Реализация принципа контролируемого выполнения прикладной программы в среде операционной системы реального времени

А.И. Грюнталь[1], К.Г. Нархов, А.М. Щегольков

1 - кандидат физико-математических наук

Аннотация: В статье рассматривается ряд вопросов, связанных с реализацией средств и механизмов контролируемого выполнения прикладной программы в среде операционной системы реального времени.

Ключевые слова: контролируемое выполнение, многопоточная программа, поток управления, очередь сообщений, сигнал, операционная система реального времени, команда управления.

Введение

Принцип контролируемого выполнения представляет собой общий концептуальный поход к проектированию и разработке программ. Принцип контролируемого выполнения введён в [1, 2]. В статье рассматривается реализация принципа контролируемого выполнения для многопоточных программ.

Программа с контролируемым выполнением – это программа со встроенными механизмами, обеспечивающими выполнение программы, вообще говоря, в условиях поступающих в программу некорректных внешних данных, ошибок в среде исполнения, ошибок в самой прикладной программе.

В настоящей статье рассматривается одна из реализаций принципа контролируемого выполнения для прикладных программ, имеющая структурные, архитектурные и низкоуровневые реализационные особенности. Говоря ниже о программе с контролируемым выполнением, подразумевается одна из структурных, архитектурных и низкоуровневых реализационных возможностей.

Основу средств контролируемого выполнения прикладной программы составляют механизмы, обеспечивающие контроль состояния выполняемой программы, сравнение параметров состояния прикладной программы с эталоном, и, в случае отклонения поведения программы от эталона, генерацию управляющих воздействий, направленных на устранение выявленных отклонений.

Реализация изложенного подхода требует наличия в прикладной программе **агентов**, регистрирующих и собирающих данные о выполнении прикладной программы (**контрольные параметры состояния**), **монитора**, анализирующего в реальном масштабе времени получаемые от агентов контрольные параметры состояния и генерирующего управляющие воздействия (**команды управления приложением**), и средств выполнения сгенерированных монитором команд управления приложением.

Кроме того, в прикладной программе должны присутствовать средства передачи контрольных параметров состояния от агентов к монитору и средства, передающие команды управления приложением. Наконец, монитор должен содержать эталон, то есть **модель корректного выполнения прикладной программы**, содержащую прогнозируемые (эталонные) значения контрольных параметров.

В общем виде прикладная программа с реализованными средствами контролируемого выполнения включает следующие **структурные компоненты**:

- собственно прикладную программу (приложение), реализующую миссию прикладной программы (основную цель выполнения программы);
- внедрённые в приложение агенты;
- монитор;
- средства реализации управляющих воздействия;
- средства передачи **контрольных данных состояния** от агентов к монитору;
- средства передачи **команд управления приложением** от монитора приложению;
- модель корректного выполнения приложения.

Прикладную программу, содержащую перечисленные структурные компоненты, будем далее называть программой с контролируемым исполнением. Заметим также, что для ясности формулировок понятие приложения и прикладной программы различаются. Под приложением понимается та часть прикладной программы, которая непосредственно отвечает за прикладную функциональность. Прикладная программа в целом содержит приложение и части, обеспечивающие организацию вычисления и контролируемое выполнение. Концептуальная структура прикладной программы с контролируемым выполнением представлена на **рисунке 1**.

1 Архитектура программы с контролируемым выполнением в среде операционной системе реального времени

Приведём конкретизацию представленной выше структуры программы с контролируемым выполнением, относящуюся к программам реального времени. Существенной чертой программ реального времени является многопоточность, обеспечивающая быстрое переключение между различными прикладными функциями в зависимости от поступающих внешних воздействий [3].

Многопоточность предполагает использование в тексте программы объектов операционной системы, связанных с организацией взаимодействия потоков управления, передачей данных и синхронизацией. Применение многопоточности вытекает из следующих характеристик программ реального времени:

- *Быстрая и своевременная реакция на прерывания*. Важной задачей управления является принятие управляющих решений и генерация управляющих сигналов в связи с изменением внешних воздействий – прерываний от внешних устройств, индицирующих определенные характеристики окружения. При этом задержка в реакции на такие прерывания может носить катастрофический характер.
- *Наличие интеллектуальной части, в которой выполняется обработка прерываний*. Обработка прерываний представляет собой логически отдельную задачу, которую также целесообразно (с точки зрения простоты архитектуры программы) выполнять в контексте отдельных потоков управления (*групп функциональных потоков управления*).

Операционные системы реального времени спроектированы так, чтобы минимизировать (при заданных аппаратных ресурсах) длительность интервала времени, затрачиваемого на переключение между различными потоками управления.

С учётом сказанного, прикладные программы реального времени целесообразно проектировать так, чтобы различные прикладные функции были локализованы в различных потоках управления. Тогда время переключения между различными прикладными функциями при изменении внешних условий может выполняться средствами операционной системы реального времени и, тем самым, будет минимальным.

Таким образом, приложение мы рассматриваем как многопоточную программу, каждый поток управления которой выполняет определённую прикладную функцию, обусловленную миссией приложения.

Функционирование приложения представляет собой псевдопараллельное выполнение потоков управления. Последовательность выполнения потоков управления определяется поступающими внешними данными, а при их отсутствии или в случае, когда реакция на внешние данные не должна быть неотложной, определяется выбранной для приложения дисциплиной планирования [4, 5, 6, 7].

Предложенная схема, согласно которой имеется взаимно однозначное соответствие между потоками управления и прикладными функциями, несколько идеализирована, поскольку наряду с потоками управления, выполняющими чисто прикладные функции, программа включает потоки управления, выполняющие вспомогательные функции, обеспечивающие организацию вычислительного процесса в целом. Например, программа может содержать потоки управления, создающие необходимые для функционирования прикладной программы программные ресурсы (очереди сообщений, таймеры, сигналы и др.) Кроме того, некоторые прикладные функции могут быть локализованы не в одном, а в нескольких потоках управления. Например, функциональность, обеспечивающая обмен данными с относительно медленными устройствами, может быть локализована в отдельных потоках управления, которые должны обладать меньшим приоритетом и быть изолированы в группы таким образом, чтобы их влияние на интеллектуальную часть программы было минимальным. Примером может служить:

- *Использование асинхронного ввода-вывода*. Программы реального времени характеризуются необходимостью ввода-вывода данных с использованием специфических внешних интерфейсов. Программирование конкретных особенностей ввода-вывода может выполняться в специализированных потоках управления, что снижает логическую сложность программы.
- *Реализация диалога с оператором*. При необходимости программа реального времени должна работать в диалоговом режиме. По сути это частный случай асинхронного ввода-вывода.
- *Использование прикладных библиотек*. Целесообразно по возможности изолировать алгоритмы, использующие в своем теле функции (API) библиотек, в специализированные потоки управления. Это позволит снизить логическую сложность программы и увеличить ее общую масштабируемость.

В связи со сказанным, будем рассматривать более общую ситуацию: будем считать, что каждая прикладная функция может быть, вообще говоря, локализована не в одном, а в нескольких потоках управления. Совокупность потоков управления, реализующих одну прикладную функцию, будем называть группой функциональных потоков управления (ГФП). При этом реакция приложения на конкретное внешнее событие состоит в передаче управления конкретному (заранее определённому при проектировании программы) потоку управления.

Такой поток управления будем называть ведущим потоком управления группы функциональных потоков управления.

С учётом того, что операционная система реального времени включает развитые стандартные средства взаимодействия между потоками управления и средства синхронизации [7, 8], [4, 5, 6], описание программы в терминах операционной системы представляет собой содержательное функциональное описание вычислительного процесса, составляющего высокоуровневую алгоритмическую основу приложения.

Высокоуровневое описание приложение в терминах примитивов операционной системы позволяет осуществить явную проекцию предъявляемых к приложению функциональных требований на структуру программы. Такой подход делает прозрачным трассировку функциональных требований на структуру программы, что соответствует требования спецификации КТ-178 [9].

Отметим также, что излагаемый подход близок к методу послойного программирования, предложенному А.Л. Фуксманом [10].

Излагаемая ниже модель контролируемого выполнения базируется, тем не менее, на многопоточном представлении приложения. А именно, взаимодействие между управляющим компонентом и группой функциональных потоков управления осуществляется путём передачи команд отдельным потокам управления, составляющим ГФП.

В рамках этой статьи не будем углубляться в прикладную специфику приложения. Управление приложением со стороны **монитора** будем рассматривать как управление на уровне потоков управления. Это означает, что команды **монитора**, направляемые приложению, относятся к потоку управления в целом и не затрагивают внутреннюю программную структуру потока управления. Дополнительная детализация функционального наполнения потока управления напрямую зависит от специфики приложения. В настоящей статье ограничимся общей схемой (рисунок 1), базирующейся на примитивах операционной системы и действующей для многопоточных приложений реального времени, независимо от их специфики.

Агенты, регистрирующие и передающие контрольные параметры состояния, представляют собой функции, выполняемые в контексте потока управления. Параметры состояния содержат идентификационную информацию о потоке управления, в контексте которого исполняется агент, о ГФП, в состав которой входит поток управления, и контрольный идентификатор состояния программы. Один поток управления может содержать несколько агентов, и поэтому параметры состояния содержат также идентификационную информацию об агенте, сгенерировавшем параметры состояния. Конкретный формат параметров состояния будет приведён ниже.

Агенты помещают параметры состояния в очередь сообщений. Текущая реализация предусматривает наличие одной очереди сообщений для каждой группы функциональных потоков управления.

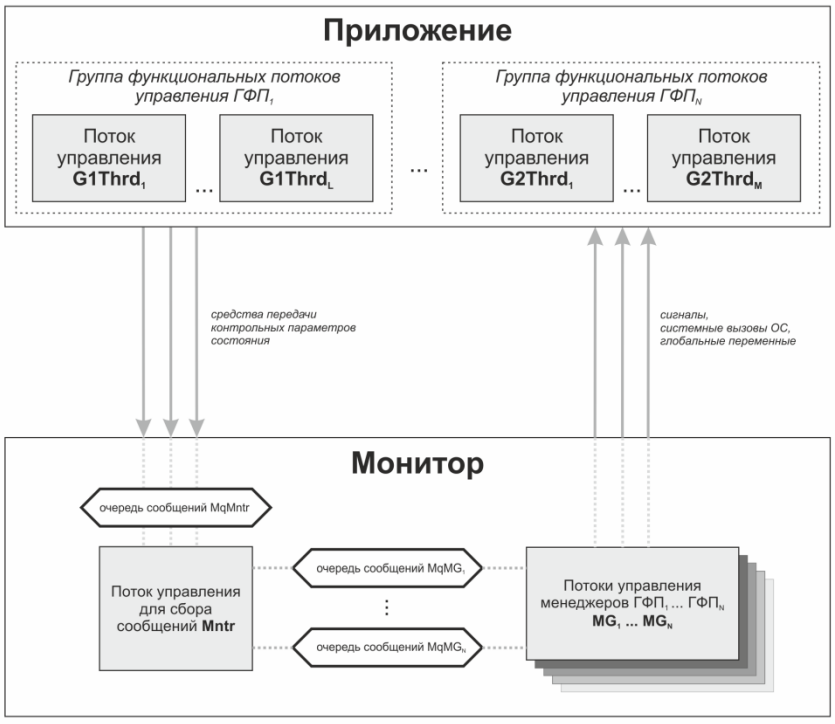
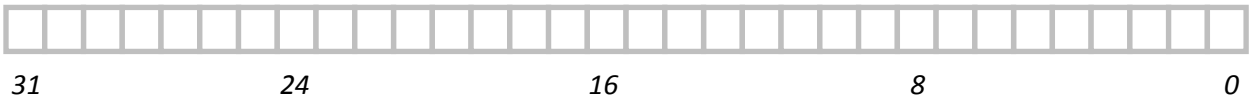


Рисунок 1 – Модель контролируемого выполнения

В соответствии с рисунком 1 данные от потоков управления приложения поступают в монитор через входную очередь сообщений. Каждое сообщение в очереди сообщений имеет следующий формат:



- битами **0-7** кодируются данные (аргумент *userdata* – см. раздел «Программная реализация агента»), передаваемые в монитор – это может быть команда, *errno* или значение переменной, анализируемой в данной контрольной точке;
- битами **8-15** кодируется идентификатор контрольной точки в
- потоке управления приложения;
- битами **16-23** кодируется тег потока управления приложения;
- битами **24-31** идентификатор группы функциональных потоков управления, в состав которой входит поток управления, передающий сообщение в монитор.

На основании полученных данных от потока управления приложения, монитор формирует реакцию. Команда от монитора упаковывается в сообщение типа int, формат аналогичен формату принимаемого сообщения. Сообщение пересылается менеджеру группы функциональных потоков управления, который имеет доступ к атрибутам (см. «Дополнительные атрибуты потоков управления») всех входящих в заданную группу потоков управления.

2 Состав и функции монитора

Монитор считывает сообщения из очереди сообщений, анализирует их, сравнивает с моделью корректного выполнения приложения, и, в зависимости от результатов анализа, формирует и направляет команды управления приложением.

В рамках предлагаемой архитектуры, команды, которые генерирует монитор, относиться к потоку управления в целом. Мы будем рассматривать следующие команды, реализация которых обеспечивается средствами операционной системы:

- завершить поток управления;
- приостановить поток управления;
- возобновить поток управления;
- приостановить поток управления на заданный промежуток времени (таймаут) и потом возобновить;
- перезапустить поток управления.

Необходимо отметить, что перезапуск потока управления требует специальных действий по удалению созданных потоком управления ресурсов, выполняемых в контексте перезапускаемого потока управления.

Предлагаемый список корректирующих действий для потоков управления аналогичен списку корректирующих действий для «процессов», предусматриваемых спецификацией ARINC 653 [11], выполняемых в результате корректирующих действий «монитора здоровья». Отметим, что понятие «процесс», используемое в ARINC 653, соответствует понятие «поток управления» из стандарта POSIX 1003.1 [8].

Характер ошибок, которые регистрируются агентами и передаются в монитор, в данном контексте не рассматривается, поскольку целью рассмотрения является архитектура программы реального времени с контролируемым выполнением и особенности ее программной реализации. Поэтому логика, лежащая в основе формирования команд управления приложением, также не является предметом настоящего рассмотрения.

В качестве средства передачи команд управления приложением и средства реализации управляющих воздействий используется единый механизм операционной системы, а именно, сигналы реального времени, системные вызовы (например, системный вызов порождения нового потока управления) и глобальные переменные.

3 Библиотека мониторинга

3.1 Особенности реализации

Предполагается, что группа функциональных потоков управления состоит из N потоков управления. В контексте этих потоков управления выполняются пользовательские функции, содержащие агентов контролируемого выполнения. Агент контролируемого выполнения – это функция, вызов которой встраивается в исходный код некоторого потока управления прикладной программы. Агенты, используя очередь сообщений к главному потоку управления монитора (на рисунке 1 – потоку управления Mntr), передают информацию о текущем состоянии приложения.

Монитор является многопоточной программой, в которой с каждой группой функциональных потоков управления приложения ассоциирован поток управления, выполняющий передачу команд от главного потока управления монитора к конкретному потоку управления приложения. На рисунке 1 – это потоки управления $MG_1 \ldots MG_N$.

Особенностью работы монитора является двухтактная система передачи информация от главного потока управления монитора (Mntr) к потокам управления приложения ($G1Thrd_1 \ldots G1Thrd_L$ и $G2Thrd_1 \ldots G2Thrd_M$). Информация на первом такте передается потоку управления $MG_к$ (где К – номер группы функциональных потоков

управления приложения) через очередь сообщений, а на втором такте из потока управления $MG_к$ в поток управления приложения посредством механизма сигналов. Данная система передачи информация используется для минимизации общего количества очередей сообщений между приложением и монитором.

Далее по тексту будем называть потоки управления $MG_1…MG_N$ *менеджерами* групп функциональных потоков управления.

3.2 Особенности интеграции прикладных функций

Использование библиотеки мониторинга предполагает, что программист выполнил проектирование системы в терминах потоков управления, а именно: определил количество групп функциональных потоков управления, количество потоков управления, входящих в каждую группу, и разработал для каждого потока управления головную функцию на языке Си.

Для удобства интегрирования библиотеки мониторинга в прикладную программу реального времени, спроектированную в терминах потоков управления, составляется файл конфигурации на языке Си. В первую очередь формируется описание групп функциональных потоков управления (в виде массива структур типа `usergfp`), а затем формируется описание потоков управления, входящих в заданную группу (в виде массива структур типа `userthreads`). Структура типа `usergfp` содержит поле `threads`, являющееся указателем на массив типа `userthreads`. Именно через это поле осуществляется связь между массивами, содержащими информацию о группах и входящих в эти группы потоках управления. Подробное описание полей структур `usergfp` и `userthreads` представлено в подразделе «Структуры данных» раздела «Программная реализация библиотеки мониторинга».

На основании описания, выполненного при конфигурировании системы, библиотека мониторинга выполняет запуск всех потоков управления прикладной программы и организует мониторинг запущенных потоков управления. Таким образом, при использовании библиотеки мониторинга прикладному программисту не требуется выполнять запуск потоков управления из прикладной программы – все описанные на этапе конфигурирования системы потоки управления будут запущены автоматически библиотекой мониторинга.

3.3 Дополнительные атрибуты потоков управления

В библиотеке мониторинга каждый запущенный поток управления описывается структурой данных типа `gfp_thread_pool`. С помощью этого описания потоку управления назначаются дополнительные атрибуты, необходимые для его идентификации и адресации в мониторе. Основные атрибуты это:

- *Идентификатор группы функциональных потоков управления*. Идентификаторы назначаются группам прикладным программистом на этапе конфигурирования.
- *Количество потоков управления в группе функциональных потоков управления*. Размер группы, в которую входит данный поток управления.
- *Тег потока управления*. Номер потока управления в группе функциональных потоков управления.
- *Идентификатор потока управления типа* `pthread_t`. Используется значение, полученное посредством системного вызова ОСРВ `pthread_self()`.
- *Флаг остановки потока управления* `is_frozen`. Используется для программной остановки потока управления.

При передачи команд от менеджера группы функциональных потоков управления заданному потоку управления приложения реальный `pthread_id` извлекается из массива `*gfp_thread_pool` по тегу потока управления (извлекается из полученного от агента сообщения, см. «Архитектура программы с контролируемым выполнением в среде операционной системе реального времени»), а затем потоку управления с заданным `pthread_id` посылается сигнал (`SIGRT_EXIT`, `SIGRT_FREEZE` и `SIGRT_UNFREEZE`, которым в ОС РВ соответствуют `SIGRTMIN+1`, `SIGRTMIN+2` и `SIGRTMIN+3`).

Отметим, что библиотека мониторинга автоматически маскирует указанные сигналы при порождении потока управления (`UNBLOCK`), а также при выполнении прерываемых сигналами системных вызовов (`BLOCK`), таких как `mq_receive()` или `mq_send()`.

Доступ к очереди сообщений для передачи данных менеджеру группы функциональных потоков управления монитор получает с помощью системного вызова `mq_open()`, в который в качестве аргумента передается предопределенное имя, ассоциированное с заданной очередью сообщений: `/gfp%d_monitor_in`, где `%d` – идентификатор группы функциональных потоков управления, который извлекается из полученного от агента сообщения.

3.4 Команды управления

Как было отмечено ранее в библиотеке мониторинга используются следующие команды управления:

- *Продолжить выполнение потока управления прикладной программы*. После приема сообщения с данной командой *менеджер* группы функциональных потоков

управления сбрасывает атрибут `is_frozen` заданного потока управления в 0.

- *Завершить поток управления прикладной программы*. После приема сигнала поток управления выполняет системный вызов `pthread_exit()`.
- *Приостановить выполнение потока управления (в том числе и на предопределенный таймаут)*. После приема сигнала поток управления устанавливает атрибут `is_frozen` в 1 и переходит в ожидание сброса этого атрибута в 0.

Реализация команды перезапуска потока выполнена следующими способами (предпочтительным к использованию является перезапуск потока управления с использованием функций свертывания, тем не менее выбор конкретного способа перезапуска потока управления остаётся за программистом):

- *Грубый перезапуск потока управления*. После приема сигнала поток управления выполняет системный вызов `pthread_exit()`. Далее *менеджер* группы функциональных потоков управления фиксирует завершение заданного потока управления и порождает его повторно с помощью вызова `pthread_create()`. При этом не гарантируется, что вновь созданный поток управления будет работоспособным, так как завершение было принудительным и не выполнялось освобождение ресурсов (например, файловых дескрипторов), связанных с потоком управления.
- *Перезапуск потока управления с использованием функций свертывания*. В библиотеке мониторинга вводится дополнительный атрибут `is_restartable`. Потоки управления с установленным при конфигурировании флагом `is_restartable` должны быть снабжены функциями свертывания. Реализация функций свертывания лежит на прикладном программисте. Библиотека мониторинга при запуске такого потока управления выполнит регистрацию пользовательских функций свертывания с помощью системных вызовов `pthread_cleanup_push()` и `pthread_cleanup_pop()`. Таким образом, при принудительном завершении заданного потока управления, ресурсы будут освобождены функциями свертывания.

4 Программная реализация библиотеки мониторинга

4.1 Структуры данных

В библиотеке мониторинга для описания, адресации и идентификации запущенных потоков управления используются следующие структуры данных:

- `gfp_thread_pool` – содержит описание дополнительных атрибутов потока управления (атрибуты, назначаемые и используемые библиотекой мониторинга, структура инициализируется библиотекой при запуске потоков управления);
- `usergfp` – содержит описание прикладной группы функциональных потоков управления (поля структуры должны быть проинициализированы программистом при конфигурировании прикладной программы);
- `userthreads` – содержит описание потоков управления, входящих в прикладную группу функциональных потоков управления (поля структуры должны быть проинициализированы программистом при конфигурировании прикладной программы).

В состав структуры `usergfp` входят следующие поля:

- `name` – имя группы (тип `char *`);
- `id` – идентификатор группы (тип `int`);
- `size` – количество потоков управления, входящих в группу (тип `int`);
- `threads` – указатель на массив описаний потоков управления группы (тип `userthreads*`).

В состав структуры `userthreads` входят следующие поля:

- `name` – имя потока управления (тип `char *`);
- `thrd_func` – указатель на головную функцию потока управления (тип `int *`);
- `_tpoolptr` – указатель на структуру с описанием атрибутов потока управления (тип `gfp_thread_pool*`, при конфигурировании поле должно быть проинициализировано прикладным программистом в значение `NULL`).

4.2 Программная реализация агента

Прототип функции агента библиотеки мониторинга:

```
int checkpoint_agent(struct gfp_thread_pool *tpool, int usercheckpoint_id, int userdata);
```

Функции передаются следующие аргументы:

- `tpool` – указатель на структуру с атрибутами потока управления, в контексте которого выполняется агент;
- `usercheckpoint_id` – идентификатор агента (номер контрольной точки);
- `userdata` – данные для передачи в монитор.

Упрощенный алгоритм работы *агента* библиотеки мониторинга представлен в листинге 1.

```
алг
нач
  если tpool <> NULL и usercheckpoint_id > 0
    то
      сформировать(сообщение, userdata);
      mq_send(сообщение);
  все
кон
```

Листинг 1 – Алгоритм работы агента библиотеки мониторинга

4.3 Программная реализация менеджера группы функциональных потоков управления

Упрощенный алгоритм работы менеджера группы функциональных потоков управления библиотеки мониторинга представлен в листинге 2.

```
алг
нач
  подсчитать(количество_потоков_управления_в_гфп);
  если количество_потоков_управления_в_гфп > 0
    то
      если mq_open("/монитор->менеджер") > 0
        то
          нц
            если mq_receive(сообщение) > 0
              то
                декодировать(сообщение);
                pthread_kill();
            все
          кц
      все
  все
кон
```

Листинг 2 – Алгоритм работы менеджера группы функциональных программ

4.4 Программная реализация монитора

Упрощенный алгоритм работы главного потока управления монитора представлен в листинге 3.

```
алг
нач
  если mq_open("/агент->монитор") > 0
    то
      нц
        если mq_receive(сообщение) > 0
          то
            декодировать(сообщение);
            определить(действие);
            если mq_open("/монитор->менеджер") > 0
              то
                сформировать(сообщение, действие);
                mq_send(сообщение);
                mq_close("/монитор->менеджер");
            все
        все
      кц
  все
кон
```

Листинг 3 - Алгоритм работы главного потока управления монитора

5 Результаты тестирования библиотеки мониторинга

Для тестирования библиотеки мониторинга была использована контрольная задача, состоящая из пяти групп функциональных потоков управления, в каждой группе работало пять потоков управления, и в каждом потоке управления было установлено пять контрольных точек (агентов мониторинга).

Каждый из потоков управления выполнил 1 млрд. итераций. При выполнении потоков управления монитор реагировал на искусственно порождаемые в потоках управления сбои и направлял в эти потоки управления команды, соответствующие сбоям (см. подраздел «Команды управления» раздела «Архитектура библиотеки мониторинга»).

При выполнении контрольной задачи сбоев в работе как библиотеки мониторинга, так и самой контрольной задачи, не выявлено.

Литература

1. В.А. Галатенко, Контролируемое выполнение / В.А. Галатенко, К.А. Костюхин, К.А., Н.В. Шмырев – М: НИИСИ РАН, 2012. – 157 с.
2. В.Б. Бетелин, Контролируемое выполнение с явной моделью / В.Б. Бетелин, В.А. Галатенко, К.А. Костюхин – ПРОГРАММИРОВАНИЕ, 2014, N- 6, с. 45-55.
3. Галатенко В.А., Вьюкова Н.И., Самборский С.В., Трифонов С.И. Программирование в среде VxWorks. - М.: Наука, 1997.
4. Безруков В.Л., Годунов А.Н., Назаров П.Е., Солдатов В.А., Хоменков И.И, Введение в ос2000, Вопросы кибернетики. Информационная безопасность, Операционные системы реального времени, Базы данных/Под ред. чл.-корр. РАН В.Б.Бетелина. - Москва; НИИСИ РАН, 1999, - С. 76 - 106.
5. Годунов А.Н., Солдатов В.А., Особенности отладки встроенных систем – Программная инженерия. 2013, №3, с. 2 – 8
6. Годунов А.Н., Солдатов В.А., Операционные системы семейства Багет (сходство, отличия, перспективы) – Программирование, 2014, №5, с. 68 - 76
7. David R. Butenhof, Programming with POSIX Threads, Addison-Wesley, 1997
8. International Standard ISO/IEC 9945-1 (ANSI/IEEE Std 1003.1) Second Edition. 1996-07-12. Information Technology - Portable Operating System Interface (POSIX) - Part 1: System Application Program Interface (API) [C Language].
9. КТ-178. Квалификационные требования. Требования к программному обеспечению бортовой аппаратуры и систем сертификации авиационной техники. – Межгосударственный авиационный комитет, Авиационный регистр, 1996.

10. А.Л. Фуксман, Технологические аспекты создания программных систем / А.Л. Фуксман. – М.: Статистика, 1979 – 184 с.
11. ARINC Specification 653P1-3: Avionics Application Software Standard Interface Part 1 - Required Services. Aeronautical Radio INC, Maryland, USA, 2010.