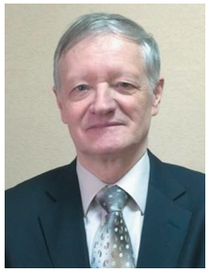


Колчин А.Ф., Михеев Н.В.

АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ СИСТЕМЫ, СВЯЗАННОЙ С БЕЗОПАСНОСТЬЮ

В статье представлены результаты анализа архитектур программного обеспечения для систем, связанных с безопасностью; выделены основные компоненты архитектуры и предложено определение архитектуры программного обеспечения систем, связанных с безопасностью; предложены критерии качества и проведено сравнение выделенных компонентов по данным критериям. В работе представлены результаты сравнения применимых структур и стилей, которые предлагается использовать при разработке программного обеспечения, удовлетворяющего требованиям функциональной безопасности, заданным в ГОСТ Р МЭК 61508.

Ключевые слова: *надежность, функциональная безопасность, архитектура, программное обеспечение.*

Введение

Современная тенденция по более строгому выполнению требований безопасности к опасным техническим системам привела к применению мер по обеспечению безопасности не только достаточных, но и действительно необходимых. Однако для этого требуется некоторая численная оценка используемых мер по обеспечению безопасности, которую можно и нужно задавать до начала создания таких систем.

Для оценки аппаратных средств систем безопасности используются в основном статистические методы, позволяющие получить численную оценку надежности аппаратных средств. Для программного обеспечения подобные методы неприменимы, так как ошибки, возникающие в программном обеспечении при его создании и эксплуатации, являются систематическими.

Одним из подходов, объединяющих методы оценки надежности как для аппаратных средств, так и для программного обеспечения, является методология функциональной безопасности, представленная в ГОСТ Р МЭК 61508 [1]. Данный стандарт охватывает разработку относительно простых систем, обеспечивающих выполнение, как правило, одной функции – функции безопасности – и называющихся «системами, связанными с безопасностью».

Одним из ключевых этапов в создании программного обеспечения систем, связанных с безопасностью, является разработка архитектуры программного обеспечения. Но исчерпывающие руководства по разработке архитектур программных систем в настоящее время отсутствует. Существующие работы рассматривают или определенный подвид архитектур, или общие методы проектирования. Одной из причин, тормозящих систематизацию знаний в данной области, является отсутствие общепринятого определения архитектуры программного обеспечения.

В настоящей статье представлены результаты анализа архитектур программного обеспечения и их разработки для систем, связанных с безопасностью, выделены основные

компоненты архитектуры и предложено определение архитектуры программного обеспечения систем, связанных с безопасностью, предложены критерии качества и проведено сравнение выделенных компонентов по данным критериям. В работе представлены результаты сравнения применимых структур и стилей, которые предлагается использовать при разработке программного обеспечения, удовлетворяющего требованиям функциональной безопасности.

1. Система, связанная с безопасностью

1.1. Структура системы

Достаточно длительный опыт разработки и применения систем, связанных с безопасностью, в различных областях позволяет утверждать, что структура такой системы уже устоялась и состоит из следующих компонент:

1. Программируемая электроника:

а) контроллер, выполняющий логику и осуществляющий взаимодействие между компонентами;

б) программное обеспечение, реализующее логику.

2. Входные/выходные устройства и вспомогательные средства:

а) датчики и прочие устройства ввода;

б) исполнительные устройства и прочие устройства вывода;

в) источники питания, средства коммуникации и др.

1.2. Требования к системе

Согласно модели жизненного цикла системы, связанной с безопасностью [1], к началу процесса создания ее архитектуры программного обеспечения разработан уже большой объем документов, описывающих управляемое оборудование, функции безопасности и различные требования к системе программного обеспечения, выявленные на предыдущих стадиях. Многие требования являются типичными для большинства разрабатываемых систем, связанных с безопасностью. Анализ требований к описанной выше структуре позволил выделить следующие существенные для программного обеспечения требования и ограничения к системе, связанной с безопасностью:

1. Строго заданное максимальное время отклика системы.

2. Разрабатываемое программное обеспечение предназначено для работы на программируемых контроллерах с ограниченными ресурсами.

3. Программное обеспечение сильно связано с более низкоуровневыми задачами (например, вводом-выводом), поэтому строго привязано к операционной системе (или даже реализует ее функции).

Таблица 1.Преимущества и недостатки структур программ

Структура	Преимущества	Недостатки
Простой цикл управления	Простота	Худшее время реакции равно сумме времен выполнения всех подзадач[1]. При добавлении новых устройств или дополнительной обработки увеличивается худшее время реакции
Система, управляемая прерываниями	Позволяет существенно увеличить время реакции для высокоприоритетных подзадач по сравнению с простым циклом управления. Является развитием идеи простого цикла управления, но остается достаточно простым	Проблема раздельной памяти, поскольку когда появляется прерывание, основной поток может остановиться на середине вычислений, в то время как код, выполняемый по прерыванию может изменить данную память
Совместная многозадачность	Отсутствие необходимости защищать все разделяемые структуры данных. Упрощает процесс переноса однопоточного кода в многопоточную среду. Достаточно простое расширение функционала (путем добавления подпрограммы в очередь выполнения)	При сбое одного потока отказывают все прочие, поскольку не вызывается операция «отдать процессорное время» Высокая сложность реализации многопоточного ввода-вывода
Вытесняю-щая многозадач-ность / RTOS	Возможность полноценной реализации многопоточного ввода-вывода. Возможность использования многопроцессорных систем. Одна отказавшая программа не ока-зывает влияния на другие программы. В случае отсутствия использования общей памяти программист может раз-рабатывать программу как однопоточную – всю работу выполняет операционная система	Накладные ресурсы на выполнение задач самой операционной системой. Сильное усложнение системы в целом. Проблема раздельной памяти

Таблица 2. Сравнение рассматриваемых структур программы по критериям качества

	Возможность установки приоритетов у подпрограмм	Независимость худшего времени реакции от колличества подпрограмм	Независимость худшего времени реакции от изменения подпрограммы	Общая простота реализации системы	Простота реализации подпрограмм	Простота добавления подпрограмм	Отсутствие накладных расходов, не связаных с основной логикой	Возможность написания «однопоточного» кода	Независимость подпрограмм
Простой цикл управления	-	-	-	+	+	+	+	+	-
Система, управляемая прерываниями	-	-	-	+	-	+	+	-	-
Совместная многозадачность	Задается программистом	-	-	-	-	+	-	-	-
Вытесняющая многозадачность / RTOS	+	+	+	-	+	+	-	+	+
Обозначения: «+» - критерий применяется; «-» - критерий не применяется.									

4. Отсутствие необходимости хранения больших объемов данных.

5. Отсутствие сложной бизнес-логики, поскольку в большинстве случаев крайне не рекомендуется реализовывать на контроллере функции, не связанные с безопасностью[1].

6. Взаимодействие с пользователем минимально, работа в основном автономна.

1.3. Требования к архитектуре программного обеспечения

Можно выделить три группы требований к архитектуре программного обеспечения[1]:

1. Требования уровня полноты безопасности к архитектуре программного обеспечения и инструментам проектирования.

2. Общие требования со стороны системы, связанной с безопасностью. Данные требования прямо или косвенно могут влиять на архитектуру.

3. Частные требования к программному обеспечению со стороны архитектуры аппаратного обеспечения.

Первая группа включает в себя различные требования, например, к документальному оформлению процесса, прослеживаемости и т.п. Также данная группа включает методы разработки архитектуры, требуемые для конкретного применения в зависимости от уровня полноты безопасности.

Вторая группа включает в себя ограничения по ресурсам, размерам, интерфейсам ввода-вывода и т. д., которые зависят от конкретной задачи. Однако можно утверждать, что данные требования в основном ограничивают ресурсоемкость архитектуры [2].

Источником третьей группы может быть вид разработки, например, если необходимо надстроить разрабатываемую систему над имеющейся основой, предоставляемой производителем контроллера. Кроме того, они могут быть связаны с особенностями аппаратного обеспечения, когда выбранный контроллер реализует специальную технологию, оптимизированную для определенной архитектуры программного обеспечения.

Проектирование программного обеспечения обычно ведется тремя разными способами [3]:

а) Разработка нового программного обеспечения на основе спецификации.

б) Разработка программного обеспечения для интеграции в существующую платформу. Данный процесс больше ориентирован не на проектирование и разработку программного обеспечения, а на отображение требуемого функционала на заданный программный каркас (framework).

в) Доработка существующего программного обеспечения. Данный процесс не рассматривается в настоящей работе, поскольку действия по доработке существенно зависят от дорабатываемой системы.

Как вариант (а), так и вариант (б) являются приемлемыми при создании программного обеспечения для новой системы, поскольку интеграция в существующую

[1] В [1] нет каких-либо ограничений на размеры или сложность программ, однако с повышением требований к отказоустойчивости к системе, усложняются методы обеспечения ее уровней полноты безопасности, поэтому рекомендуется реализовывать более простые (как вариант – реализовывать частично) функции безопасности на отдельной системе, связанной с безопасностью.

Таблица 3. Преимущества и недостатки стилей архитектур

Архитектура	Преимущества	Недостатки
Конвейеры и фильтры	Возможность представления всего поведения программы как простой последовательности отдельных фильтров. Повторное использование фильтров. Простое добавление нового функционала путем добавления фильтра в очередь обработки. Возможно проведение специальных проверок, таких как анализ на взаимные блокировки или пропускной способности. Потенциально простое распараллеливание кода	Зачастую требуется организация пакетной обработки. Ввиду требования независимости фильтров, проектировщику приходится считать, что данные полностью перерабатываются каждым фильтром. В частности, может потребоваться каждый раз приводить данные к общему виду и в каждом фильтре отдельно их разбирать
Абстракция данных и объектно-ориентированная организация	Возможность изменения реализации без последствий для клиентов. Объединение данных и функций, которые их обрабатывают, позволяет проектировщикам декомпозировать задачу до набора взаимодействующих между собой сущностей	Ввиду специфики процедурных вызовов, вызывающий объект должен иметь явный доступ к вызываемому, в отличие от конвейеров или событийной системы. Как следствие – при изменении идентификатора объекта требуется явно оповестить все вызывающие объекты. Возможно наличие сторонних эффектов (например, если объект A использует B и объект C также использует B, то изменения объекта B объектом C выглядят как сторонние эффекты для объекта A, и наоборот)
Событийная система, неявный вызов	Большие возможности по повторному использованию: в систему может быть добавлен новый компонент путем простой регистрации как обработчика событий. Неявный вызов упрощает развитие системы: любой компонент может быть переделан или заменен без влияния на прочие компоненты	Контроль над проводимыми вычислениями лежит не на компонентах программного обеспечения, а на системе. Нет гарантии реакции на событие Как следствие предыдущего положения, подтверждение реакции на событие должно производиться явно. Сложные процедуры обмена большими объемами данных
Уровневая (слоевая) система	Легкое наращивание абстракций. Широкие возможности повторного использования, подобно абстрактным типам данных. Возможность реализации отдельных уровней разными способами путем декларации интерфейсов взаимодействия между уровнями	Не все задачи можно достаточно просто разложить на уровневую структуру. Достаточно сложно определить подходящий уровень абстракции. Наличие дополнительных накладных расходов ввиду перевода абстракции с одного уровня на другой

платформу рассматривается как использование вновь уже существующего программного обеспечения, а не его доработка.

2. Проектирование архитектуры программного обеспечения

2.1. Определение архитектуры программного обеспечения

Среди разработчиков различных систем существует общее понимание важности архитектурного уровня создаваемой системы. Однако пока еще не появилось никакого устоявшегося согласия по точному определению архитектуры системы вообще.

Выполненный анализ показал, что архитектуру программного обеспечения систем, связанных с безопасностью, можно представить и описать с помощью следующих трех инвариантных составных частей (которые иногда называют *шаблонами архитектуры*):

1. Структура программы [4] описывает способы организации базовых функций, таких как управление памятью, управление потоками и т.п.

2. Стиль архитектуры [5] описывает способ логического деления системы, которое реализуется на основе структуры системы.

3. Решение задачи, описываемое в терминах структуры программы и стиля архитектуры.

Таким образом, в настоящей статье под архитектурой программного обеспечения систем, связанных с безопасностью, будем понимать набор представлений и описаний:

- структуры программы;
- стиля архитектуры;

Таблица 4. Сравнение стилей архитектуры

	Простота	Поддерживаемость	Повторное использование	Производительность	Масштабируемость	Переносимость
Конвейеры и фильтры	+	+ Легко заменить фильтр	+ Позволяет добиться разных эффектов за счет смены порядка	- Нет взаимосвязи между фильтрами (нельзя передать управление из одного фильтра к другому)	±	- Не переносимо
Абстракция данных и объектно-ориентированная организация	±	+ Множество принципов, позволяющих увеличить абстракцию (инкапсуляция, инверсия управления и т.п.)	+	±	±	±
Событийная система, неявный вызов	- Иногда поведение непредсказуемо, сложно управлять	+ Заменить или убрать компонент можно без влияния на другие	+ Компоненты могут быть зарегистрированы в системе для обработки любых событий	- Компоненты не имеют возможности управлять вычислениями	- Могут возникать сторонние эффекты, если два компонента используют третий	+ Компоненты могут реагировать на любые события
Уровневая (слоевая) система	±	+ Изменение одного уровня воздействует только на два соседних	+ Не зависят от вышележащих уровней	±	+ Может быть объединен с другими стилями	
Обозначения: «+» – критерий применяется; «-» – критерий не применяется; «±» – критерий применяется частично.						

- выполняемой функции безопасности в терминах структуры программы и стиля архитектуры.

2.2. Структура программы

Системы, связанные с безопасностью, по сути, являются подклассом встраиваемых систем, поэтому варианты организации структуры программы довольно схожи с вариантами, применяемыми в обычных встраиваемых системах. Для реализации программного обеспечения систем, связанных с безопасностью, можно использовать следующие структуры программ [6]:

1. Простой цикл управления (simple control loop).
2. Система, управляемая прерываниями (interrupt controlled system).
3. Совместная или кооперативная многозадачность (cooperative multitasking).
4. Вытесняющая многозадачность (preemptive multitasking) или многопоточность (multi-threading).
5. Прочие варианты операционных систем реального времени.

В таблице 1 представлены полученные в результате анализа преимущества и недостатки перечисленных структур при создании систем, связанных с безопасностью, что позволило выделить следующие критерии сравнения рассматриваемых структур.

1. Возможность установки приоритетов у подпрограмм.
2. Независимость худшего времени реакции:
   а) от количества подпрограмм;
   б) от изменения подпрограммы.
3. Общая простота реализации системы.
4. Простота реализации подпрограмм.
5. Простота добавления подпрограмм.
6. Отсутствие накладных расходов, не связанных с основной логикой решаемой задачи.
7. Возможность написания «однопоточного» кода.
8. Независимость подпрограмм.

В таблице 2 рассмотренные структуры сравниваются по выделенным критериям.

2.3. Стиль архитектуры

Существует множество различных архитектурных конструкций, часто называемых шаблонами архитектуры, которые представляют собой решения в рамках некоторо-

Таблица 5. Возможные сочетания структуры и стиля архитектуры

	Простой цикл управления	**Система, управляемая прерываниями**	**Совместная многозадачность**	**Вытесняющая многозадачность / RTOS**
Конвейеры и фильтры	Существуют разработки, показывающие возможность эффективно обрабатывать данные в такой конфигурации, однако основная область применения данного сочетания технологий – управление сетевым трафиком [17]	В общем случае такое сочетание не применяется	Конвейеры и фильтры можно рассматривать как вариант совместной многозадачности	Сочетание возможно
Абстракция данных и объектно-ориентированная организация	Сочетание возможно	Сочетание возможно	Сочетание возможно	Сочетание возможно
Событийная система, неявный вызов	Несовместимо	Сочетание возможно	Сочетание возможно	По сути, RTOS является операционной системе, основанной на событиях. Как минимум, это один из основных инструментов, позволяющий назначать приоритеты отдельным процессам. Также, пересылка события является самым простым способом взаимодействия между потоками.
Уровневая (слоевая) система	Сочетание возможно	Сочетание возможно	Сочетание возможно	Сочетание возможно

го повторяющегося контекста. На сегодняшний день нет какого-то единого списка подобных шаблонов, или даже единого мнения по поводу того, насколько они должны быть абстрактными, однако в рамках задачи, рассматриваемой в настоящей работе, можно выделить следующий набор шаблонов (или стилей) архитектур[1] [3]:

1. Конвейеры и фильтры (pipes and filters).
2. Абстракция данных и объектно-ориентированная организация (Data Abstraction and Object-Oriented Organization).
3. Событийная система, неявный вызов (Event-based, Implicit Invocation).
4. Уровневая (слоевая) система (Layered system).

В таблице 3 рассмотрены преимущества и недостатки перечисленных стилей программ при создании систем, связанных с безопасностью.

В работе для сравнения стилей архитектуры были использованы следующие критерии качества [7]:

1. Поддерживаемость – степень простоты внесения изменений (добавление новых обработчиков, удаление старых, модификация существующих и т.п.).
2. Повторная используемость – степень применимости составных частей существующих программ при создании новых.
3. Производительность.
4. Простота – понятность системы для новых разработчиков, возможность быстро разобраться в существующих модулях.
5. Масштабируемость.
6. Переносимость – отсутствие привязки к определенным инструментам (среде разработки, операционной системе, компилятору и т.п.).

В таблице 4 рассмотренные стили сравниваются по перечисленным критериям.

Безусловно, не все стили можно применять при реализации рассмотренных структур программ. В таблице 5 представлены все возможные сочетания рассмотренных выше структур и стилей программ. Применение данной таблицы позволит повысить корректность и эффективность разработки программного обеспечения систем, связанных с безопасностью, на этане выбора архитектуры программного обеспечения.

2.4. Решение задачи

Как следует из определения, при формировании архитектуры используется высокоуровневое представление функций безопасности, а не ее спецификация, сформированная на соответствующей стадии жизненного цикла системы, связанной с безопасностью. Необходимая детализация выбирается исходя из требований к раз-

[1] Безусловно, возможны как и комбинации указанных архитектур, так и совершенно новые

рабатываемой системе, связанной с безопасностью. В частности, для архитектуры программного обеспечения должно быть описано следующее:

1. Основные компоненты системы, их интерфейсы, компоновка и способы взаимодействия друг с другом.

Архитектура должна отражать высокоуровневое представление компонентов системы (подсистем), наиболее важных с точки зрения большинства участников разработки программного обеспечения, поскольку их требуется учитывать при разработке большинства прочих компонентов системы. Одни подсистемы требуются для работы большинства компонентов (например, подсистема ввода-вывода, инкапсулирующая физический уровень взаимодействия с каналами связи). Другие – являются источниками событий для прочих компонентов (например, сторожевой таймер). Третьи компоненты могут быть наиболее ресурсоемкими и/или критичными для выполнения основной функции системы – функции безопасности.

2. Ключевые шаблоны проектирования и технологии, применяемые в проекте.

Поскольку определенные шаблоны могут влиять на программу в целом (например, пул соединений), то их требуется описать на уровне архитектуры. В том числе, паттерны, которые допустимо использовать на более низких уровнях.

3. Интерфейсы взаимодействия с внешними системами.

4. Сервисы, обеспечивающие поддержку работы основного функционала (ведение журналов, хранение данных и т.п.)

Заключение

В работе представлены результаты анализа архитектур программного обеспечения и их разработки для систем, связанных с безопасностью, а также определены основные компоненты архитектуры.

Предложено определение архитектуры программного обеспечения систем, связанных с безопасностью.

В работе представлены результаты сравнения применимых структур и стилей, которые предлагается использовать при разработке программного обеспечения, удовлетворяющего требованиям функциональной безопасности [1].

Использование рассмотренного подхода позволит повысить корректность и эффективность разработки программного обеспечения систем, связанных с безопасностью, на этапе выбора архитектуры программного обеспечения.

Литература

1. ГОСТ Р МЭК 61508–2012 Функциональная безопасность систем электрических, электронных, программируемых электронных, связанных с безопасностью. Части 1 – 7.

2. **Philip J. Koopman Jr.**, Design constraints on embedded real time control systems. Systems Design & Networks Conference, (1990).

3. **Douglas Densmore, Roberto Passerone, Alberto Sangiovanni-Vincentelli**, A Platform-Based Taxonomy for ESL Design. IEEE Software.

4. **David Stonier-Gibson,** Understanding embedded microcontroller multitasking RTOS alternatives.

5. **David Garlan, Mary Shaw**, An Introduction to Software Architecture.

6. http://en.wikipedia.org/wiki/Embedded_system#Embedded_software_architecturs

7. **Mary Shaw** (1995), Comparing Architectural Design Styles. IEEE Software.