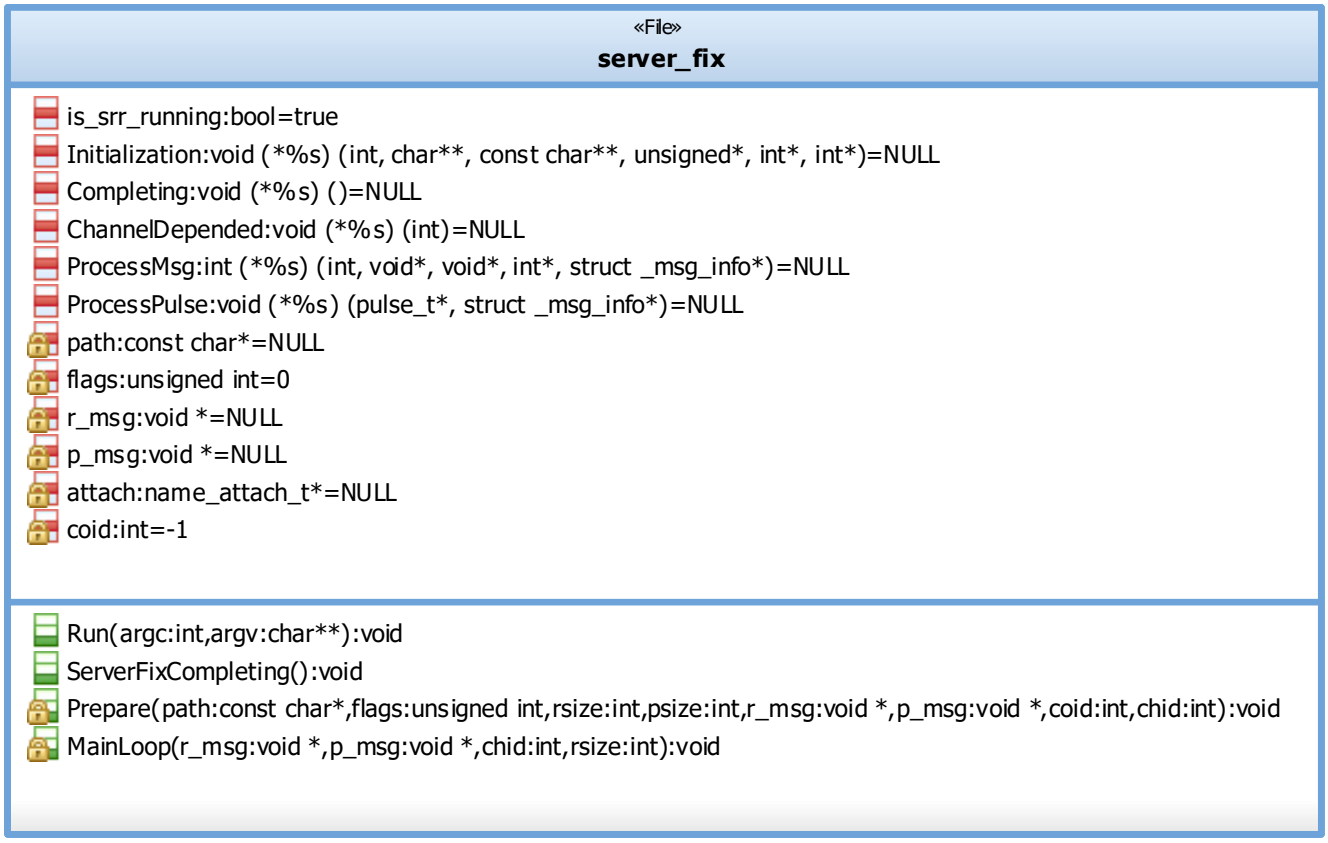

Рисунок 1– Структура файла *server_fix*

Функция *Run()* единственная, которая вызывается из *main()*. В ней сначала выполняются подготовительные действия, затем вызывается *MainLoop()*, содержащая цикл приема и обработки сообщений.

Пять указателей на функции являются внешними вызовами (*callback*). Их значения должны быть определены в программе.

При инициализации последовательно вызываются *Initialization()*, *Prepare()*, *ChannelDepended()*.

Prepare() проверяет, что внешние вызовы получили значения, выделяет память для сообщений и создает канал вместе с регистрацией заданного имени.

MainLoop() содержит цикл приема и обработки сообщений. Выход из цикла определяет *is_srr_running*. После приема сообщения выполняется внешний вызов *ProcessMsg()* или *ProcessPulse()* для сообщений и импульсов соответственно.

Если *ProcessMsg()* возвращает значение, не равное (-1), то клиенту передается ответное сообщение. Такая проверка позволяет реализовать отложенный ответ.

Завершающие действия фиксированной части *ServerFixCompleting()* вызываются из внешнего вызова *Completing()*, который выполняется при завершении работы программы (задается в *main()*).

3.3. Файл *server_ser*

Реализует пять внешних вызовов, используемых в фиксированной части (рис. 2). В соответствии с соглашением об именовании они имеют префикс *se*.

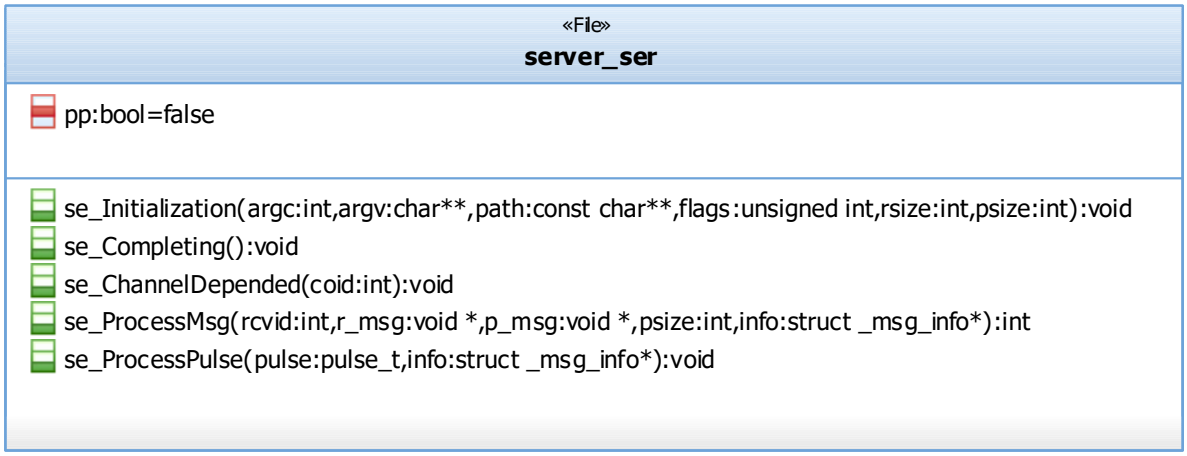

Рисунок 2 – Структура файла *server_ser*

se_Initialization() получает параметры строки запуска программы и должна задать значения пути регистрации программы, флаг типа регистрации (локальный или глобальный) и максимальные размеры входных и ответных сообщений.

se_Completing() вызывает завершающие действия обработчика сообщений и фиксированной части.

В *se_ChannelDepended()* выполняются операции, зависящие от созданного канала. К таковым относится создание событий.

se_ProcessMsg() – обработчик сообщений. Для того, чтобы был послан ответ клиенту, надо установить возвращаемое значение отличное от (-1), задать само сообщение и его длину, возможно нулевую.

se_ProcessPulse() – обработчик импульсов.

В обработчиках вставлена обработка системных сообщений и импульсов по образцу примера из описания функции *name_attach()* в документации на *QNX*.

3.4. Файл *server*

Содержит фактические обработчики сообщений (рис. 3).

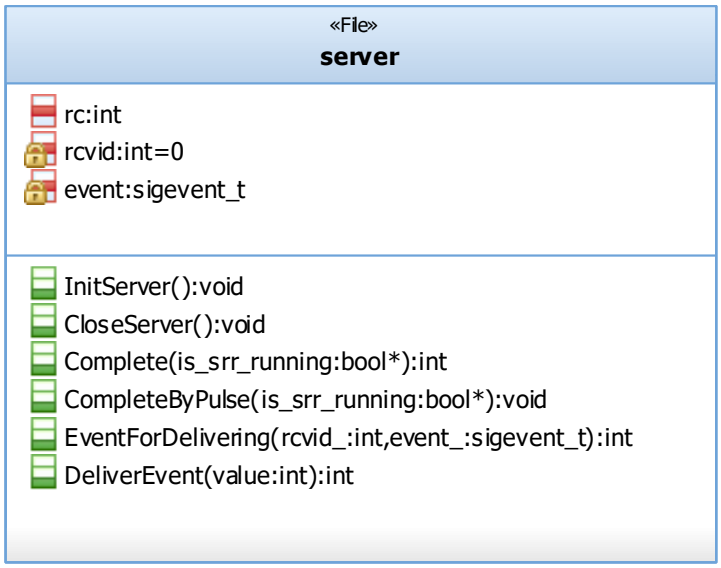

Рисунок 3 – Структура файла *server*

В шаблоне обрабатываются команды, которые считаются общеупотребительными. В настоящей версии к ним отнесены три команды.

Complete() и *CompleteByPulse()* завершают работу программы при поступлении сообщения или импульса.

EventForDelivering() запоминает отправителя и событие для последующего возвращения.

Функция *DeliverEvent()* возвращает запомненное событие. Вызывается не по команде, а из самой программы.

Текущая реализация предполагает только одно запоминаемое событие. Если в прикладной программе таковых должно быть больше, то их надо создавать отдельно.

При расширении функциональности путем наследования настоящий файл подключается к дочерней программе. Видимость переменной *rc* установлена открытой, чтобы не переопределять ее в дальнейшем.

3.5. Файл *server_srr*

Содержит описания типов и структур сообщений (рис. 4).

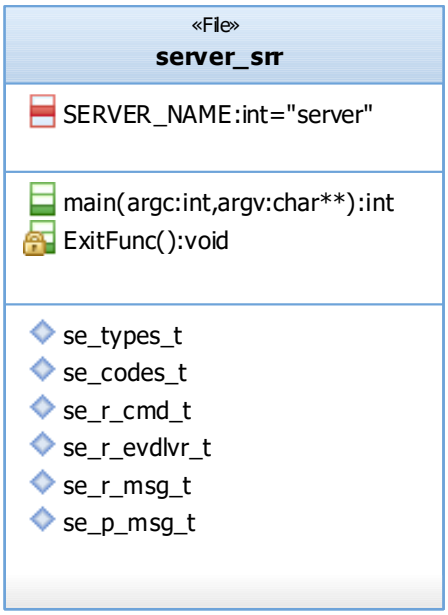

Рисунок 4 – Структура файла *server_srr*

В файле описаны типы команд (*se_types_t*), коды импульсов (*se_codes_t*), обрабатываемые в шаблоне, формат принимаемых структур (*se_r_*_t*) (ответные отсутствуют), а также объединения принимаемых (*se_r_msg_t*) и ответных (*se_p_msg_t*) структур. Все идентификаторы составлены в соответствии с соглашением об именовании.

Тип *se_r_cmd_t* описывает сообщение без данных, содержащее только тип команды. Этот тип используется и в функциях-обертках наследующих программ.

Кроме того, задано имя (или его базовая часть), с которым будет связан создаваемый канал.

Функция *main()* должна где-то присутствовать в программе. Решено разместить ее в данном файле. В ней получают значение указатели на внешние вызовы, устанавливается вызов функции завершения при прекращении работы и, наконец, вызывается *Run()* из фиксированной части, которая и запускает программу как сервер.

3.6. Файл *server_lib*

Библиотека функций, являющихся обертками над посылкой сообщений (рис. 5).

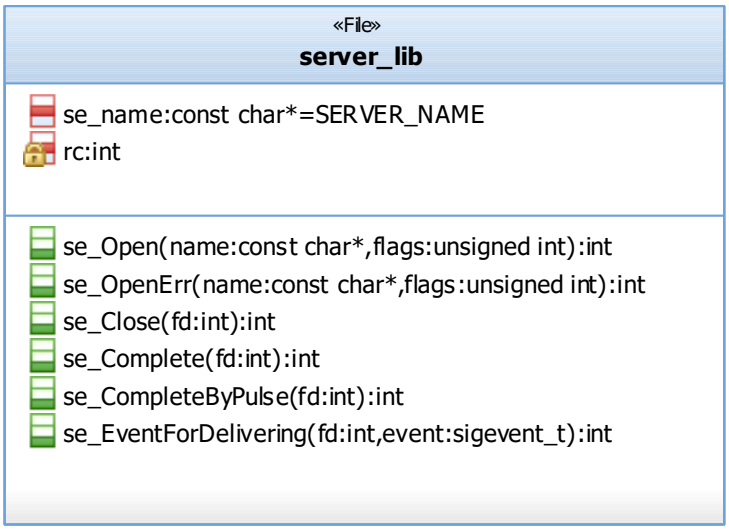

Рисунок 5 – Структура файла *server_lib*

Для каждого обрабатываемого сообщения создается своя функция.

Функции *se_Open()* и *se_Close()* не посылают сообщения. Они устанавливают и прекращают связь с каналом. Из-за того, что они всегда используются при работе с программой, решено разместить их в данном файле.

se_OpenErr() является специальным случаем. Она делает попытки установить связь в течение 5 с, при неудаче выдает сообщение и прекращает работу программы. Функция полезна в случаях, когда зависимый компонент в комплексе не смог начать работу по каким-либо причинам. Считается, что 5 с достаточно, чтобы стартовали все модули в комплексе.

3.7. Файл *server_opt*

Заготовка для обработки параметров строки запуска программы (рис. 6).

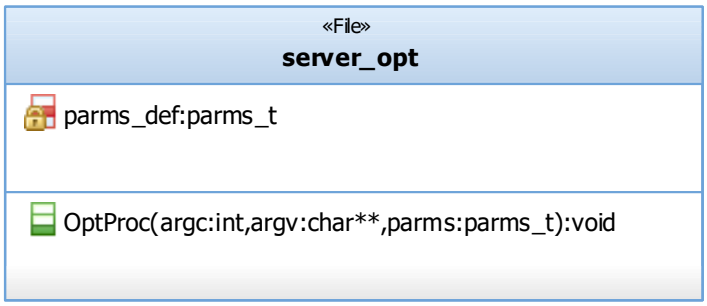

Рисунок 6 – Структура файла *server_opt*

В настоящей версии шаблона данный файл не используется. Он открыт для заполнения в прикладной программе.

3.8. Файл *server_types*

Содержит описания типов, полезных в прикладных программах (рис. 7).

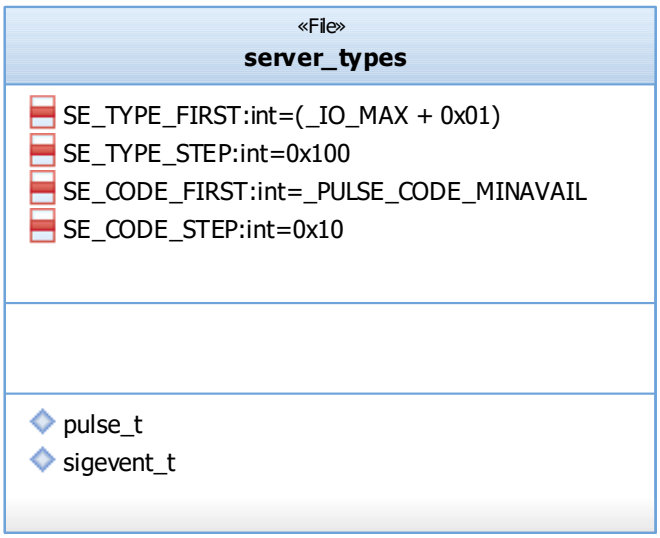

Рисунок 7 – Структура файла *server_types*

Определены константы, задающие диапазоны значений типов команд и кодов импульсов при расширении путем наследования.

Для удобства структуры, описывающие импульсы и события, переопределены в более привычный вид с отказом от слова *struct* и с добавлением суффикса *_t*.

3.9. Файл *tools*

Вспомогательные функции (рис. 8).

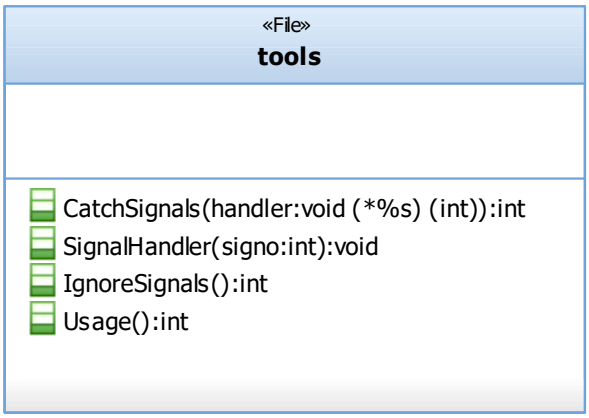

Рисунок 8 – Структура файла *tools*

CatchSignals() подставляет в качестве обработчика всех сигналов *SignalHandler()*, в котором печатается сообщение о номере сигнала.

IgnoreSignals() игнорирует реакцию на все сигналы, которые это допускают.

Usage() выводит сообщение об использовании программы, выдаваемое по команде *use*.

3.10. Шаблон *program*

Описанные выше файлы, находятся в директории *server*, и их вполне можно применять в качестве шаблона при создании прикладных программ. Но при этом в состав каждого модуля текстуально будет входить фиксированная часть и уже имеющиеся обработчики.

Задачей данного проекта является создание шаблона, в котором не было бы ничего лишнего, а все необходимое подключалось бы в виде библиотек.

Для этого создан шаблон *program* путем наследования функциональности *server*.

В директорию *program* скопированы файлы *server*, кроме *server_fix, server_types, tools*. Файлы и некоторые идентификаторы переименованы в соответствии с соглашением об именовании. Удалена обработка системных и общих сообщений. В обработчики *program* добавлены соответствующие вызовы обработчиков *server*.

Шаблон *program* настоятельно рекомендуется использовать в качестве основы при разработке прикладных программ.

Взаимосвязи между файлами *program** и *server** показаны на рис. 9.

3.11. Последовательность действий

На рис. 10-12 показано взаимодействие составляющих шаблона *program* с составляющими серверной части *server* при инициализации, завершении работы и обработке сообщений.

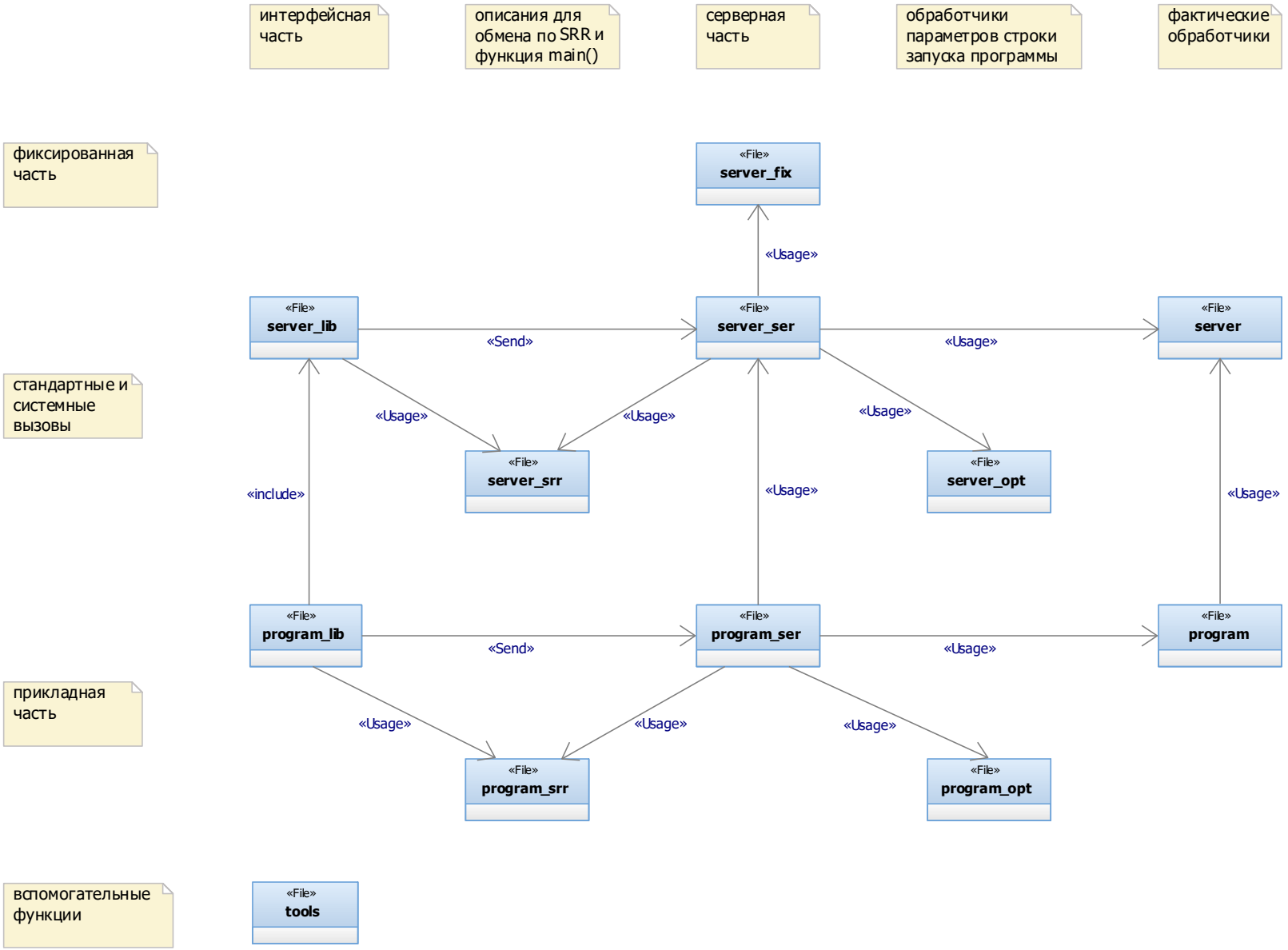

Рисунок 9 – Диаграмма файлов шаблона программы-сервера

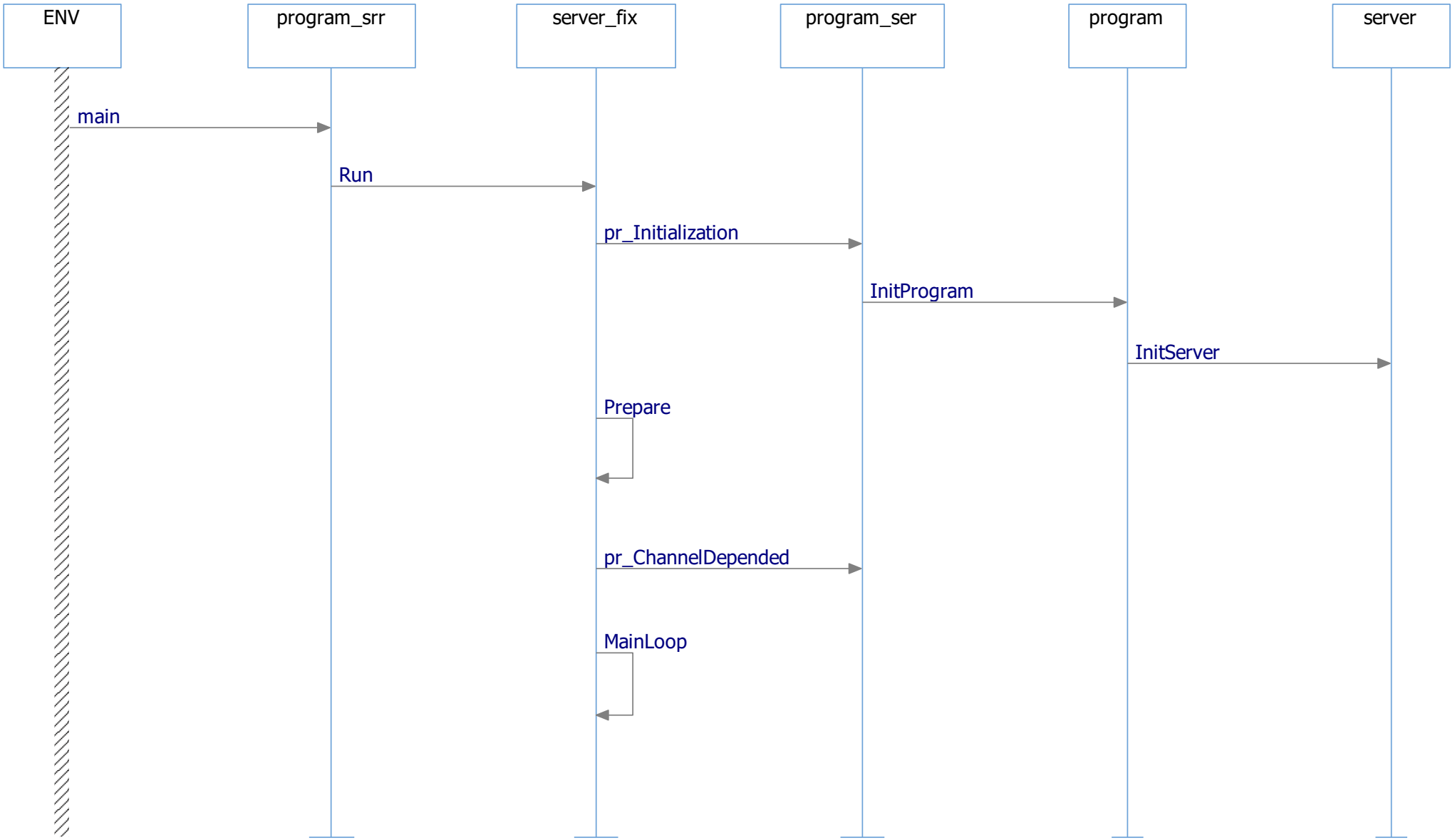

Рисунок 10 – Действия при инициализации

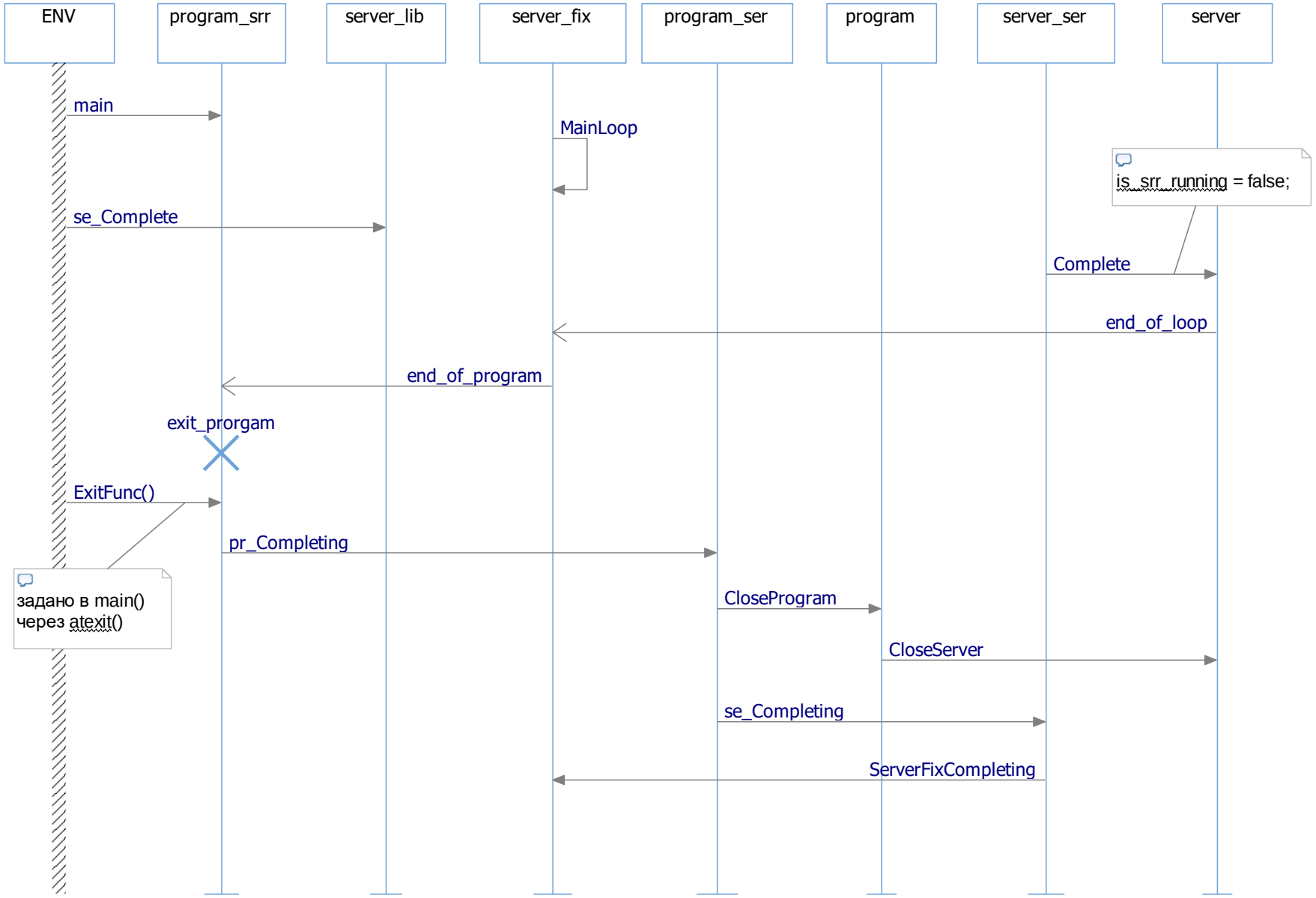

Рисунок 11 – Действия при завершении работы

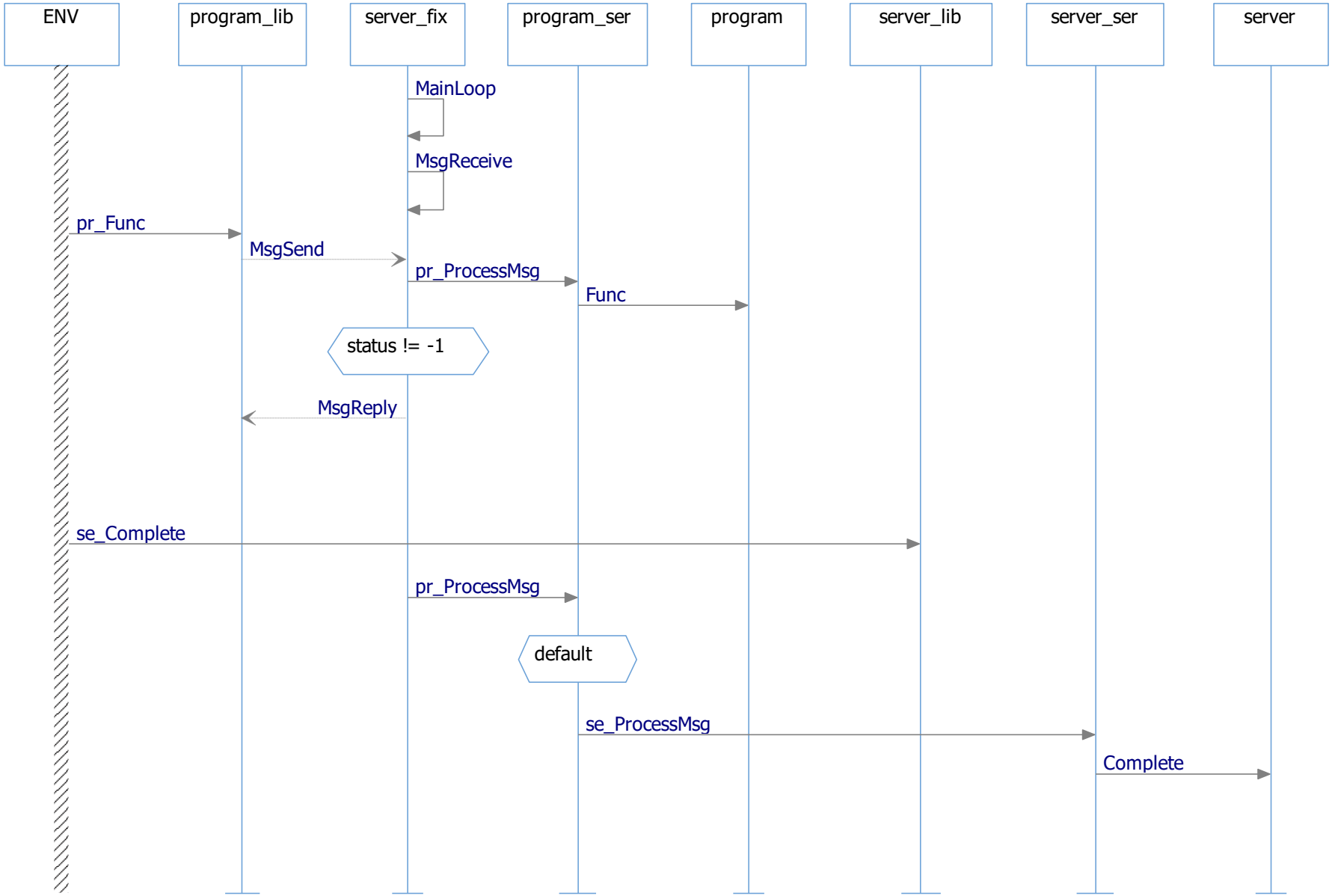

Рисунок 12 – Действия при обработке команды

4. ИСПОЛЬЗУЕМЫЕ ТЕХНИЧЕСКИЕ СРЕДСТВА, ВЫЗОВ И ЗАГРУЗКА, ВХОДНЫЕ И ВЫХОДНЫЕ ДАННЫЕ

Шаблон является основой для создания прикладных программ для *QNX6*. Самостоятельно применяться не может, следовательно, технические средства не используются, вызов и загрузка, входные и выходные данные отсутствуют.

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*