Обработка исключений в Symbian OS

Автор: Бен Моррис
Перевод: Александр Смирнов

Опубликовано на сайте Symbian Developer Network

Версия: 1.02 – июль 2008

1 Вступление

До выхода Symbian OS версии 9, система поддерживала только нестандартную обработку исключений C++, используя функции «сброса» и стек очистки. Если вы когда-нибудь работали с Symbian C++, то вероятно уже заметили эту особенность из числа первых, отличающих Symbian C++ от стандартного C++.

Начиная с Symbian OS версии 9, система так же начала поддерживать стандартную обработку исключений C++. Это позволяет разработчику сделать выбор между оригинальным способом обработки исключений в Symbian C++, и стандартным способом обработки исключений C++, а так же дает возможность (с некоторыми ограничениями) смешивать их между собой.

Это, безусловно, хорошая новость, так как подобное решение позволяет развиваться Symbian C++ в потоке общепринятого течения, и дает новым разработчикам возможность более быстрого и легкого портирования имеющегося C++ кода на платформу Symbian. Однако так как оригинальный механизм обработки исключений по-прежнему поддерживается системой, разработчики могут оказаться в замешательстве при решении вопроса: которым из способов воспользоваться, и когда.

Данная статься представляет собою обзор обоих доступных механизмов обработки исключений C++ в Symbian OS. Так же здесь будут обсуждаться вопросы, который из механизмов приемлем, и когда, и как эти два механизма обработки исключений могут быть безопасно смешаны в коде.

2 Исключения, ошибки и дефекты

Исключением является реакция приложения на ошибку (возникшую по какой-либо причине), которое приложение не может обработать локально. Механизм обработки исключения заключается в способе передачи данных об ошибке другому модулю или коду, который специально *подготовлен* для обработки таких ошибок.

Как указывает Бьярне Строуструп (1994, страница 384), исключения являются концепцией уровня языка программирования. Разные языки программирования исповедуют разные подходы к обработке исключений. К примеру, Java и Python советуют разработчикам использовать исключения для обработки ***всех*** ошибок. Такой подход делает возвращаемые значения функций совершенно ненужными, и позволяет получить более чистый, удобочитаемый и менее «завинченный» код.

В противоположность этим языкам, C++ (как описывает Строуструп) использует более консервативный подход, оставаясь ближе к своим истокам из языка C, и придерживается принципа *использования исключений только для исключительных ситуаций* (Kernighan and Pike, 1999). О «неисключительных» ошибках программа должна извещать через возвращаемые коды ошибок, или через возвращаемый `NULL`-указатель, как это обычно делается в C-программах. При этом для «настоящих» ошибок обработка исключений будет использоваться весьма скупо.

Поэтому в C++, когда код может предотвратить возможную ошибку, он должен постараться ее предотвратить (например, проверив, что необходимый файл существует и доступен для чтения). Только в тех случаях, когда код не может предотвратить ошибку, он должен обратиться к механизму обработки исключений, и постараться таким образом решить возникшую проблему.[1]

Во всех системах генерация исключения является способом изменения нормального хода выполнения программы. При этом вместо прямой обработки ошибки, в стек вызова передается информация о возникшей проблеме, и ошибку начинает обрабатывать любой заинтересованный в этом код (даже не смотря на тот факт, что в Java и Python исключение может «путешествовать» только в пределах смежного блока кода.

---

[1] Такой подход является «классическим» в C++, и не описан в стандарте языка. Поэтому программисты могут сами решать когда они должны генерировать исключение, а когда – вернуть код ошибки.

Не смотря на различные философии языков, существуют определенные виды ошибок, которые программы никогда не должны пытаться обработать сами. Например, дефекты в самой программе. Когда в программе находятся такие дефекты, программа должна просто прекратить свою работу, а не пытаться справиться с ними.

Подобным образом существуют определенные виды ошибок, которые программа всегда должна обрабатывать локально. Например, ошибки ввода данных: программа никогда не должна прекращать свою работу, если пользователь неправильно набрал свое имя, или попытался ввести некорректные данные в диалог ввода.

Мы можем составить очень простую классификацию ошибок используя следующие принципы:

- Какие-то ошибки всегда должны останавливать программу.
- Какие-то ошибки никогда не должны останавливать программу.
- Какие-то ошибки лучше всего обрабатывать сразу же, и локально.
- О некоторых ошибках лучше всего сообщить пользователю или вышестоящим компонентам.

Языки программирования (и некоторые программные системы) различаются по тому, как они впишутся в представленную ниже матрицу, созданную на основе данной классификации ошибок. Различия так же заключаются в поддержке различных методов обработки ошибок, представленных в следующих категориях:

	Сообщается	**Не сообщается**
Выполнение программы прекращается	Фатальная ошибка, обработанная на системном уровне	Фатальная ошибка, обработанная локально
Выполнение программы не прекращается	Исключение	Не исключение

Исключениями являются ошибки, о которых программа всегда извещает свои компоненты (и изменяет ход своего исполнения), однако автоматическая остановка работы при этом не подразумевается (правда обработчик исключения имеет полное право остановить работу программы, если сочтет это нужным).

Возвращаясь обратно к C++, Строуструп в качестве принципиальной цели дизайна языка C++ описывает возможность разработки систем, устойчивых к ошибкам и неисправностям (1994, глава 16). Однако под этим не подразумевается, что любая части каждой программы C++ должна быть способна к обработке возникающих ошибок; так же не каждая функция должна обрабатывать все возникающие ошибки. Основной причиной наличия обработки исключений в C++ является насущная необходимость программ в возможности сообщать и обрабатывать возникающие ошибки в соответствии с текущими возможностями. Что полностью согласуется с основным принципом построения устойчивых систем, гласящим, что они должны быть многоуровневыми. Другими словами, «в какой-то момент компонента должна прекратить свои попытки справиться с ошибкой, и просто передать управление более "вышестоящей" компоненте» (Stroustrup, 1994, страница 385).

3 Изначальная обработка исключений в Symbian OS

С самых первых своих версий Symbian OS предлагает своим разработчикам встроенный и очень простой способ обработки исключений C++. Исключения инициализируются при помощи «сбросов», и ловятся при помощи макросов-ловушек. Когда какой-либо метод вызывает сброс, нормальный ход выполнения программы тут же прерывается, и информация о сбросе передается через стек вызова в вышестоящие функции до тех пор, пока сброс не будет пойман макросом-ловушкой. Если программа, в которой случился сброс, не поймает и не обработает сброс, за работу примется сама система. Результат такого вмешательства может варьироваться от платформы к платформе, но обычно информация о сбросе демонстрируется системой в виде паники.

Вместе с механизмом сбросов и ловушек, Symbian OS так же предлагает механизм само-очистки, названный «стеком очистки». В случае возникновения сброса стек очистки позволяет

автоматически удалять помещенные в него объекты, предотвращая таким образом возможные утечки памяти.

В остальном Symbian C++ следует тем же принципам, что и стандартный C++: исключения используются только в исключительных ситуациях, а для «предсказуемых», или «неисключительных» ошибок больше используются возвращаемые коды этих самых ошибок.

Метод сбросов и ловушек был спроектирован в виде структурированной и объектно-ориентированной альтернативы стандартной обработки исключений, доступной в «чистом» C++, и основанной на вызовах команд `setjmp` и `longjmp` (в то время это был единственный метод обработки исключений в компиляторе C++, использованном для компиляции операционной системы).

До Symbian OS версии 9, обработка всех исключений в Symbian C++ происходила при помощи сбросов и ловушек, поэтому умение рационально и эффективно использовать сбросы, ловушки и стек очистки было обязательным предварительным условием для полноценной работы с платформой. Код, написанный на «чистом» C++, в Symbian OS компилировался только при отсутствии в нем стандартных исключений.

В то время как разница в быстродействии между изначальным и стандартным способами обработки исключений была существенно уменьшена благодаря унифицированной реализации (начиная с Symbian OS версии 9, механизмы сброса и ловушек выполнены на основе стандартных операторов C++ `catch` и `throw`), изначальный способ обработки исключений по-прежнему дает разработчикам ряд преимуществ при создании C++ кода под Symbian OS. В частности, явное использование фреймворка очистки (требуемое при работе с изначальными исключениями Symbian OS) заставляют разработчика создавать весьма дисциплинированный и сфокусированный код. Как заявил Джон Пагонис (John Pagonis), оригинальный фреймворк «всегда перед глазами» (Pagonis, 2006).

Харрисон так же подчеркнул (Harrison and Shackman, 2007, стр. 70), что очистка является фундаментальным аспектом программирования под Symbian OS: «На каждую строку кода которую вы пишите или читаете, будет действовать мысль об очистке». Данная особенность – одна из самых важных для написания стабильного кода под Symbian OS, и программисты должны знать и уметь пользоваться ею для безопасного инициализирования объектов с помощью стека очистки.

Так же стоит напомнить причину, по которой изначальный способ обработки исключений был, во-первых, столь важен, а во-вторых, столь тщательно интегрирован в систему (к примеру, класс `CBase` содержит в себе встроенную поддержку стека очистки). Symbian OS нацелена на очень специализированный сегмент устройств: мобильные устройства для коммуникаций, и мобильные телефоны в частности. В таких устройствах вероятность неудачного распределения ресурсов гораздо выше, чем в больших системах, на которые первоначально был ориентирован язык C++ (такие например, как телефонные узлы).

3.1 Использование сбросов и ловушек

Существует множество хороших руководств по программированию в Symbian OS, описывающих как использовать сбросы и ловушки (посмотрите на ссылки в конце этой главы). Как суммирует Стичбри (Stichbury, 2004, стр.15):

- `User::Leave()` аналогичен оператору `throw` в C++
- Макросы-ловушки `TRAP` и `TRAPD` аналогичны комбинации операторов `try` и `catch` (так же как и комбинация `try-catch` примерно эквивалентна функциям `setjmp` и `longjmp`).

Макросы-ловушки `TRAP` и `TRAPD` окружают вызовы функций, и таким образом позволяют отловить сбросы, возникающие в окружаемых ими функциях:

```
TRAPD( error, MyLeavingMethodL() );

if ( KErrNone == error )
    {
    // все прошло хорошо, никакого сброса не возникло
    }
```

```
else
    {
    // возник сброс, и его необходимо обработать
    // возможно, информацию о сбросе следует передать вышестоящим компонентам
    }
```

Макрос TRAPD является всего лишь более удобной версией макроса TRAP, который сам создает и инициализирует переменную error.

Если вызываемый в макросе метод MyLeavingMethodL(), или любой другой метод, вызовет сброс, код возникшей ошибки будет сохранен в переменной error, а ход исполнения программы будет возвращен макросу-ловушке. Код, располагающийся после макроса, сможет затем изучить код возвращенной ошибки, и соответствующим образом обработать возникшее исключение.

Сброс вызывается в функции MyLeavingMethodL() при помощи специальных методов из библиотеки пользователя EUser. Эти методы имеют название User::Leave(); так же существуют еще три других метода, в названии которых указаны условия возникновения сброса: User::LeaveIfError(), User::LeaveNoMemory() и User::LeaveIfNull(). В частности, метод User::LeaveIfError() используется для преобразования функций, возвращающих коды ошибок (например, методы файлового сервера), в функции, запускающие сброс.

Пользовательская библиотека так же содержит перегруженный оператор new, вызывающий сброс в случае неудачи при выделении памяти под объект:

```
// Создать новый экземпляр класса MyObject
MyObject* obj = new (ELeave) MyObject;
```

В данном случае перегруженный оператор new, вызовет сброс, если под новый объект не удастся выделить достаточно памяти. Следует так же обратить свое внимание на следующие нюансы:

- При программировании на Symbian C++ очень важно использовать окончание «L» в названиях функций, способных вызвать сброс, как, например, метод MyLeavingMethodL(). Весь системный код Symbian OS использует это правило, и вашему коду следует так же его использовать, чтобы разработчик, пользующийся вашим методом, смог сразу же понять, что ваша функция может вызвать сброс.
- Сброс может быть вызван либо кодом, содержащимся в каком-либо методе, либо явно при помощи специальных функций из библиотеки пользователя.
- Когда метод вызывает сброс, вместе с информацией о сбросе так же передается и код возникшей ошибки, поэтому нет необходимости возвращать код ошибки внутри функции. Обычно метод, способный вызвать сброс, должен возвращать тип void, а все ошибки, возникающие в этом методе, должны передаваться наружу посредством сбросов.
- Возникновение сброса приводит к отмене нормального хода программы, и выполнение передается тому участку кода, где сброс впервые ловится при помощи макроса-ловушки.

Следует заметить, что не нужно ловить каждый сброс. В некоторых случаях сброс можно передать системе, которая и обработает возникшую критическую ситуацию (обычно обработка выражается в виде паники и прекращения работы программы). Слишком обильное использование макросов-ловушек может привести к нежелательному увеличению объема кода программы, когда на стадии компиляции вызовы макросов-ловушек будут заменяться их настоящим кодом.

3.1.1 Выводы

Сделаем выводы:

- Метод является «сбрасывающим», если он способен вызвать сброс, например, при помощи прямого вызова метода User::Leave(), либо косвенно через вызов той или иной функции, включая и метод NewL() – перегруженный оператор new.
- Возникновение сброса прекращает работу как самой функции, так и работу любого

другого метода, вызвавшего ее (прямо или косвенно). При этом информация о сбросе будет передаваться каждому из методов, находящихся в стеке вызова функций до тех пор, пока сброс не будет «пойман» в какой-нибудь функции макросом-ловушкой (Harrison, 2003, стр. 138).

- Сбрасывающие методы не должны возвращать коды ошибок, так как коды возникающих ошибок копируются в информацию о сбросе.
- Функции, возвращающие только коды ошибок, можно легко трансформировать в сбрасывающие методы. Для этого в тело функции нужно вставить вызов соответствующего сбрасывающего метода, например, `User::LeaveIfError()`.
- Сбросы в основном используются для методов, чье успешное выполнение не может быть стопроцентно гарантированно, а так же для любых операций, связанных как с прямой, так и с косвенной работой над ресурсами. Как поясняет Харрисон (Harrison, 2003), ситуации возникновения проблем с ресурсами, в которых нуждается ваша программа, должны интерпретироваться как системные ошибки, и на эти ситуации ваша программа должна реагировать при помощи сбросов.
- Если сброс не будет пойман макросом-ловушкой, система передаст информацию о нем вышестоящим методам.
- Не каждый сброс должен ловиться. На самом деле, поимка и обработка сброса – скорее исключение, чем правило, поэтому позволяйте сбросам дойти до определенного уровня, прежде чем обработать их.
- Если макросы `TRAPD` оказываются вложенными, то переменная, хранящая возвращаемый код ошибки, начинает использоваться в них заново, что естественно приводит к некорректным значениям возвращаемых кодов ошибки. Поэтому лучше всего использовать макросы `TRAP` (Stichbury, 2004, стр. 21).
- Макросы-ловушки в основном используются для поимки и обработки возникающих ошибок без дальнейшей их передачи в вышестоящие методы, как, например, в методе `Draw()` (который не должен никогда сбрасывать). Иными словами, пойманный вами сброс должен быть обработан, а не передан выше (Harrison, 2003, стр.153).
- Механизмы очистки могут быть использованы для поимки и обработки ошибок ввода, например, неправильного ввода в пользовательских диалогах (Harrison, 2003, стр.172). Данный прием будет описан в следующих главах.

3.2 Очистка после сброса

Когда возникает сброс, выполнение кода передается первому макросу-ловушке, где этот возникший сброс ловится и обрабатывается. Передача информации о сбросе идет от функции к функции, данные о которых сохранены в стеке вызова. Все текущее окружение исполняемого кода уничтожается во время возникновения сброса, и заменяется на окружение, связанное макросом-ловушкой. Таким образом, сброс уничтожает стековое окружение той функции, где он возник, и восстанавливает стековое окружение той функции, в которой расположен ближайший макрос-ловушка.

На деле все это приводит к тому, что любые объекты, размещенные на «куче» до возникновения сброса, и любые автоматические переменные-указатели, указывающие на такие объекты, просто теряются. При этом в случае возникновения каких-либо непредвиденных ситуаций, стандартные механизмы Си++ рассчитаны только на уничтожение локальных переменных, а объекты, размещенные на «куче», так и остаются нетронутыми. При таком ходе дел очень вероятно возникновение утечек памяти.

В Symbian OS существует особый «стек очистки», благодаря которому разработчики могут «пометить» объекты, размещаемые на «куче», в качестве кандидатов на автоматическое удаление в случае возникновения сброса. В свою очередь, механизм сброса, использующийся в Symbian OS, при возникновении сброса первым делом приведет стек очистки к его первоначальному состоянию, уничтожив все объекты, помещенные в него, начиная от места вызова макроса-ловушки до кода, вызвавшего сброс.

Таким образом все объекты, создаваемые внутри потенциально опасной функции, должны помещаться в стек очистки, а после удачно выполненных операций – уничтожаться, попутно извлекаясь из стека очистки. В случае возникновения сброса, стек очистки сам автоматически удалит помещенные в него объекты при помощи вызова их деструкторов.

Поэтому к вышеприведенному примеру кода, мы можем теперь добавить и вызов функций

стека очистки:

```
// помещаем указатель на "something" в стек очистки
CleanupStack::PushL( something );

// в данном примере мы не используем макрос-ловушку
MyLeavingMethodL();

// все завершилось хорошо, и метод MyLeavingMethodL() не вызвал сброс
// теперь мы можем извлечь из стека очистки указатель на "something"
CleanupStack::Pop( something );
```

В данном примере приведены лишь пара методов стека очистки, в то время на самом деле он имеет еще множество других, включая и метод `CleanupStack::PopAndDestroy()`, благодаря которому объект, помещенный в стек очистки, не только извлекается из него, но еще и удаляется из памяти мобильного устройства. Опять же, существует множество хороших руководств по использованию стека очистки в Symbian OS. Обратите внимание на ссылки в конце этой статьи.

3.2.1 Выводы

Эффективное и правильное использование стека очистки представляет собою значительную долю компетентности разработчика Symbian OS. Вам так же стоит помнить о следующих вещах:

- Стек очистки всегда резервирует для себя последнюю свободную ячейку памяти, чтобы операция помещения в него какого-нибудь объекта никогда не приводила к сбросу. За более детальной информацией о работе и устройстве стека очистки, следует обратиться к (Harrison 2003, стр.14) или (Stichbury 2004, стр.35).
- Класс `CBase` включает в себя виртуальный деструктор Си++, благодаря которому любой другой класс, унаследованный от `CBase`, может быть удален стеком очистки.
- Каждый поток в Symbian OS имеет свой стек очистки. Для обычных пользовательских программ, стек очистки автоматически создается системными модулями UI, однако если внутри пользовательской программы будут создаваться дополнительные потоки, для каждого из них придется вручную создавать отдельные стеки очистки. Консольным приложениям так же придется создавать свой стек очистки вручную (Harrison 2003, стр.11).
- Обычный способ использования стека очистки заключается в помещении в него объекта, размещенного на «куче»; выполнении потенциально опасных операций; и извлечении объекта из стека очистки после удачного завершения всех работ. В случае возникновения сброса, все объекты, помещенные в стек очистки, будут автоматически извлечены из него и удалены. В стеке очистки имеется набор методов для помещения и извлечения объектов, как, например, метод `PopAndDestroy()`, который извлекает объект из стека, и вызывает его деструктор.
- Всегда ставьте букву «L» в конце имен функций, способных вызвать сброс. Таким образом вы предупредите других разработчиков о возможных сбросах в ваших методах.
- Используйте инструмент «LeaveScan» (установленный в вашем SDK в директории `\epoc32\tools`), для проверки ваших исходников на предмет наличия необъявленных L-методов, содержащих другие сбрасывающие методы.

4 Стандартные исключения Си++ в Symbian OS версии 9

В сравнении со стандартным механизмом исключений в Си++, механизм сбросов в Symbian OS ставил перед собой прежде всего задачу легкости и компактности его реализации (Stichbury 2004, стр. 27). При этом механизмом сбросов выполнялась та же самая работа по передаче информации об исключительной ситуации с места, где она возникла, к месту, где она могла бы быть обработана. Начиная с девятой версии, Symbian OS начала так же поддержку стандартных исключений Си++. В действительности же все многие части Symbian OS были переписаны для возможности использования обоих механизмов обработки исключительных ситуаций. И в то время, как оригинальный механизм сбросов и ловушек все еще присутствует в системе (в целях совместимости со старым кодом), его действительная реализация основывается уже на стандартных операторах Си++: `catch` и `throw`. За более детальными разъяснениями стоит обратиться к небольшой статье Джейсона Морли (Jason

Morley 2007).

4.1 Использование операторов try/catch/throw

Теперь, вместо сбросов и макросов-ловушек можно так же использовать стандартные операторы Си++ `try`, `catch` и `throw`:

```
// блок кода
{
    ...
    // блок try
    try {
        // выполнить какую-нибудь потенциально опасную операцию,
        // например, попытаться выделить память под объект,
        // или предпринять какие-нибудь действия с файлами или с сетью
    }
    catch ( exception ) {
        // обработать исключение
        return failure;
    }
    ...
    return success;
}
```

Блок «`try`» содержит любой потенциально опасный код, способный привести к возникновению исключения. Исключения, в свою очередь, являются типизированными объектами (Stroustrup 1994, стр. 387). Блок «`catch`» отлавливает исключения указанного типа, и выполняет соответствующие операции по их обработке. Само исключение возникает в коде благодаря вызову оператора `throw`:

```
// что-то не получилось, сообщаем об исключении
throw ( exception );
```

Выполнение кода прерывается в момент вызова оператора `throw`, а объект исключения `exception` передается в блок `catch`. За более детальными разъяснениями лучше всего обратиться к хорошему примеру программирования на Си++.

Механизм стандартных исключений Си++ не использует стек очистки, поэтому разработчики должны сами проверять свой код на наличие возможных утечек памяти при возникновении исключений, как, например, при помощи использования «умных указателей», описанных в (Meyers 1996). Во время портирования стандартного кода Си++ в Symbian OS, такие проверки являются одними из самых необходимых, и помогают убедиться в корректном поведении кода на платформе Symbian.

5 Смешивание сбросов и стандартных исключений Си++

Так как начиная с девятой версии Symbian OS механизмы сброса и макросов-ловушек реализованы на основе стандартного механизма исключений Си++, у разработчиков есть возможность смешивать в коде приложений сбросы и исключения, хотя такую практику и не рекомендуют использовать в теле одного и того же бинарного модуля.

Для смешивания сбросов и исключений, рекомендуется следовать следующим правилам:

1. «Сбрасывающие» методы не должны выбрасывать исключений. По крайней мере, если они сами их не ловят и не обрабатывают их внутри себя. Макрос-ловушка, используемая в Symbian OS, может поймать только определенный тип исключений, а не все классы стандартного исключения Си++. Все исключения, не поддерживаемые макросом-ловушкой, должны быть пойманы еще до того, как макросу будет передано выполнение кода. Так как разработчики могут использовать в коде как исключения, так и сбросы, все же нельзя смешивать контексты этих механизмов между собой: вы не можете вызвать сброс, а потом ловить исключение; как и не можете выбросить сначала исключение, а потом ловить сброс.
2. Код, выбрасывающий исключения, не должен пользоваться стеком очистки, и не должен зависеть от любого другого кода, который в своей работе использует стек

очистки. Это условие должно выполняться из-за отсутствия возможностей у стека очистки удалить помещенные в него объекты в случае возникновения исключений Си++.
3. «Сбрасывающий» код никогда не должен вызывать код, способный выбросить исключение.
4. В то же время код, способный выбрасывать исключения (например, любой код, портированный на платформу Symbian OS), может свободно пользоваться кодом, способным вызвать сброс (системный код Symbian OS).

Деструкторы объектов никогда не должны вызывать сбросы, и выбрасывать исключений. Об этом читайте в нижележащей главе.

6 Избегайте вложенных исключений

Реализация механизма стандартных исключений Си++ в Symbian OS всегда резервирует достаточно памяти, чтобы всегда имелась возможность разместить в памяти мобильного устройства хотя бы один объект исключения, даже когда у мобильного устройства не окажется никакой свободной памяти. Однако вложенные исключения требуют гораздо большее места в памяти, чем это обычно требуется. В ситуациях нехватки памяти (out-of-memory, или OOM), операционная система не может гарантировать места больше чем для одного объекта исключения.

По этой причине Symbian OS не позволяет использовать вложенные исключения Си++. И хотя код с вложенными исключениями будет работать на эмуляторе, тот же самый код в бинарной форме для ARM-устройства приведет к системному отказу.

Вложенные исключения могут так же возникать в тех случаях, когда во время обработки какого-либо исключения возникает другое исключение. Например, когда деструктор класса, вызванный во время обработки исключения, сам приводит к возникновению нового исключения. Поэтому любые методы и деструкторы, вызываемые в блоке обработки исключения, никогда не должны вызывать новых исключений.

Так как теперь сбросы в Symbian OS реализованы при помощи вызова стандартного оператора Си++ `throw`, те же самые ограничения теперь накладываются и на вложенные сбросы, хоть в действительности эти ограничения не всегда имеют силу. За дополнительными деталями обратитесь к статье (Morley 2007).

6.1 Выбирайте правильную стратегию

Выбор между привычными сбросами Symbian OS, и стандартными исключениями Си++, лучше всего делать в зависимости от природы вашего кода, когда вы будете создавать либо совершенно новый код, либо портировать на платформу уже существующий.

Какой бы выбор вы ни сделали, необходимо всегда следовать трем главным правилам обработки ошибок:

1. Всегда будьте готовы обрабатывать ошибки.
2. Обрабатывайте ошибки без потери данных.
3. Обрабатывайте ошибки без утечек памяти или ресурсов.

6.2 Портирование кода в Symbian OS

Переписывание уже существующего кода, использующего в своей работе стандартные исключения Си++, с целью встраивания в него сбросов Symbian OS, на самом деле задача не только трудоемкая, но еще и чреватая ошибками и совершенно бессмысленная. Однако все же стоит перепроверить код на наличие вложенных исключений (см. главы выше), и иметь в виду, что портируемый код не использует стек очистки, и поэтому может вызвать утечки памяти. Для борьбы с утечками памяти, возможно пригодятся «умные указатели». Опять же, важно помнить, что код, написанный для мобильных устройств, отличается от кода, созданного для обычных компьютеров, и поэтому следует уделить достаточно времени и сил, чтобы сделать портируемый код надежным и эффективным в мобильных устройствах.

6.3 Создание кода для нескольких платформ

Когда вы начнете создавать Си++ программу для нескольких платформ сразу, используйте стандартные исключения Си++.

6.4 Создание кода специально для Symbian OS

Имеющиеся в Symbian OS механизмы сброса и ловушек достаточно быстры, легки и эффективны. Область действия этих механизмов было очень хорошо определена, проверена на практике, и значительно оптимизирована в девятой версии Symbian OS. Поэтому использование сбросов и ловушек рекомендовано везде, где это только возможно. В частности, сбросы и ловушки используются во всех API операционной системы и ее модулях. Механизмы сбросов и ловушек создавались специально для мобильных устройств с целью обработки ситуаций нехватки памяти и ресурсов. При этом использование стека очистки является более легким решением для управления динамической памятью, чем использование стандартных исключений Си++, и уменьшает риск возникновения утечек памяти.

Все эти преимущества берут верх над тем обстоятельством, что механизмы сброса и ловушек не являются стандартными, и поэтому подразумевают собою значительные трудности для осваивания новыми разработчиками.

7 Вспомогательные механизмы

7.1 Проверки и паники

Проверки представляют собой фатальные ошибки, информация о которых никуда не передается, но которые останавливают программу на месте своего возникновения. Symbian OS предлагает разработчикам следующие макросы проверок: `__ASSERT_DEBUG`, `ASSERT` и `__ASSERT_ALWAYS`. Проверки – один из самых эффективных способов поиска ошибок программирования. Они заставляют программу прекратить свою работу на месте обнаружения какой-либо ошибки. Обычно проверки используются для обнаружения несоответствия начальных и конечных условий выполнения какого-либо кода, а так же нахождения «невозможных» значений переменных (Hanson, стр. 59). Когда программа начинает свою работу, макросы проверок останавливают ее выполнение в случае обнаружения несоответствия какого-либо условия заданному, и выводят на экран сообщение об этом.

Как правило, после обнаружения такого несоответствия разработчик должен разобраться в коде, и исправить найденную ошибку.

Проверки реализуются в системе в виде паник. Паники создаются с соответствующей категорией и кодом. Категория определяет подсистему Symbian OS, в которой ошибка была обнаружена, а код паники определяет причину ее возникновения. Исключение, оставленное в программе без обработки, так же будет интерпретировано системой как паника. Более детальную информацию о категориях паник и их кодах вы можете обнаружить в соответствующем разделе библиотеки разработчика Symbian. Symbian OS так же предлагает разработчикам возможность сохранять информацию о возникающих паниках в системном логе. За более детальными разъяснениями обратитесь к библиотеке разработчика Symbian.

8 Цель адаптации исключений Си++

Работа над предшественницей Symbian OS началась за несколько лет до появления ISO стандарта языка Си++. Чистый язык Си++ имел неструктурированные методы `setjmp` и `longjmp` (использующиеся соответственно для определения обработчика и запуска исключения), которые на самом деле работали как оператор `goto`. Любая дополнительная поддержка исключений и их обработки, и даже реализация методов `setjmp` и `longjmp`, на деле оказывались несовместимыми между различными производителями компиляторов. Даже когда стандарт Си++ начал свое широкое распространение, его поддержка различными производителями была совершенно различной. Так, например, еще в середине 2003-го года Эрик Рэймонд сетовал на то, что еще ни один компилятор не поддерживал ISO стандарт языка Си++ (ISO C++99) в полной мере, хотя компилятор GCC C++ в этом смысле преуспевал больше других (Raymond 2004, стр. 410).

Стэнли Липман провел интересное исследование компиляторов, сравнивая размер кода, и скорость его работы при использовании исключений (даже когда эти исключения не были совсем активизированы в коде). Он так же рассказал несколько интересных анекдотов на эту тему. Например, дальнейшая разработка изначальной реализации интерпретатора Си++, cfront, генерирующей на выходе код Си, была остановлена по причинам невозможности реализации универсальных и переносимых механизмов обработки исключений. А шахматная программа Junior за ночь до соревнования была перекомпилирована при помощи нового компилятора Си++, поддерживающего исключения, однако она столь много выросла в размерах, что перестала умещаться на компьютере. К счастью, заново скомпилированная на старом компиляторе, она добилась успеха вместе с суперкомпьютером IBM Deep Blue.

Когда стандартные исключения Си++ были включены в Symbian OS, размер бинарных модулей системы в ROM памяти сразу же подскочил на десять процентов. Хотя данное обстоятельство и не кажется таким уж непомерным для мобильных телефонов нынешнего поколения, все же цена затрачиваемых материалов, и издержки производительности по прежнему остаются критичными факторами на рынке мобильных устройств. Поэтому было потрачено немало усилий для возвращения размеров бинарных модулей в их изначальные размеры.

Поддержка стандартных исключений Си++ в Symbian OS была прежде всего обусловлена желанием сделать платформу более доступной для разработчиков с обычными навыками Си++, а не специфическими знаниями Symbian OS. А также желанием сделать платформу более открытой для большого количества уже существующих программ, написанных на стандартном Си++.

Все эти перечисленные цели имеют значение на корпоративном рынке, с его большим количеством собственного программного обеспечения самих компаний, а так же на рынке мобильных операторов, стремящихся развернуть свои мобильные платформы на всех мобильных устройств, находящихся в сети. Однако те же самые цели имеют свой вес и на рынке пользовательского программного обеспечения. Стандартизация берет свой верх.

9 Дополнительные материалы

Книга Джо Стичбри (Stichbury 2004, глава 2) дает лучшие примеры использования механизма сбросов и ловушек в Symbian OS, а так же дает всю необходимую информацию о стеке очистки.

Касательно темы портирования кода на ранние версии Symbian OS, обратитесь к статье о переносе скриптового языка Simkin на платформу Symbian, находящейся в архивах сети разработчиков Symbian:
developer.symbian.com/main/downloads/papers/porting_simkin/Porting_Simkin.pdf.

Техническая статья Джейсона Морли (Jason Morley, 2007) представляет собой прекрасный материал о реализации новых механизмов обработки исключений в Symbian OS.

Многие официальные форумы разработчиков Symbian (принадлежащих Nokia, UIQ, и Sony Ericsson), содержат множество полезной информации касательно обработки исключений в девятой версии Symbian OS.

С точки зрения стандартного Си++, некоторые детали обработки исключений описаны в книге Стэнли Липмана «*Inside the C++ Object Model*» (Stanley Lippman, 1996, глава 7). Авторы Коэнинг и Му (Koenig and Moo, 1996, стр. 284) так же рассказывают о том, что происходит в случае возникновения проблем у оператора `new`.

«Умные указатели» описаны в книге Майерса (Meyers 1996). Для адаптации этой концепции в Symbian OS, обратитесь к статье Сандера ванн дер Вала (Sander van der Wal, 2002).

Если вы хотите услышать общие рассуждения об ошибках и исключениях, то статья Кернинга и Пайка (Kernighan and Pike 1999) великолепно умещает в себе все необходимое.

За весьма интересными рассуждениями о пользе проверок начальных и конечных условий в коде, обратитесь к статье Джезике и Майера (Jezequel and Meyer, 1997), размещенной на сайте языка Eiffel, и рассказывающей о крушении ракеты «Ариан».

Статья Райдера и Софа (Ryder and Soffa, 2003), повествует интересные исторические факты о языках программирования и ислючениях.

9.1 Ссылки

Hanson, David R. (1997) *C Interfaces and Implementations*, Addison-Wesley

Harrison, Richard. (2003) *Symbian C++ for Mobile Phones Volume 1*, John Wiley/Symbian Press

Harrison, Richard and Shackman, Mark. (2007) *Symbian C++ for Mobile Phones Volume 3*, JohnWiley/Symbian Press

Jezequel and Meyer. (1997) *Design by Contract: The Lessons of Ariane*

Kernighan, Brian and Pike, Robert. (1999) *The Practice of Programming,* Addison-Wesley

Koenig, Andrew and Moo, Barbara. (1996) Andrew Koenig and Barbara Moo, *Ruminations on C++,* Addison-Wesley

Lippman, Stanley B. (1996) *Inside the C++ Object Model,* Addison-Wesley

Meyers, Scott. (1996) *More Effective C++,* Addison-Wesley

Morley, Jason. (1997) *Leaves and Exceptions,* at developer.symbian.com/main/downloads/papers/Leaves%20and%20Exceptions.pdf

Pagonis, John. (2006) Presentation, Wireless Developer Forum, Cambridge UK, December 2006

Raymond, Eric S. (2004) Eric S Raymond, *The Art of Unix Programming,* Addison-Wesley

Ryder, Barbara G. and Soffa, Mary Lou. (2003) *Influences on the Design of Exception Handling,* SIG, www.sigsoft.org/impact/docs/p29-ryder.pdf

Stichbury, Jo. (2004) *Symbian OS Explained,* John Wiley/Symbian Press

Stroustrup, Bjarne. (1994) *The Design and Evolution of C++,* Addison-Wesley

van der Wal, Sander. (2002) *Creating the C++ auto_ptr<> utility for Symbian OS*, at developer.symbian.com/main/downloads/papers/auto_ptr/auto_ptr.pdf

9.2 Веб-ресурсы

Сеть разработчиков платформы Symbian, The Symbian Developer Network, или SDN, можно найти по адресу developer.symbian.com.

10 Об авторе

Бен Моррис является автором многих статей и архитектором, специализирующимся на Symbian OS. Он так же является автором книги «*The Symbian OS Architecture Sourcebook: Design and Evolution of a Mobile Phone OS*», опубликованной издательством Symbian Press. Найти его контакты вы можете на сайте www.wordmatter.co.uk.