УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЦЕССОМ

Е. И. Клименков

Белорусский университет информатики и радиоэлектроники,
кафедра программного обеспечения информационных технологий
ул. Гикало, 9, г. Минск, Республика Беларусь
телефон(ы): + (37533) 6045147; e-mail: ZarathustrA@np.by

В данном докладе подробно рассматривается управление программным процессом. Определяется универсальность функционала управления и его сквозная природа.

Ключевые слова — управление, программный процесс, сквозной функционал.

1 УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЦЕССОМ

Как известно, любая система требует управления. Наиболее четкое выражение этот тезис нашел в технических системах. Именно в области управления технических систем на стыке кибернетики и теории информации возникла наука, получившая название "теория управления", в которой предметом исследования являются принципы и методы управления различными процессами системами и объектами. Сфера программных систем, как и любых других технических систем, не минула проблем связанных с управлением. С ростом объема кода и сложности программных систем все более очевидным становиться тот факт, что они требуют использования специализированного органа управления взамен традиционного распыления функции управления по всей системе.

Под управлением чаще всего понимают организацию и координацию деятельности объектов в соответствии с определенной политикой для достижения четко определенных целей. При этом функцию управления можно разделить на несколько взаимосвязанных подфункций:

1. Планирование
2. Организацию
3. Обеспечение ресурсами
4. Руководство
5. Контроль

Под программным процессом мы будем понимать самостоятельный элемент программной системы, реализуемый отдельным исполнимым файлом программной системы и выполняющим определенную функцию в программной системе. Соответственно, под управлением программным процессом мы будем понимать как управление процессом в системе, так и управление компонентами в программном процессе.

При соотнесении вышеперечисленных подфункций с областью управления программными процессами мы обнаруживаем следующую специфику. Планирование, организация и обеспечение ресурсами программного процесса выполняются статически самим текстом программы. Именно текстом программы определяется порядок действий производимых программным процессом, организация его компонентов и естественно получение всех необходимых ему ресурсов. Соответственно, под управлением программным процессом мы можем понимать только руководство и контроль.

Прослеживается также и тот факт, что функция управления представляет собой подавляющую часть сквозной функциональности программного процесса. Напомним, что под сквозной функциональностью понимают функциональность, которую невозможно выделить в отдельные сущности средствами ООП, так как ее реализация рассыпана по различным модулям программы. Сквозная функциональность приводит к рассредоточенному и запутанному коду, сложному для понимания и сопровождения. Ведение лога и обработка ошибок - типичные примеры сквозной функциональности. Другими примерами являются трассировка, авторизация и проверка прав доступа, контрактное программирование. Следует отметить, что проблема существования сквозного функционала является достаточно серьезной, и уже привела к появлению такой парадигмы программирования как Аспектно-Ориентированное Программирование (АОП) и целой плеяде языков программирования и их диалектов, поддерживающих эту парадигму [1].

Практический опыт управления программными процессами позволил нам взглянуть на управление программными процессами с другой стороны. С точки зрения программиста проектирующего, реализующего и поддерживающего программную систему, состоящую из одного и более программного процесса, каждый из которых представляет собой сложноорганизованную многослойную систему программных модулей, в функцию управления программным процессом укладываются следующие подфункции:

1. Управление динамикой протекания программного процесса
2. Реализация монопольной работы процесса

3. Ведение логов
4. Ведение прогресса
5. Сообщение с управляющим процессом
6. Лицензирование
7. Реакция на критические системные события
8. Гибкое управление параметрами программного процесса

Данное разбиение на функции является достаточно общим, учитывающим только те функции управления, которые реализуются в большинстве программных продуктов. Все подфункции управления программного процесса переплетены друг с другом и тесно взаимосвязаны. Так подфункция "сообщение с управляющим процессом" может перекрываться с другими подфункциями. Например, лог об ошибках может вестись как традиционно в файл, так и дублироваться передачей лога в управляющую инстанцию, в роли которой, в большинстве случаев, выступает графический пользовательский интерфейс.

2 УПРАВЛЕНИЕ ПАРАМЕТРАМИ ПРОГРАММНОГО ПРОЦЕССА

Большое количество ПО прочно обосновалось в семье оконных приложений базирующихся на каком-то виде графического пользовательского интерфейса, в эту семью входят наиболее сложное программное обеспечение, всевозможные текстовые процессоры, среды разработки и т.д. В основном это прикладное ПО, для которого преимущества, предоставляемые дружелюбным пользовательским интерфейсом является существенным, если не определяющим. Но остается большим, и продолжает расти и развиваться парк консольных приложений. Это Мекка системного ПО, для которого более существенным является возможность объединять работу разных утилит в одно задание посредством написания программы на скриптовом языке. Такой подход позволяет автоматически производить сложные отнимающие много времени манипуляции с системой автоматически, тем самым, экономя большое количество времени IT-специалиста и обладает непревзойденной гибкостью, позволяя объединять в решении одной задачи разное ПО написанное различными компаниями.

С каждым годом происходит рост объема кода и сложности ПО. Этот тезис относится как к оконным, так и к консольным приложениям. Стремясь вместе с ростом сложности и объема кода в ПО сохранить и гибкость присущую консольным утилитам разработчикам приходится увеличивать количество всевозможных параметров и флагов командной строки, которые и управляют работой утилиты определяя специфику задания для нее. Как на пример, можете взглянуть на интерфейс командной строки любого компилятора или линковщика. Естественно, чем большим количеством разных параметров обладает приложение, тем гибче оно становиться, но наряду с этим возрастает и скрытая от пользователя сложность управления всеми этими параметрами. Ведь их всех нужно принять, где-то хранить и обеспечивать доступ к ним всем заинтересованным в них участкам кода.

Традиционно проблема управления параметрами утилиты решается сочетанием трех методов: использованием глобальных переменных, традиционной передачей параметров через функции и разбором командной строки. Но при увеличением количества параметров задачи, каждый из которых нужно доставить заинтересованным модулям и при постоянном изменении количественного и качественного состава этих параметров такой способ управления параметрам становится неудобным. Когда количество параметров составляет несколько десятков, управление ими становится проблематичным.

Данная проблема может разрешиться при организации в программе базы данных параметров задачи с глобальной точкой доступа, к которой подключаются все заинтересованные в получении параметров задачи модули. База данных храниться локально в базовой функции программы (в C\C++ это функция main()). Глобальная точка доступа является простейшим объектом (например, указателем на объект базы данных), поэтому ее использование как глобальной относительно безопасно за счет того, что проблемы связанные с конструкторами/деструкторами исчезают. Заметим также, что глобальная область видимости программы дополняется только одним именем, а не множеством равным количеству параметров задачи. При этом исчезает проблема передачи параметров заинтересованным в ней модулям через множество функций. Любой заинтересованный в параметре задачи модуль в силах сам запросить его из базы данных параметров задачи без чьей-либо посторонней помощи. Логика разбора параметров унифицируется и становиться универсальной, то есть применимой к подавляющему большинству задач. Классические методы разбора параметров пропадают как явление.

Использование базы данных параметров программного процесса улучшает модульность программы за счет локализации сквозного функционала управления параметрами, минимизации глобальных объектов, упрощения добавления/удаления параметра программного процесса. Данный механизм обладает гибкостью унаследованной от формата командной строки и рассчитан на использование в многопоточных приложениях. Более того, он позволяет отдельным модулям влиять на поведение друг друга, устраняя непосредственные физические связи между самими модулями. Поскольку эпоха консольных приложений еще не скоро завершится, применение данного подхода к управлению программным процессом является актуальным и перспективным.

3 МОНОПОЛЬНАЯ РАБОТА ПРОЦЕССА

В управлении программными процессами есть уже ставшей классической задача по обнаружению одновременной работы более одной копии программы. Эта зада-

ча актуальна не для всех программных средств. Множество программ таких как, например notepad, calc и т.п. прекрасно обходятся без ее решения. Для других они упрощают работу, как например для браузеров, более сложных текстовых редакторов и процессоров, которые поддерживают GUI, в котором каждый документ представляется вкладкой, в одном приложении, вместо представления множества документов в множестве копий приложения. Для многих же средство обнаружения одновременной работы нескольких копий программного средства является важнейшим средством синхронизации и обеспечения целостности данных. В конечном итоге все зависит от специфики предметной области программного средства и его архитектуры.

4 УПРАВЛЕНИЕ ДИНАМИКОЙ ПРОТЕКАНИЯ ПРОГРАММНОГО ПРОЦЕССА

Протекание программного процесса является динамичным. То есть процесс может протекать с какой-то определенной скоростью или менять скорость своей работы в зависимости от каких-либо факторов. Процесс может быть заморожен или вовсе остановлен. Зачастую при управлении программным процессом сложного ПО возникает необходимость в управлении динамикой его протекания. Например, для повышения приоритета процесса на критических участках кода или же для сохранения целостности данных при останове процесса. Актуальным также является возможность заморозки процесса при отсутствии каких-нибудь необходимых ресурсов для работы процесса, чтобы пользователь смог освободить или создать ресурс и затем продолжить работу процесса.

В целом все возможности управления динамикой процесса можно разделить на две группы:

1. Останов процесса.
2. Замораживание процесса (постановка на паузу).

Останов процесса может быть мягкий и жесткий. Последний представляет собой принудительное завершение всех потов процесса. Но при управлении программным средством у нас может возникать упоминаемая ранее задача сохранения целостности обрабатываемых данных или любая другая задача связанная с выходом из приложения. В таком случае можно вспомнить про концепцию событийно-ориентированного программирования и попытаться реализовать обработку события завершения работы процесса специальной функцией-обработчиком этого события. Но этот подход является ограниченным по гибкости, так как в большом количестве случаев невозможен универсальный алгоритм корректного завершения программы, и перечень выполняемых при завершении приложения действий напрямую зависит от той точки в программе при исполнении, которой было получена команда на останов.

Более гибкий вариант предполагает, что программный процесс может находиться в определенном состоянии: нормальная работа, состояние останова или состояние паузы. При этом любой модуль может изменять состояние процесса и узнавать его текущее состояние для изменения своего поведения. Таким образом, каждый модуль в наиболее удобные для себя моменты времени может узнавать текущее состояние процесса, и, в случае обнаружения состояния останова, произвести те действия, которые ему необходимы для корректного завершения своей работы. В этом случае, однако, если мы рассчитываем оперативно получать отклик на запрос останова процесса, то следует регулярно в ходе программы узнавать текущее состояние процесса, чтобы быстро завершить работу.

Замораживание процесса – это переключение задачи на другой процесс на уровне операционной системы с исключением всех потоков процесса (за исключением особого управляющего потока) из очереди активных задач. При этом полностью сохраняется контекст потоков, таким образом, что процесс может возобновить свою работу с точно того же места, в котором он был остановлен и в точно том же контексте (в том же адресном пространстве, с тем же стеком и состоянием регистров процессора).

Так же как и останов процесса, замораживание процесса может проходить в жестком и мягком режимах. Мягкий режим в целом аналогичен такому же режиму в останове процессора [2].

Как мы уже говорили выше, состояния останова и паузы могут быть жесткими и мягкими. Таким образом, мы получаем пять состояний, в которых может находиться процесс:

1. Работа
2. Жесткий останов
3. Мягкий останов
4. Жесткая пауза
5. Мягкая пауза

Ниже представлен граф переходов между состояниями процесса.

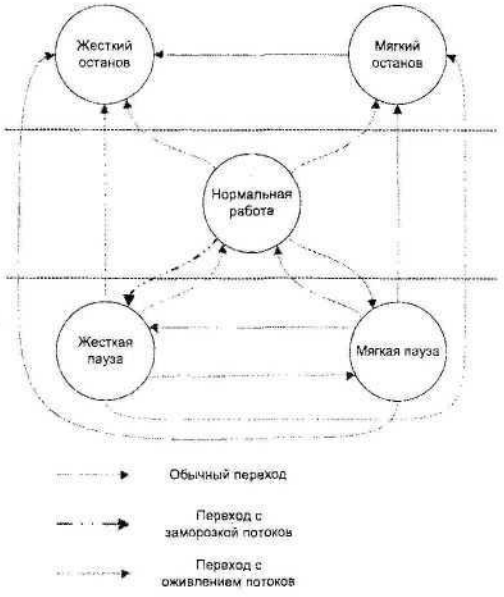

Рис.2. Граф переходов между состояниями процесса.

5 УПРАВЛЕНИЕ ЛИНЦЕНЗИЯМИ В РАМКАХ УПРАВЛЕНИЯ ПРОЦЕССОМ

Проблема технического обеспечения лицензирования программного обеспечения является весьма сложной и актуальной. В связи с этим эта область сейчас активно разрабатывается. И ее полное решение в рамках управления программным процессом является невозможным. Однако полностью игнорировать этот вопрос мы не можем. Консольные утилиты с точки зрения управления лицензиями обладают замечательным свойством статичности параметров задачи. Данное свойство позволяет минимизировать обращения к менеджеру лицензий.

6 УПРАВЛЕНИЕ ПРОГРАММНЫМ ПРОЦЕССОМ И РЕАКЦИЯ НА СИСТЕМНЫЕ СОБЫТИЯ

К сожалению, программное и аппаратное окружение, в котором исполняется программное средство, является весьма сложным и в связи с этим создает определенное воздействие на исполняющуюся программу [2]. И правилом хорошего тона считается реализация адекватного реагирования на эти воздействия, с целями сохранения целостности данных, максимального облегчения взаимодействия пользователя с программой и информирования разработчика о возникающих проблемах.

Все внешние относительно программы события можно разделить по природе происхождения на аппаратные и системные. Аппаратные события по своей сути являются реакциями на попытку нарушения защиты реализуемой процессором или на возникновение ошибочной ситуации при исполнении программы. Еще их называют исключениями (exceptions). С большой долей вероятности программа не сможет нормально продолжать свою работу после появления такой ошибки, поэтому невозможно обеспечить целостность данных, но может быть произведена попытка сохранить данные о самой ошибке, чтобы в дальнейшем облегчить ее локализацию и исправление.

Программные события возникают в рамках системы и чаще всего являются либо извещениями об изменении состояния операционной системы, либо являются представлениями пользовательских команд. И в том и в другом случае они направлены на завершение работы программного средства. При появлении программных событий приложение остается в полностью работоспособном состоянии, поэтому главной задачей в данном случае является сохранение целостности данных приложения и его корректное завершение.

Применение механизма реагирования на события программного и аппаратного окружения в рамках управления программным процессом позволяет повысить устойчивость приложения к низкоуровневым ошибкам, облегчить локализацию и исправление этих ошибок, повысить надежность приложения и сохранить целостность используемых данных, а также повысить юзабилити путем автоматической поддержки еще одного канала общения программного средства с пользователем.

ЛИТЕРАТУРА

[1] Аспектно-ориентированное программирование (АОП): Для чего его лучше использовать? / Г. Полличе, [Электрон, ресурс] // Режим доступа: http://www.ibm.com/developerworks/ru/library/pollice/index.html

[2] Windows для профессионалов: создание эффективных Win32 приложений с учетом специфики 64-разрядной версии Windows / Дж. Рихтер // Русская Редакция – 2001.